HelixSwarm

Helix Swarm Guide

2020.2

December 2020

PERFORCE

www.perforce.com

Contents

How to use this guide

This guide tells you how to use Helix Swarm for collaboration and code review for teams using Helix Core server. It is intended for anyone using Swarm to perform code review tasks with Helix server.

This section provides information on typographical conventions, feedback options, and additional documentation.

Syntax conventions

Helix documentation uses the following syntax conventions to describe command line syntax.

Notation	Meaning
`literal`	Must be used in the command exactly as shown.
italics	A parameter for which you must supply specific information. For example, for a *serverid* parameter, supply the ID of the server.
`-a -b`	Both *a* and *b* are required.
`{-a | -b}`	Either *a* or *b* is required. Omit the curly braces when you compose the command.
`[-a -b]`	Any combination of the enclosed elements is optional. None is also optional. Omit the brackets when you compose the command.
`[-a | -b]`	Any one of the enclosed elements is optional. None is also optional. Omit the brackets when you compose the command.
`...`	Previous argument can be repeated. ■ `p4 [g-opts] streamlog [ -l -L -t -m max ] stream1 ...` means `1` or more stream arguments separated by a space ■ See also the use on `...` in Command alias syntax in the *Helix Core P4 Command Reference* **Tip** `...` has a different meaning for directories. See Wildcards in the *Helix Core P4 Command Reference*.

Feedback

How can we improve this manual? Email us at manual@perforce.com.

Other documentation

See https://www.perforce.com/support/self-service-resources/documentation.

> **Tip**
> You can also search for Support articles in the Perforce Knowledgebase.

Earlier versions of this guide

Earlier versions of this guide: 2020.1, 2019.3, 2019.2, 2019.1

To find even earlier versions of this guide, use the following URL and replace *v17.3* with the version number you are looking for: https://www.perforce.com/manuals/*v17.3*/swarm/index.html

Helix Swarm User Workflow Guide

For a step-by-step walk-through of setting up a workflow, adding it to a project, and using it to progress through a review, see the Helix Swarm User Workflow Guide.

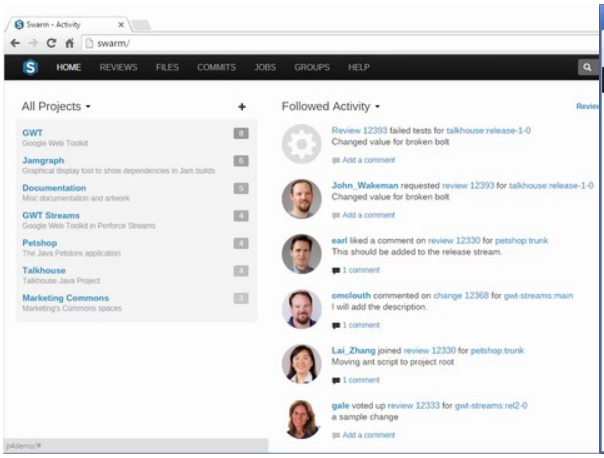
1 | About Helix Swarm

Swarm enables collaboration and code review for teams using Helix Core server, helping ship quality software faster. Swarm is a collaboration tool for all types of intellectual property. It unites teams to foster brainstorming, formalize content reviews, and keep projects moving forward. Contributors share files, comment, suggest tasks, vote up or down, and submit work directly within its elegant, web-based interface. Swarm automates the entire process via notifications and makes it easy to monitor progress.

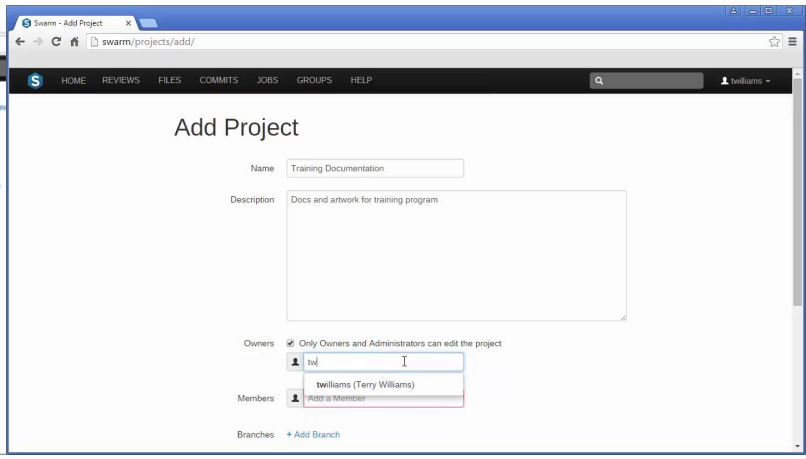
Swarm video tutorials

Use the following Swarm tutorial videos to get an overview of Swarm and to understand the basics of Swarm:

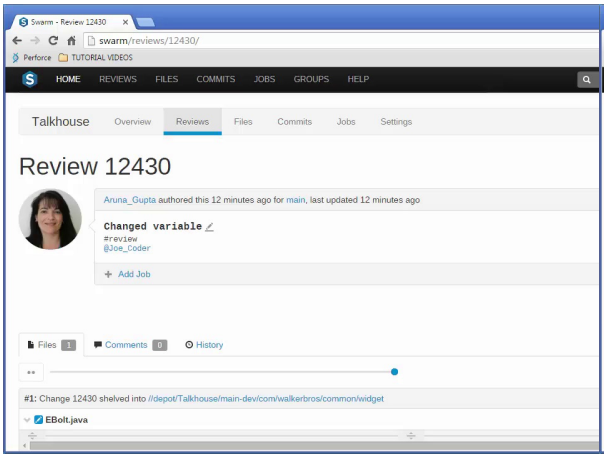
Swarm Overview



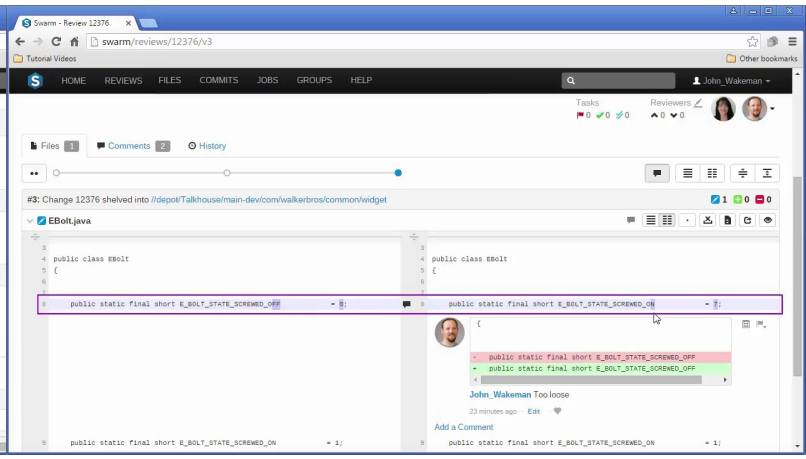
Create a Project in Swarm



Create a Swarm Review



Collaborate on a Review in Swarm



Related topics

Getting started:

- If you are experienced with code review but unfamiliar with Swarm, start with the Quick reference and Quickstart sections.
- If you are unfamiliar with code review, start with the Basics section.
- If you are unfamiliar with the Swarm workflow feature, start with the Workflow overview section.
- For a step-by-step walk-through of setting up a workflow, adding it to a project, and using it to progress through a review, see the Helix Swarm User Workflow Guide.

Users:

- Code reviews
- Projects
- Groups
- Workflows

Administrators:

- Swarm administration
- Swarm Integrations
- Extending Swarm
- Install and configure Swarm
- Update Swarm

What's new

This section provides a summary of the notable changes in Swarm for the 2020.2 release. Full details are available in the distribution's `RELNOTES.txt` file.

Major new functionality

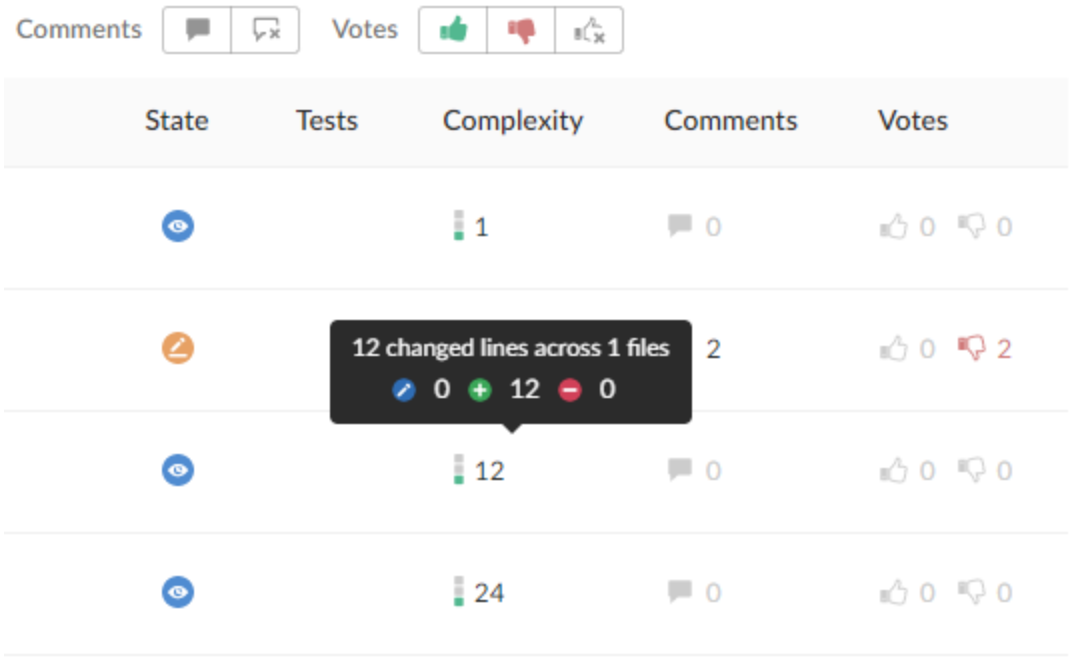
Updated Swarm Reviews list UI

The Swarm Reviews list page UI has been updated to show the information you need more clearly than ever before. New icons make it easier to see the review states and test states for the reviews. Review filtering has also been improved to make it easier to see which filters you have selected. For information on the reviews list, see "Reviews list" on page 342.

See relative review complexity at a glance from your Reviews list

The new traffic light complexity column on the reviews list page shows the relative complexity and the total number of changes for each review at a glance.

Want more detail before opening a review, simply hover over the complexity for the review you are interested in:



For information on review complexity, see "Reviews list" on page 342.

Introducing P4Search for full content search in Swarm - Technology preview feature

Extend standard Swarm searches to search file content and changelist description by using P4Search. The P4Search service needs a connection to Elasticsearch and the Helix Core server. For more information on Helix Core Search API, see the Overview section of the Helix Core Search Developer Guide.

Changes to operating systems supported by Swarm

- We now support the installation of Swarm on Ubuntu 20.04 LTS, CentOS 8, and RHEL 8, see "Recommended operating systems" on page 102.
- Swarm 2020.2 is the final version that will support Swarm installation on Ubuntu 16.04. This is part of our commitment to focus on supported technology platforms.
- We no longer support Swarm installation on CentOS 6 and RHEL 6. This is part of our commitment to focus on supported technology platforms.

Minor new functionality

New API endpoints

A number of new endpoints have been added to the API, see the "Swarm API" on page 664.

Important information

Changes to Swarm support for Ubuntu 16.04

Swarm 2020.2 is the final version of that will support Swarm installation on Ubuntu 16.04. This is part of our commitment to focus on supported technology platforms.

Swarm no longer supports CentOS 6 and RHEL 6

Swarm 2020.2 does not support Swarm installation on CentOS 6 and RHEL 6. This is part of our commitment to focus on supported technology platforms.

Helix Core server 2020.1 stream spec file permissions change

Helix server 2020.1 and later, permissions have changed for viewing and editing stream spec files in Swarm. To view and edit stream spec files in Swarm, the Swarm user must have *admin* permissions for the entire depot `//...`

Trigger script updated for Swarm 2020.1

The **`swarm.shelvedel shelve-del`** trigger script has been updated in Swarm 2020.1, the trigger script must be updated in the Helix server trigger table or added if it does not already exist, see " Update the Helix server triggers table to run the trigger script." on page 164

Custom module support in Swarm 2020.1 and later

The Zend 3 Framework project has been forked to an open source project called Laminas, see https://getlaminas.org/about/. Swarm 2020.1 now uses the Laminas framework, this is part of our commitment to move away from using versions of platforms that have reached End-of-Life (EOL).

If you have existing custom Swarm modules created for Swarm 2019.3 or earlier, you must update them to use the Laminas framework. For instructions on updating your Zend custom modules to Laminas, see "Migrate existing custom modules to the Laminas framework" on page 616.

Upgrading from Swarm 2019.1 and earlier

Swarm 2019.2 introduced a Redis in-memory cache to improve performance and reduce the load on the Helix Core server. This replaces the file-based cache that was previously used by Swarm.

On Swarm systems with a large number of users, groups, and projects, the initial population of this cache can take some time. If you have a large Swarm system you should read through the Redis server connection and configuration options before installing or upgrading Swarm, see "Redis server" on page 539.

API support changing for Swarm User Interface release (2020)

We have started adding a new set of v10 APIs to Swarm. These will provide a new endpoint and response pattern, and are designed for use with the new rich User Interface that is being introduced in 2020. Swarm will continue to support the v9 APIs for some time, see "API versions" on page 664.

API support changed for Swarm 2019.3

APIs older than v9 are being deprecated, support for them will be removed in a future release. See "API versions" on page 664.

Workflow feature support changed for Swarm 2019.2

The Swarm workflow feature is enabled by default. If you are upgrading from an earlier version you will need to update your triggers, see "Upgrade Swarm" on page 193.

PHP and Apache version support changed for Swarm version 2019.1

We have removed support for versions of PHP older than 7.0 in Swarm 2019.1. As a result of this change, support for Apache 2.2 has also been removed. This is part of our commitment to move away from using versions of platforms that have reached End-of-Life (EOL).

Ensure that you can install a supported version of PHP and Apache before upgrading to Swarm 2019.1. For information on versions of PHP and Apache supported by Swarm, see "PHP" on page 105 and "Apache web server" on page 103.

P4PHP 2019.1 or later required for Swarm 2019.1 and later

The latest version of P4PHP is included in the Swarm package, OVA, and tarball installations.

Known limitations

Swarm shelvedel trigger can fail when Helix server is on Windows and a shelf is deleted from the command line

The Swarm `shelvedel` trigger will fail in the following specific situation:
When the Helix server is hosted on Windows and a user runs the shelf delete command while in the root of one of their local drives.
For example, `c:\p4 -u bruno -c my_shelf shelve -d -c 9`

Unsupported characters in user names and group names in Swarm 2019.2 and later

Swarm does not support the following characters in user names and group names: `:@{}()`

Swarm support for the "Private editing of streams" feature in Helix server 2019.1 and later

Supported: stream specs can be edited in your workspace using the Private editing of streams feature and they are displayed in reviews.
Not supported: Swarm cannot commit locked stream specs.

Multiple-Helix server instances on a single Swarm instance

Issue: Swarm will lose connection to all of the Helix servers if you edit the base_url configurable value in the `environment` block of `<swarm_root>/data/config.php`. This will stop your system working.

Fix: Remove the `base_url` configurable from the `environment` block of `<swarm_root>/data/config.php`.

Global Dashboard does not support Single Sign-On (Helix Authentication Service)

Issue: If Helix Authentication Service is enabled for Swarm and the **Try to login to all available servers with these credentials** checkbox or the **All available servers** option is selected in a login dialog, Swarm will not try to log in to any of the other Helix server instances that are configured for Helix Authentication Service.
Workaround: Log in to them individually using the instance **Log in** button in the sidebar or by including the server instance name in the URL, for example: `https://swarm.company.com/serverA`.

Project Commits tab can fail to show some Helix server commits in the top level view

The **Project Commits** tab top level client view is made up of all of the branches of the project.

For example, Project Alpha:

Branch: QA:
`//depot/alpha/dev/QA/...`

Branch: Dev :
`//depot/alpha/dev/...`
`-//depot/alpha/dev/QA/...`

The project commits tab view is generated by processing the branches in the order that they were created in and from top to bottom for the paths in each of those branches.
For the project Alpha example above:

The first path includes all of the directories and files under `//depot/alpha/dev/QA/` in the project commits tab top level view.
The second path includes all of the directories and files under `//depot/alpha/dev/` in the project commits tab top level view.
The third path excludes all of the directories and files in `//depot/alpha/dev/QA/` from the project commits tab top level view.

Result: commits made to `//depot/alpha/dev/QA/` that should be shown for the **QA** branch are not displayed in the **Project Commits** tab top level view.

When you have a simple branch structure this can be avoided by considering this issue when you create your branches. In the example above, creating the **Dev** branch first and then creating the **QA** branch avoids the problem because`//depot/alpha/dev/QA/` is not excluded from the **Project Commits** tab top level view.

> **Tip**
> Individual branch views display the commits correctly.

Swarm OVA installation fails with a `Run p4 login2` error

Issue: Swarm OVA deployment against a Helix server configured for Helix Authentication Service fails with a `Run p4 login2` error.
Workaround: You must run `p4 login2` for a super user account configured for Helix Authentication Service before deploying the Swarm OVA. For `p4 login2` detail, see p4 login2 in *Helix Core P4 Command Reference*.

Task Stream Reviews

Pre-commit reviews in a task stream are not yet supported.

2 | Quickstart

Often, the best way to learn how software works is to try it. If the interface is sufficiently discoverable, you may not need to refer to documentation much, or at all. Sometimes, you just need to see the required steps to complete a task; that's what this chapter is all about.

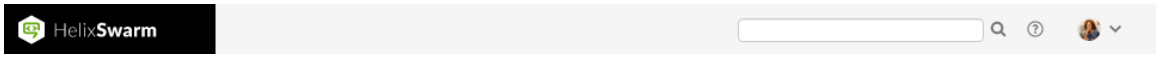
Swarm quick reference

This section contains quick reference guides to help to familiarize you with Swarm:

Menu quick reference

The Header, Main, and Project menus are used to quickly navigate around Swarm.

Header

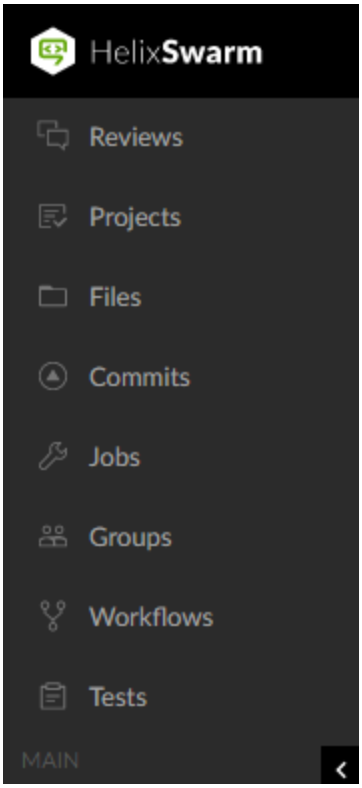


-  click to display the Dashboard and Activity tabs.
- **Search box:** search for projects, groups, users, and files that match your search string.
-  click to view the Swarm help.
- **Log in** (only displayed if you are not logged in to Swarm): click to log in to Swarm, see Log in to Swarm.
- **User id** dropdown menu (only displayed if you are logged in to Swarm):

- **My Profile:** click for your activity, shelves, profile settings, and notification settings. For more information about your profile, see "Viewing your user profile" on page 282.
- **System Information** (*admin* and *super* users only): click to view the Swarm system information page. For details about the information on this page, see "System Information" on page 580.
- **About Swarm**: click to view Swarm version information.

  If you are a user with *super* privileges, Swarm also displays the trigger token required when you configure Helix Core server for Swarm
- **Log Out:** log out of Swarm, see "Log out of Swarm" on page 322.

Main menu

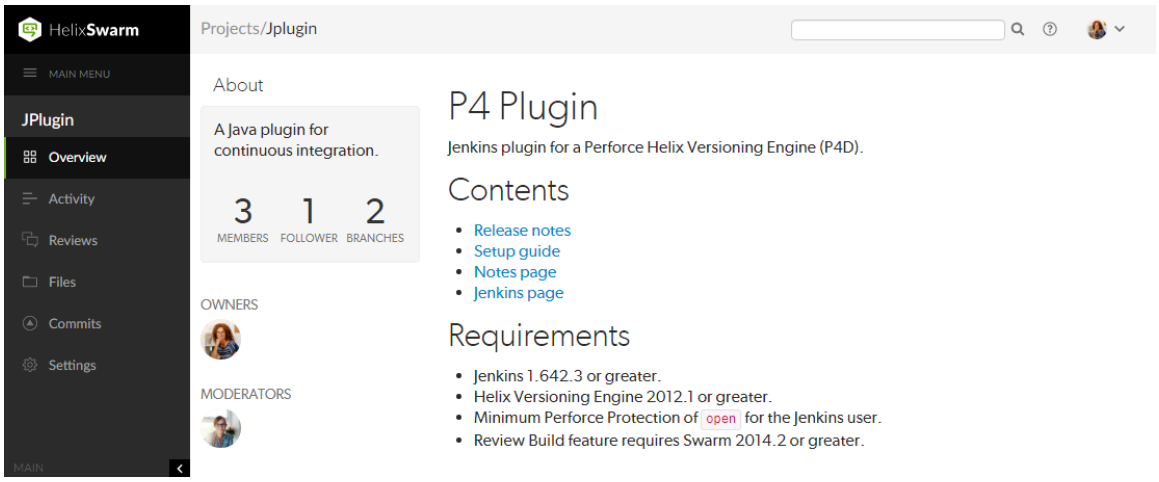


-  click to display the Dashboard and Activity tabs.
- **Reviews:** click to display a list of open and closed reviews. For more information about the reviews list page, see "Reviews list" on page 342.
- **Projects:** click to display a list of Swarm projects. For more information about the projects page, see "Projects" on page 299.
- **Files:** click to display a list of files in the Helix Core server, starting at the top level. For more information about the files page, see "Files" on page 243.
- **Commits:** click to display a list of commits made in the Helix Core server. For more information about the commits page, see "Commits" on page 252.
- **Jobs** (only displayed if Perforce jobs are configured): click to display a list of Perforce jobs in the Helix Core server. For more information about the jobs page, see "Jobs" on page 254.
- **Groups** (displayed by default but can be hidden by administrator, see "Groups configuration" on page 501.): click to display the Helix Core server groups in Swarm, see "Groups" on page 393.
- **Workflows** (not displayed if the Workflow feature is disabled): click to display a list of Swarm workflows. For more information about the workflows page, see "Workflows" on page 307.
- **Tests** (not displayed if the Workflow feature is disabled): click to display a list of Swarm tests. For more information about the tests page, see "Tests" on page 312.
- **<** (only displayed if the main menu is expanded): click to collapse the menu.
- **>** (only displayed if the main menu is collapsed): click to expand the menu.
- **Helix server instance name** (only displayed if Swarm is connected to multiple Helix servers): the instance name is displayed in the bottom left of the menu.

Project menu

1. Click the **Projects** link in the Swarm main menu.
2. Select a project from the Projects page.

   The project overview page is displayed:



- click to display the Dashboard and Activity tabs.
- **MAIN MENU** click to display the "Main menu" on the previous page.
- **Overview** (only displayed if there is a README markdown file in the project's mainline, see "Markdown in projects" on page 334): click to display an overview of the project. For more information about the project overview page, see "Overview page" on page 301.
- **Activity:** click to display the activity stream for the project. For more information about the project activity page, see "Activity page" on page 302.
- **Reviews:** click to display a list of open and closed code reviews for the project. For more information about the project reviews page, see "Reviews page" on page 303.
- **Files:** click to display a list of files for the project. The project Files page shows a list of files for the project, starting with a folder view representing each branch. Branches are designated with the *branch* icon . For more information about the files page, see "Files page" on page 303.
- **Commits:** click to display a list of changes made to the project. For more information about the commits page, see "Commits page" on page 303.
- **Jobs** (only displayed if Perforce jobs are configured): click to display a list of jobs associated with the project. For more information about the jobs page, see "Jobs page" on page 304.
- **Settings** (only displayed if you have permission to view or edit the project): click to display the project settings. For instructions on how to change the project settings, see "Add a project" on page 399.
- **<** (only displayed if the project menu is expanded): click to collapse the menu.

- **>** (only displayed if the project menu is collapsed): click to expand the menu.
- **Helix server instance name** (only displayed if Swarm is connected to multiple Helix servers): the instance name is displayed in the bottom left of the menu.

Quick URLs quick reference

Swarm handles URLs intelligently to reduce the amount of information you need to enter to quickly locate what you are looking for:

1. Enter a URL in the following format:

   `https://myswarm.url/<identifier>`
2. Swarm checks the `<identifier>` and redirects you to the first match it finds. Swarm checks for the identifier in following order:
   - Reviews
   - Changelists
   - Depot paths
   - Projects
   - Jobs
   - Users
   - Groups
   - Depot names
   - Git Fusion SHA1 (or fragments).

Note

- Changelist and review identifiers must be numeric.
- Git Fusion SHA1 identifiers (or a fragment of one) work when the SHA1 refers to a single changelist. If the SHA1 refers to more than one changelist, which can occur for Git branches, Swarm reports a `404 error`.
- If you enter an identifier that does not exist for any type of resource, Swarm displays a `Page Not Found` error.

Example usage

Display the latest version of a review

For example, to display review 124: enter the URL `https://myswarm.url/124`, Swarm redirects to `https://myswarm.url/reviews/124`.

Tip

- To display a specific version of a review, enter the full URL including **reviews** and append **/v<*n*>/** to the URL. For example, to display version 2 of review 124: enter **https://myswarm.url/reviews/124/v2/**
- To display a diff between two versions of a review, enter the full URL including **reviews** and append **/v<*n,n*>/** the URL. For example, to display the diff between version 2 and 4 of review 124: enter **https://myswarm.url/reviews/124/v2,4/**
- To open a review with a specific tab open, append **#<*tab name*>** to the end of the URL (**#<*tab name*>** must be the last item in the URL):
  - **Files tab:#files** (default if **#<*tab name*>** is not specified)
  - **Comments tab:#comments**
  - **History tab:#history**

Display a changelist

For example, to display changelist 123: enter the URL **https://*myswarm.url*/123**, Swarm redirects to **https://myswarm.url/changes/123**.

Tip

To open a changelist with a specific tab open, append **#<*tab name*>** to the end of the URL:

- **Files tab:#files** (default if **#<*tab name*>** is not specified)
- **Comments tab:#comments**

Display the latest version of a file

Enter the URL **https://*myswarm.url*/depot/alpha/readme.txt**, Swarm redirects to **https://myswarm.url/files/depot/alpha/readme.txt**.

Tip

- To display a specific version of a file, enter the full URL including **files** and append **?v=<*n*>** to the URL. For example, to display version 2 of the **depot/alpha/readme.txt** file: enter **https://myswarm.url/files/depot/alpha/readme.txt?v=2**
- To open a file with a specific tab open, append **#<*tab name*>** to the end of the URL (**#<*tab name*>** must be the last item in the URL):
  - **View tab:#view** (default if **#<*tab name*>** is not specified)
  - **Commits tab:#commits**

Display the user profile page for jsmith

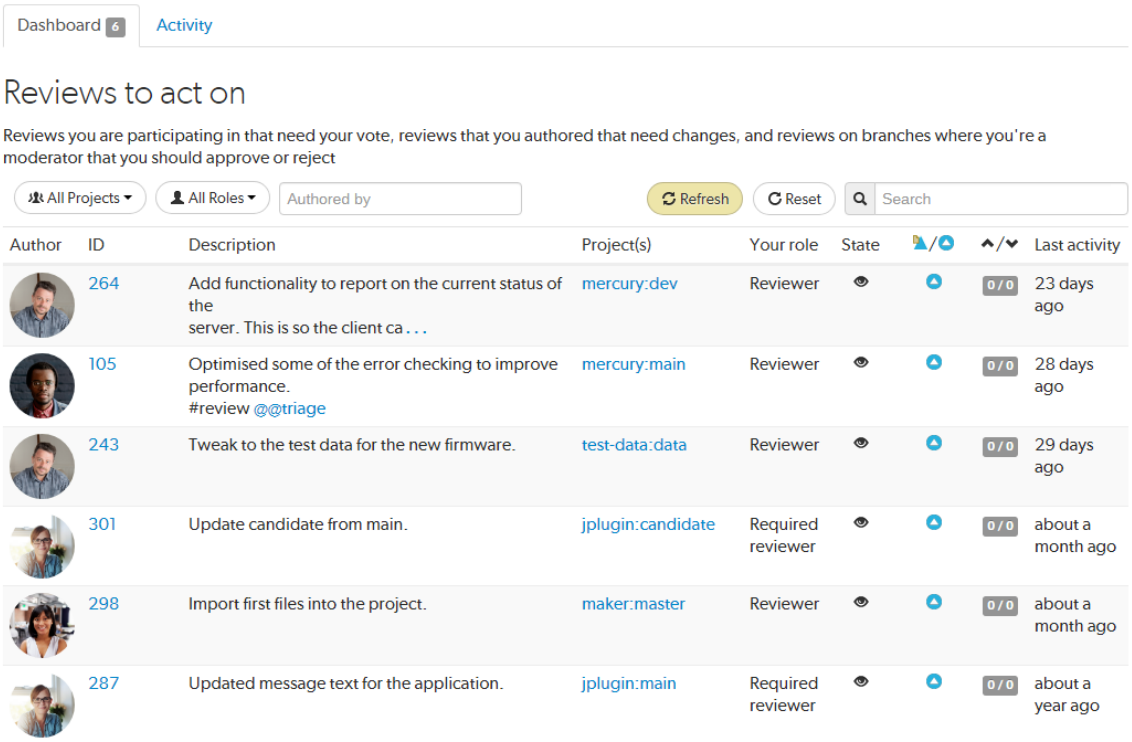
Enter the URL **`https://myswarm.url/jsmith`**, Swarm redirects to **`https://myswarm.url/users/jsmith`**.

Dashboard page quick reference

The purpose of the dashboard is to allow you to focus on reviews that need to be done, so that other users are not blocked. The dashboard lists the most recently modified reviews first, and shows your role in the review.

> **Note**
> Since it is tied to the logged in user, the dashboard is only populated if you are logged in.

1. Log in to Swarm.
2. If the Swarm dashboard is not already displayed, click the Swarm icon above the menu.



Jump directly to a specific area of the dashboard page using the following links:

- "Dashboard filters" on the next page
- "Dashboard content" on the next page

Dashboard filters

The filters are used to filter your dashboard reviews, the following filters are available:

- **Projects:** filter by the project the review is part of, limiting results to **My Projects** (projects you are an owner of or a member of) or to an individual project. The **All Projects** drop down will only show projects for which there are reviews in your dashboard.
- **Roles:** filter by your specific role in a review, limiting results to reviews for which you are the author, a reviewer, a required reviewer, or a moderator. The **All Roles** drop down will only show roles for which there are reviews in your dashboard.
- **Authored by:** filter the reviews to only those that have been authored by a certain user. Type in this field to get a drop down list of users to filter by.
- **Reset** button: click to reset all dashboard filters back to their defaults.
- **Search:** filter the reviews by searching the review descriptions.

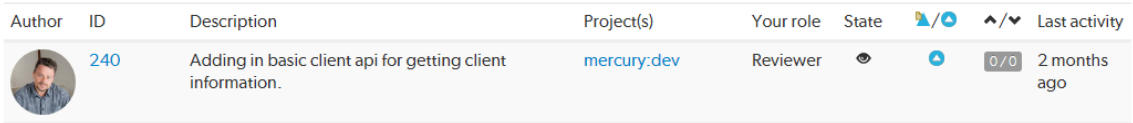
Refresh button

Refresh button (only displayed when new content is available for your dashboard): click **Refresh** to update your dashboard with the new content.

> **Note**
> By default, when your dashboard is open, Swarm checks for new content every five minutes.

Dashboard content

The dashboard displays a summary for each review.



- **Author** avatar: the review author's avatar, hover over the avatar to see the ID and name of the review author. Click on the avatar to go to the profile of the review author, see "Viewing another user's profile" on page 289.
- **ID:** the unique number used to identify the Swarm review. Click the review ID to display the review, see "Review display" on page 349.
- **Description:** the review description may be truncated if it is too long, in which case click on the ellipsis **...** to expand it.

- **Project(s):** lists the project branches this review covers. A review may span multiple branches and projects. Click a branch link to navigate to the project page for that branch.
- **Your role:** displays your role in the review and is the reason the review is in your dashboard. This can be Author, Reviewer, Required Reviewer, or Moderator.
- **State:** a review can be in one of the following states:
  - **Needs Review:** the review has started and the changes need to be reviewed.
  - **Needs Revision:** the changes have been reviewed and the reviewer has indicated that further revisions are required.
  - **Approved:** the review has been approved. The changes may need to be committed.

  > **Note**
  > The Approved state only applies to review authors, it is only shown for a review that has been approved but has not been committed.

- **Type:** displays the type of review, either a pre-commit review or a post-commit review.
- **Votes up/down:** displays the number of up votes and down votes for the review.
- **Last activity:** displays the last time that any changes were made to the review, including votes, comments, commits, and file changes.

Global dashboard page quick reference

> **Note**
> Global dashboard is only available if your Swarm administrator has configured Swarm to connect to more than one Helix Core server instance.

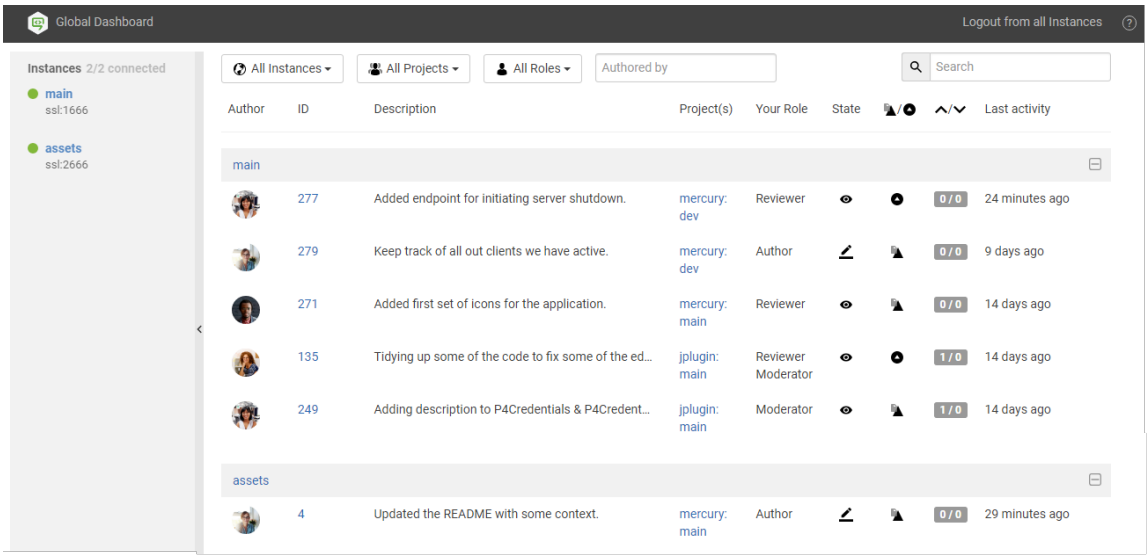
The purpose of the global dashboard is to allow you to focus on reviews that need to be done, so that other users are not blocked. The global dashboard lists reviews by Helix server according to the most recently modified first, and shows your role in the review.

1. Enter the basic Swarm URL without a Helix server instance name, for example: `https://swarm.company.com`
2. If the log in dialog is displayed it is the same dialog that is displayed when you click **Log In** from the banner, log in to the global dashboard.

   For instructions about logging in to the global dashboard, see "Log in to one or more Helix server instances" on page 233.

> **Tip**
>
> - Since it is tied to the logged in user, the global dashboard is only populated for the Helix server instances you are logged in to in Swarm.
> - If you are already logged in to a Helix server instance in Swarm, the dashboard for that server will be automatically populated when you open or refresh the global dashboard.

The global dashboard is displayed:



The global dashboard is made up of three main areas:

- "Toolbar" below: log in to Helix server instances, log out of all Helix server instances, and open the Swarm help from the global dashboard toolbar.
- "Sidebar" on the facing page: Log in/logout of individual Helix server instances and view you profile for any instance that you are logged in to.
- "Helix server dashboards" on the facing page: view, filter and search the dashboards of the Helix server instances you are logged in to. Jump directly to a Helix server, review, or project by clicking on a link.

Toolbar

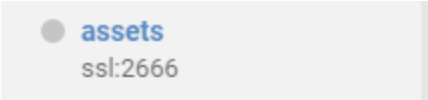
- **Log In** (only displayed in the toolbar if you are not logged in to any Helix server instances): click to log in to one or more Helix server instances, see "Log in to one or more Helix server instances" on page 233.
- **Logout from all instances** (only displayed in the toolbar if you are logged in to at least one Helix server instance): click to logout from all of the Helix server instances, see "Logout of all Helix

server instances" on page 235.

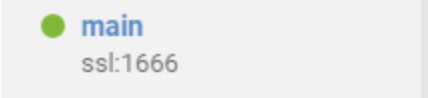
- **Help**: click to view the Swarm help.

Sidebar

- **Log in status:**
  - Not logged in to the Helix server instance:



  - Logged in to the Helix server instance:

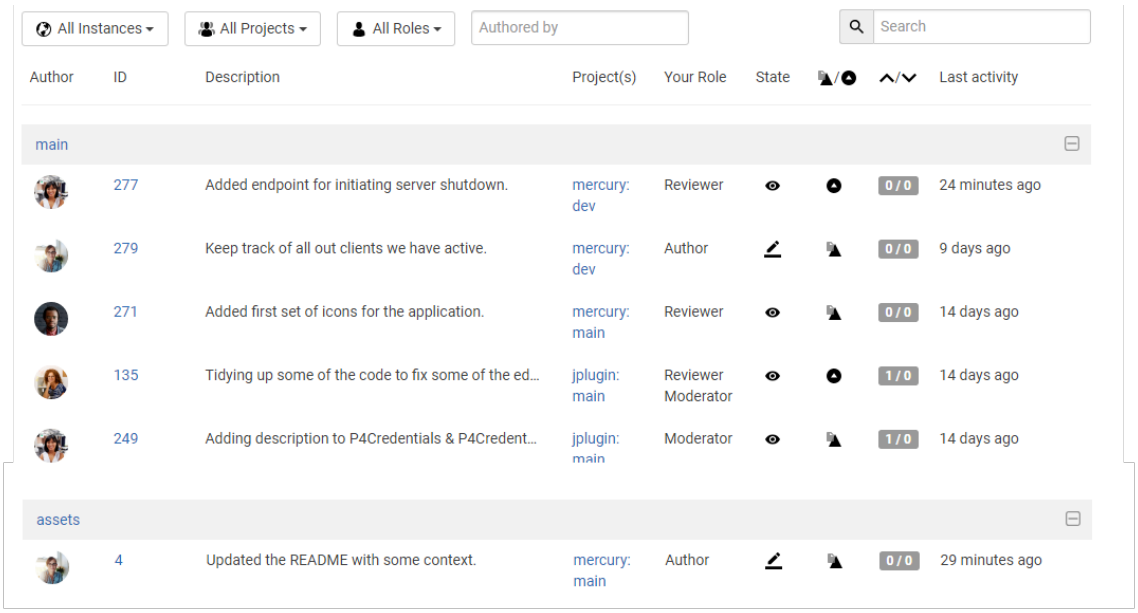


- Helix server instance actions:
  - **Log in** (only available if you are not logged in to the Helix server instance): hover over the instance name and click the **Log in** button when it is displayed, see "Log in to a Helix server instance" on page 236.
  - **Logout** (only available if you are logged in to the Helix server instance): hover over the instance name, click the **Gear** button , and select **Logout** from the dropdown menu to logout of the instance, see "Logout of a single Helix server instance" on page 238.

Helix server dashboards

The global dashboard lists reviews by Helix server according to the most recently modified first, and shows your role in the review.

Example global dashboard showing a number of Helix server dashboards:



Jump directly to a specific area of the dashboard page using the following links:

- "Dashboard filters" below
- "Dashboard content" on the facing page

Dashboard filters

Used to filter your dashboard reviews, the following filters are available:

- **Instances:** filter by the Helix server instance, limiting results to **All Instances** or to an individual instance.
- **Projects:** filter by the project the review is part of, limiting results to **All Projects** or to an individual project. The **All Projects** drop down will only show projects for which there are reviews in your dashboard.
- **Roles:** filter by your specific role in a review, limiting results to reviews for which you are the author, a reviewer, a required reviewer, or a moderator. The **All Roles** drop down will only show roles for which there are reviews in your dashboard.
- **Authored by:** filter the reviews to only those that have been authored by a certain user. Type in this field to get a drop down list of users to filter by.
- **Reset** (only displayed if one or more filters are set): click the **Reset** button to reset all dashboard filters back to their defaults.
- **Search:** filter the reviews by searching the description and review ID.

Dashboard content

The dashboard for each Helix server shows a summary of the information for each review.

> **Tip**
> - Click on the Helix server name in the header bar to open Swarm for that instance in a new tab.
> - Click on the header bar to the right of the Helix server name to collapse the dashboard for that instance. Click again to expand the dashboard for the instance .

- **Author** avatar: the review author's avatar, hover over the avatar to see the ID and name of the review author. Click on the avatar to go to the profile of the review author, see "Viewing another user's profile" on page 289.
- **ID:** the unique number used to identify the Swarm review. Click the review ID to display the review, see "Review display" on page 349.
- **Description:** the review description may be truncated if it is too long, in which case click on the review description to expand it.
- **Project(s):** lists the project branches this review covers. A review may span multiple branches and projects. Click a branch link to navigate to the project page for that branch.
- **Your role:** displays your role in the review and is the reason the review is in your dashboard. This can be Author, Reviewer, Required Reviewer, or Moderator.
- **State:** a review can be in one of the following states:
  - **Needs Review:** the review has started and the changes need to be reviewed.
  - **Needs Revision:** the changes have been reviewed and the reviewer has indicated that further revisions are required.
  - **Approved:** the review has been approved. The changes may need to be committed.

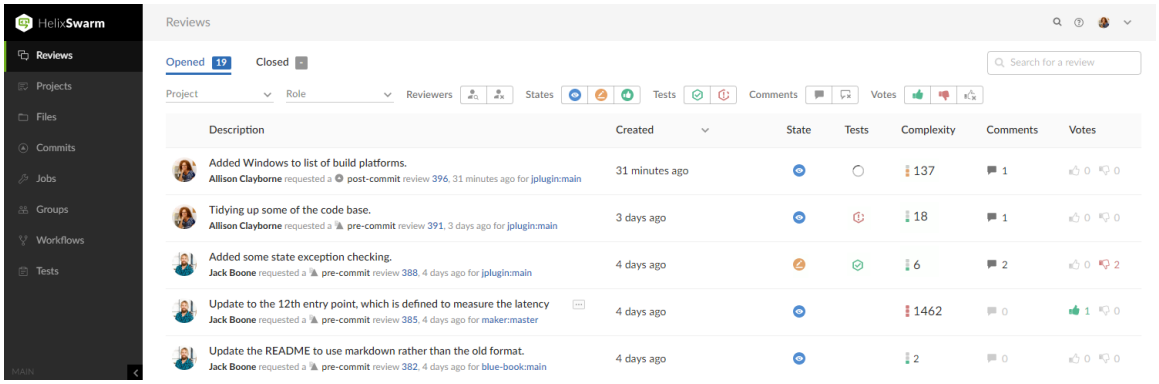
  > **Note**
  > The Approved state only applies to review authors, it is only shown for a review that has been approved but has not been committed.

- **Type:** displays the type of review, either a pre-commit review or a post-commit review.
- **Votes up/down:** displays the number of up votes and down votes for the review.
- **Last activity:** displays the last time that any changes were made to the review, including votes, comments, commits, and file changes.

Reviews list page quick reference

The **Reviews** page lists code reviews that are in progress, and code reviews that are complete.

1. Log in to Swarm.
2. Click **Reviews** in the main menu.



Jump directly to a specific area of the **Reviews** page using the following links:

- "Opened and Closed tabs" below
- "Review filters" on the facing page
- "Reviews list content" on page 45

Opened and Closed tabs

The **Opened** and **Closed** tabs display:

- **Opened** tab: displays a list of all code reviews that have started, are being reviewed, are awaiting revisions, or need to be committed.
- **Closed** tab: displays a list of all code reviews that have been approved and committed, rejected, or archived.

Tip

- The **Opened** and **Closed** tabs display the number of reviews in each tab. Initially this can be an over estimate which Swarm dynamically corrects as more information becomes available to it.
- The **Closed** tab initially displays a – character. The – is replaced by a number when you navigate to the **Closed** tab.

Review filters

> **Tip**
> Swarm updates the URL in your browser to reflect the filter options you have selected. This makes it easy to bookmark or share your review list filter settings with the URL.
>
> If you share a URL with **Comment** or **Vote** filters selected, those filters are applied to the user running the URL. This means that their reviews list page will contain different reviews to your reviews list page.

The following filters are available (from left to right):

- **Branch** (only available on the project "Reviews page" on page 303): a dropdown menu that lets you filter which reviews to display based on the current project branches:

  - **All branches:** all reviews are displayed for all branches within the current project.
  - **Select branch:** selecting a branch name displays only reviews for that project branch.
  - **Branch search:** An auto-complete search field that allows you to choose one of the branches within the current project. Once specified, only reviews for the selected project branch are displayed.
- **Project** dropdown: filter by the project the review is part of:

  - **All projects:** displays all reviews for all projects.
  - **My projects:** displays all reviews for all of the projects you are participating in, as a member, owner, moderator, or follower.
  - **Select project:** selecting a project name only displays reviews for that project.
  - **Project search:** an auto-complete search field that allows you to choose one of the projects defined in Helix server. Once specified, only reviews for the selected project are displayed.
- **Role** dropdown: filter by the author of the review, options are:

  - **All reviews:** displays all reviews.
  - **Reviews I've authored**: displays reviews that you have authored.

- **Reviews I'm participating in:** displays reviews that you are a reviewer of, but not an author of.
- **Reviews I'm an individual reviewer of:** displays reviews that you are an individual reviewer of , but not a group reviewer of, or an author of.
- **Review I've authored or I'm participating In:** displays reviews that you have authored, or are a reviewer of.
- **Authored by:** an auto-complete search field that allows you to choose one of the user accounts defined in the Helix server. Once specified, only reviews authored by the user are displayed.

- **Reviewers** buttons, (**Opened** tab only):

  - **Has reviewers**: displays reviews that have one or more reviewers.
  - **No reviewers**: displays reviews that have no reviewers.
- **States** buttons, (**Opened** tab):

  - **Needs Review**: displays reviews that need to be reviewed.
  - **Needs Revision**: displays reviews that have been reviewed, but need further revisions before the review can be accepted.
  - **Approved**: displays reviews that have been approved, and should be committed.
- **States** buttons, (**Closed** tab):

  - **Approved**: displays reviews that have been approved and committed.
  - **Rejected**: displays reviews that have been rejected.
  - **Archived**: displays reviews that have been archived.
- **Tests** buttons (when automated tests are enabled for the associated project):

  - **Tests passed** displays reviews that have passed tests.
  - **Tests failed:** displays reviews that have failed tests.
- **Comments** buttons:

- **I have commented on:** displays reviews that you have commented on.
- **I have not commented on:** displays reviews that you are participating in but have not commented on

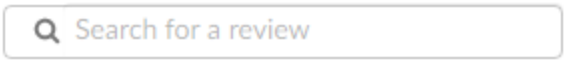
> **Note**
> Filters for commenting only apply to reviews which you are a participant of. Commenting on or voting on a review will automatically add you as a participant. If you leave the review after commenting on it, then this review will not be included in the list.

- **Votes** buttons:

  - **I have voted up:** displays reviews that you have voted up.
  - **I have voted down:** displays reviews that you have voted down.
  - **I have not voted on:** displays reviews that you are participating in but have not voted on.

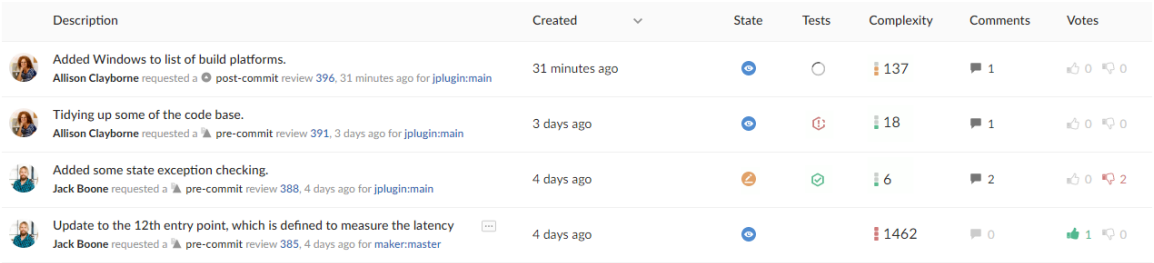
> **Note**
> Filters for voting only apply to reviews which you are a participant of. Commenting on or voting on a review will automatically add you as a participant. If you leave the review after voting on it, then this review will not be included in the list.

- **Search for a review:** search for a partial match of review description content and an exact match for *reviewid*, *userid*, and *projectid*.

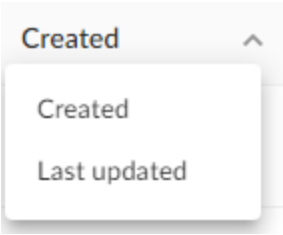


Reviews list content

The review list displays a summary for each review.



- **Avatar** displays the avatar of the review author, click to view the profile of the author
- **Description** contains:
  - first line of the review description, click to open the review. If the review description is too long it is truncated, click on the ellipsis ··· to expand it in the list page.
  - review author, click to view the profile of the author
  - review type, pre-commit or post-commit
  - review number and version, click to open the review
  - review creation date and time
  - review project branches, click to open the project
- **Created** or **Last activity** dropdown: click to change the order the reviews are displayed in, options are:



  - **Created**: Reviews sorted by when they were created
  - **Last activity**: Reviews sorted by when they were last updated

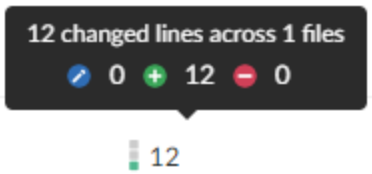
> **Note**
> - If the **Result order** button is not displayed reviews are sorted by when they were created.
> - **Result order** button display is a global setting controlled by the Swarm administrator. See "Reviews filter" on page 551 for details.
> - When review results are older than 24 hours they are displayed in numerical order within each day.

- **State:** a review can be in one of the following states:
  - **Needs Review:** the review has started and the changes need to be reviewed.
  - **Needs Revision:** the changes have been reviewed and the reviewer has indicated that further revisions are required.
  - **Approved:** the review has been approved. The changes may need to be committed.
- **Tests:** displays the test suite state for the review, either tests in progress, tests passed, or tests failed.

- **Complexity:** a traffic light icon and number shows the relative complexity of the review and the total number of lines changed in the review. By default, complexity icon displays:
  - **Red:** ≥ 300 changes
  - **Amber:** < 300 and >30 changes
  - **Green:** ≤ 30 changes

  Tip
  - Review complexity is only calculated for a review when the review is updated and the file content has changed.
  - Review complexity is only stored for the current revision of a review.

  Hover over the complexity icon to display more detailed information:

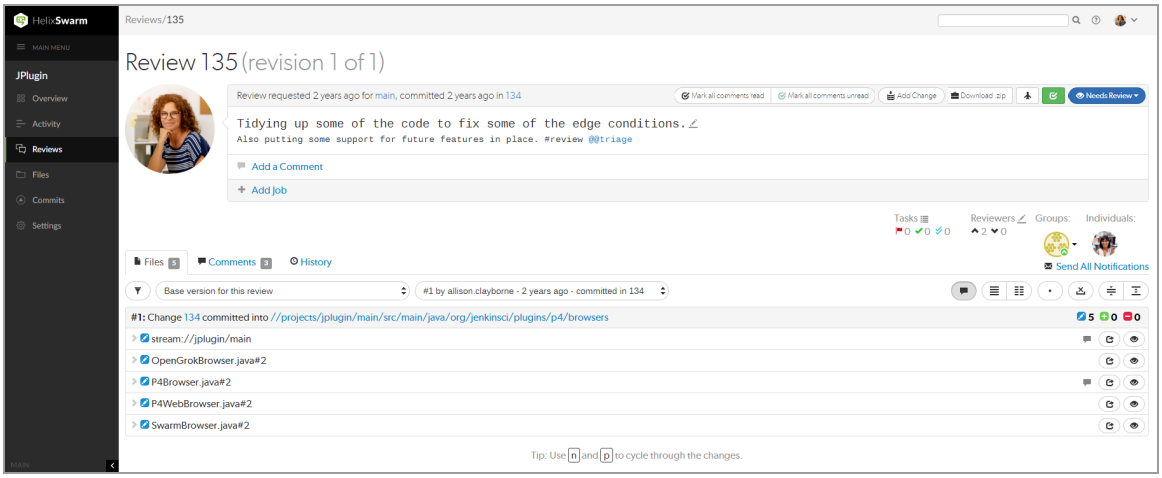
- **Comments:** displays the number of open (non-archived) comments that are associated with the review. The icon is filled if there are comments on the review.
- **Votes:** displays the number of up votes and down votes for the review. The appropriate vote icon is filled if you have voted on the review, vote icons with only an outline show votes by others.

Review page quick reference

The Swarm review page is used when reviewing changes:

1. Log in to Swarm.
2. To display the Swarm home page, click the Swarm icon above the menu.
3. If your dashboard is not already displayed, click the **Dashboard** tab.

4. To open a review, click the **ID** of the review.



Jump directly to a specific area of the review page using the following links:

- "Review description" below
- "Tasks, author, and reviewer area" on page 51
- "Files tab" on page 52
  - "File diff view" on page 55
- "Comments tab" on page 57
  - "Comment view" on page 58
  - "Add a comment" on page 60
- "History tab" on page 60

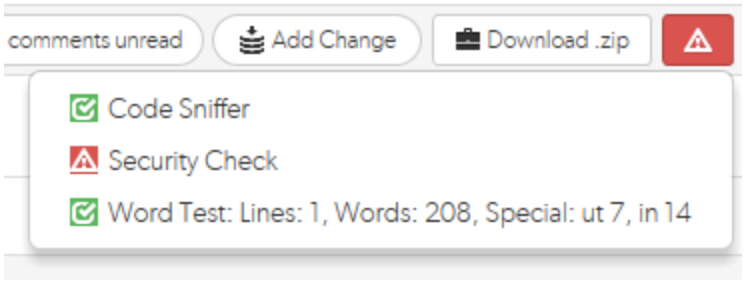
Review description

The review description is made up of the following elements (from left to right):

- **Review ID:** the unique number used to identify the Swarm review. The revision of the review being viewed and the total number of revisions available are shown in brackets.
- **Review author** avatar: the review author's avatar, hover over the avatar to see the ID and name of the review author. Click on the avatar to go to the profile of the review author, see "Viewing another user's profile" on page 289.

- **Review header:**
  - **Review information:** displays when the review was requested, the project branch the files are in, when the review was committed, and the changelist that the review was requested for.
  - **Mark all comments read** button: click to mark all of the comments on this review as read, see "Mark comments as read" on page 279.
  - **Mark all comments unread** button: click to mark all of the comments on this review as unread, see "Mark comments as unread" on page 279.
  - **Add Change** button: click to add a changelist to the review. Options available depend on whether the review is pre-commit or post-commit. For information about adding a changelist to a review, see "Add Change button" on page 350.
  - **Download .zip** button (if configured): click to download a compressed archive of all of the files in the review, see "Download files as a ZIP archive" on page 73.
  - **Deployment status** button (if configured): indicates the success  or failure  of the review deployment. If your deployment system provides a URL to the deployment you can click it and view the deployment results.
  - **Test status** button (if configured): indicates the current test status for the review revision:
    -  **Tests in Progress:** displayed if any of the tests for the review revision are running.
    -  **Tests Passed:** displayed if all of the tests for the review revision have passed.
    -  **Tests Failed:** displayed if any of the tests for the review revision have failed.

    To view the individual tests for the review revision, click the **Test status** button to open the dropdown list:

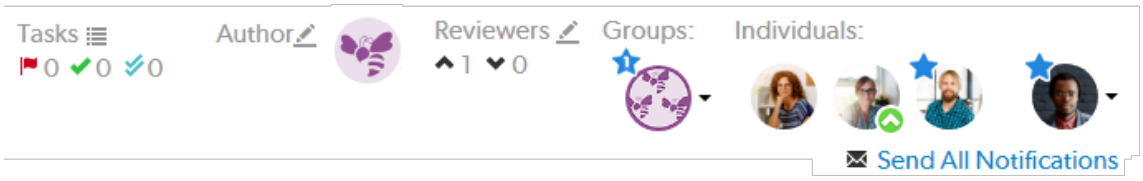
    

    If your continuous integration system calls back to Swarm with a URL, the test run in the dropdown will be linked to the URL provided.
  - **Review state** button: indicates the current state of the review and is used to change the state of the review:

- **Review states:** a review can be in one of the following states:
  - **Needs Review:** the review has started and the changes need to be reviewed.
  - **Needs Revision:** the changes have been reviewed and the reviewer has indicated that further revisions are required.
  - **Approved:** the review has been approved. The changes may need to be committed.
  - **Rejected:** the review has been completed. The changes are undesirable and should not be committed.
  - **Archived:** the review has been completed for now but it is not rejected, or approved. The review has been filed away in case it is needed in the future.
- **Change review state:** click the **Review State** button and select a new state from the review dropdown menu. State change options are only displayed if you are authorized to make the state change:
  - **Needs Revision:** select to request changes to the files in the review.
  - **Needs Review:** select to request further review of the changes.
  - **Approve** (only available if the voting requirements for the review are satisfied. For information on voting requirements, see "Required reviewers" on page 380.): select to approve the review.
  - **Commit** (only available for pre-commit reviews that have been approved): select to commit the review.
  - **Approve and Commit** (only available for unapproved pre-commit reviews when the voting requirements for the review are satisfied. For information on voting requirements, see "Required reviewers" on page 380.): select to approve and commit the review in a single step, see "Approve and Commit" on page 391.
  - **Reject:** select to reject the review.
  - **Archive:** select to archive the review.
  - **Obliterate Review** (by default, only available for users with *admin* or *super* user rights): see "Obliterate Review" on page 533.
- **Review description:** the review description is automatically copied from the changelist that was originally used to create the review. To change the review description, click the **Edit Description**  button.
  - **Update pending changelist** checkbox in the **Edit Description** dialog (only available if you are editing the description of a pre-commit review and you are the original author of the changelist that created the review): select the checkbox to also apply your review description changes to the original changelist description.

- **Add a Comment** button: click to add a comment to the review description, view existing description comments, or hide existing description comments, see "Commenting on a changelist or review description" on page 268.
- **Jobs** (if configured): Perforce jobs can be linked to the review. For more information on Perforce jobs, see "Jobs" on page 254.
  - Add Job link: click to add a job to the review. Select the job from the dialog that is displayed. For more information on linking a job to a review, see "Add jobs" on page 257.
  - **jobnnnnnn** (where **nnnnnn** is the job number): click the job number to view details of the Perforce job. For more information about Perforce jobs, see "Job display" on page 255.
  - **Unlink Job** button: to unlink a job from the review, click the **Unlink Job** button to the left of the job you are unlinking from the review.

Tasks, author, and reviewer area



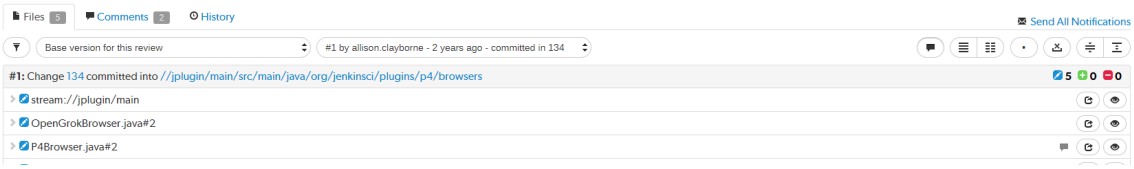
The tasks, author and reviewers area is made up of the following elements (from left to right):

- **Tasks**
  - **Tasks list** button: click to display a dialog listing all tasks associated with the review. The task list can be filtered by task type, and tasks can be viewed from the task list, see "Task list" on page 271.
  - **Task Summary** icons: shows a summary of the number, and status of comments flagged as tasks for the review. For more information about tasks, see "Tasks" on page 270.
    - **Red Flag** icon: indicates the numbers of open tasks on the review.
    - **Green Check Mark** icon: indicates the numbers of tasks that have been addressed on the review.
    - **Blue Double-Check Mark** icon: indicates the number of tasks that have been addressed and verified on the review.
- **Author:**
  - **Edit Author** button (if enabled): click to change the review author, see "Changing the review author" on page 360.
  - **Review author** avatar: the review author's avatar, see "Avatars" on page 328.

- **Reviewers:**
  - **Edit Reviewer** button (if enabled): click to edit the reviewers for the review
  - **Up** vote and **Down** vote count: indicates the number of up votes and down votes the review has.
- **Groups:** lists groups that are reviewers for the review. When at least one person in the group has voted, the avatar displays a badge indicating whether the group, as a whole, has voted up or down. Click on the group to see who has voted, and how they have voted, see "Group reviewer" on page 355.
- **Individuals:** lists individuals that are reviewers for the review. When an individual has voted on the review, their avatar displays a badge indicating whether they voted up or down. For more information about individual reviewers, see "Individual reviewer" on page 357.
- Your **Avatar**, located to the right of the reviewers: click your avatar to interact with the review. This allows you to join the review, leave the review, vote up, vote down, clear your vote, disable notifications for this review, make your vote optional, or make your vote required. For more information about interacting with the review, see "Reviewers" on page 355.
- **Send All Notifications** link: comment notifications are delayed by default, click to manually send the notification immediately for the review. For more information about comment notification delay, see "Comment notification delay" on page 273.

Files tab

Use the **Files** tab to view the files in the review and to see how they have changed using the Swarm diff view.



Swarm supports stream specs in your workspace using the Private editing of streams feature. If a changelist or review contains a stream spec, it will be displayed first in **Files** with the prefix **stream:** //, for example: **`stream://MyStreamDepotName/MyStreamSpecLocationName`**. A changelist/review can only contain one stream spec.

The **Files** tab is made up of the following elements (from left to right):

- **Filter comments** button: click to limit displayed comments to those made on the selected review version.
- **Review revision selectors:** select which revisions of the review you want to diff. For details on using the revision selectors, see "Select review revisions to view" on page 361.

- **Global diff view** buttons (from left to right): control the initial diff view for all of the files in the review:

  - **Show Comments** button: toggles the display of comments to inline in files or only in the **Comments** tab.
  - **Show Diffs In-Line** button: displays all diffs as inline.
  - **Show Diffs Side-by-Side** button: displays all diffs as side-by-side.
  - **Toggle Show Whitespace** button: toggles the display of whitespace characters (such as space, tab, and newline) for all files.
  - **Toggle Ignore Whitespace** button: toggles the highlighting of whitespace changes in all of the file diffs.
    - **Highlight whitespace changes:** makes it easier to identify changes in file types where whitespace is important. This is the default value.
    - **Ignore whitespace changes:** whitespace changes are not highlighted, this makes it easier to see the important changes in file types where whitespace changes are not important.
  - **Collapse All** button: collapses all files
  - **Expand All** button: expands all files. By default this button is disabled if there are more than 10 files in the review. For details, see Expand All Limit.

> **Tip**
> The default states for the **Show Comments**, **Show Diffs Side-by-Side**, **Toggle Show Whitespace**, and **Toggle Ignore Whitespace** buttons are set in your user settings, see User Settings.

- **File listing** header:
  - Comparing the files in the latest version of the review to Base or Head: displays the current version of the files in the review, the changelist that the review is based on, and the common path for the review files.
  - Comparing two versions of the review files: displays which two versions of the review files are being compared, which changelists the files are in, and the common path for the files in both versions of the review.
- **File change type** icons: indicate the type of change for each file in the review:

- Added/Branched/Imported
- Edited/Integrated
- Deleted

- **File diff view** buttons (from left to right): control the diff view for each individual file:

> **Tip**
> Only the **Show Full Context** button is available for stream specs.

  - **Comments in File** icon (only displayed if the file contains comments): indicates that the file contains comments.
  - **Show In-line** button: highlights line additions, modifications, and removals in a single pane.
  - **Show Side-by-Side** button: highlights additions, modifications, and removals in two panes, with the older version of the file on the left, and the newer version on the right.
  - **Show Whitespace** button: makes whitespace characters more visible; spaces show up as dots, tabs show up as arrows that point to a bar, and line endings show up as down-pointing arrows.
  - **Ignore Whitespace** button: toggles the highlighting of whitespace changes in a file, making it easier to identify non-formatting changes. Normally, whitespace is not ignored.
  - **Toggle Ignore Whitespace** button: toggles the highlighting of whitespace changes in the file diff.
    - **Highlight whitespace changes:** makes it easier to identify changes in file types where whitespace is important. This is the default value.
    - **Ignore whitespace changes:** whitespace changes are not highlighted, this makes it easier to see the important changes in file types where whitespace changes are not important.
  - **Show all diffs** button (edited and integrated files only): toggles between displaying all the diffs for the file and only the first few. The limit of what are shown by default is configurable by the administrator (see Max Diffs) and defaults to 1500. If there are fewer than that then this button is hidden.
  - **Show Full Context** button (edited and integrated files only): toggles between displaying only the portions of the file that have changed and the full file.
  - **Open File** button (edited or integrated files only): opens a new browser tab/window display the full file (where possible), provide access to its history, and a button to download the file.

- **Mark file as read** button (displayed for code reviews only): helps you (and others) keep track of which files have been reviewed. This is particularly useful when a code review consists of many files.

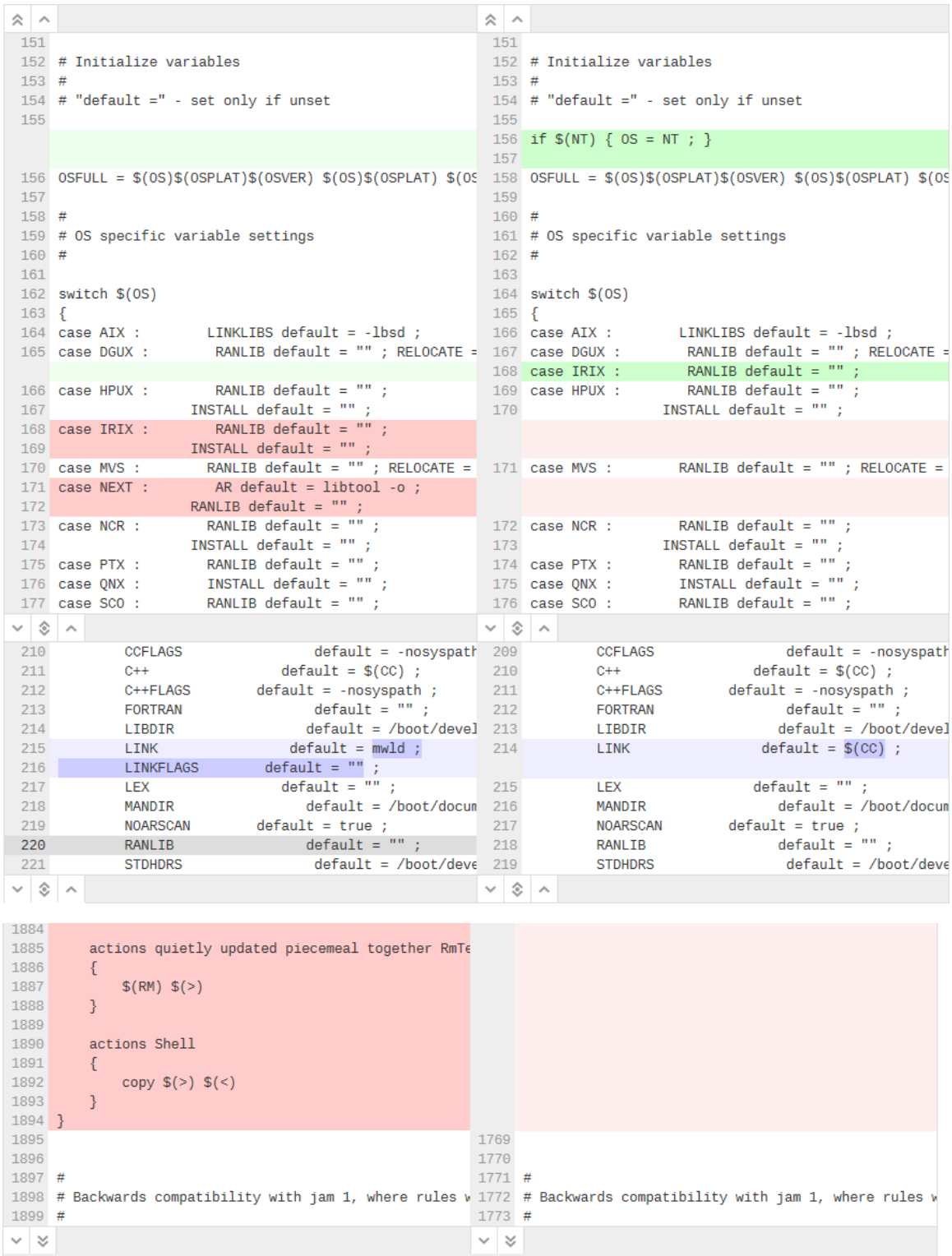
File diff view

- **Expand file** button: click to expand the file to see what has changed, this displays the diff view for the file. Use the **File diff view** buttons to change how the diff is displayed:

  When you view a diff, the changes are highlighted:

  - Red indicates lines that have been removed.
  - Blue indicates lines that have been modified.
  - Green indicates lines that have been added.

  Example side-by-side diff view:



```
151
152 # Initialize variables
153 #
154 # "default =" - set only if unset
155
156 OSFULL = $(OS)$(OSPLAT)$(OSVER) $(OS)$(OSPLAT) $(OS
157
158 #
159 # OS specific variable settings
160 #
161
162 switch $(OS)
163 {
164 case AIX :        LINKLIBS default = -lbsd ;
165 case DGUX :        RANLIB default = "" ; RELOCATE =
166 case HPUX :        RANLIB default = "" ;
167                  INSTALL default = "" ;
168 case IRIX :        RANLIB default = "" ;
169                  INSTALL default = "" ;
170 case MVS :         RANLIB default = "" ; RELOCATE =
171 case NEXT :         AR default = libtool -o ;
172                  RANLIB default = "" ;
173 case NCR :         RANLIB default = "" ;
174                  INSTALL default = "" ;
175 case PTX :         RANLIB default = "" ;
176 case QNX :         INSTALL default = "" ;
177 case SCO :         RANLIB default = "" ;
```

```
151
152 # Initialize variables
153 #
154 # "default =" - set only if unset
155
156 if $(NT) { OS = NT ; }
157
158 OSFULL = $(OS)$(OSPLAT)$(OSVER) $(OS)$(OSPLAT) $(OS
159
160 #
161 # OS specific variable settings
162 #
163
164 switch $(OS)
165 {
166 case AIX :        LINKLIBS default = -lbsd ;
167 case DGUX :        RANLIB default = "" ; RELOCATE =
168 case IRIX :        RANLIB default = "" ;
169 case HPUX :        RANLIB default = "" ;
170                  INSTALL default = "" ;
171 case MVS :         RANLIB default = "" ; RELOCATE =
172 case NCR :         RANLIB default = "" ;
173                  INSTALL default = "" ;
174 case PTX :         RANLIB default = "" ;
175 case QNX :         INSTALL default = "" ;
176 case SCO :         RANLIB default = "" ;
```

```
210         CCFLAGS                default = -nosyspath
211         C++                  default = $(CC) ;
212         C++FLAGS           default = -nosyspath ;
213         FORTRAN                default = "" ;
214         LIBDIR               default = /boot/devel
215         LINK                 default = mwld ;
216         LINKFLAGS          default = "" ;
217         LEX                  default = "" ;
218         MANDIR                 default = /boot/docum
219         NOARSCAN           default = true ;
220         RANLIB                 default = "" ;
221         STDHDRS                default = /boot/deve
```

```
209         CCFLAGS                default = -nosyspath
210         C++                  default = $(CC) ;
211         C++FLAGS           default = -nosyspath ;
212         FORTRAN                default = "" ;
213         LIBDIR               default = /boot/devel
214         LINK                 default = $(CC) ;
215         LEX                  default = "" ;
216         MANDIR                 default = /boot/docum
217         NOARSCAN           default = true ;
218         RANLIB                 default = "" ;
219         STDHDRS                default = /boot/deve
```

```
1884
1885     actions quietly updated piecemeal together RmTe
1886     {
1887         $(RM) $(>)
1888     }
1889
1890     actions Shell
1891     {
1892         copy $(>) $(<)
1893     }
1894 }
1895
1896
1897 #
1898 # Backwards compatibility with jam 1, where rules w
1899 #
```

```
1769
1770
1771 #
1772 # Backwards compatibility with jam 1, where rules w
1773 #
```

- **Show more context buttons:**

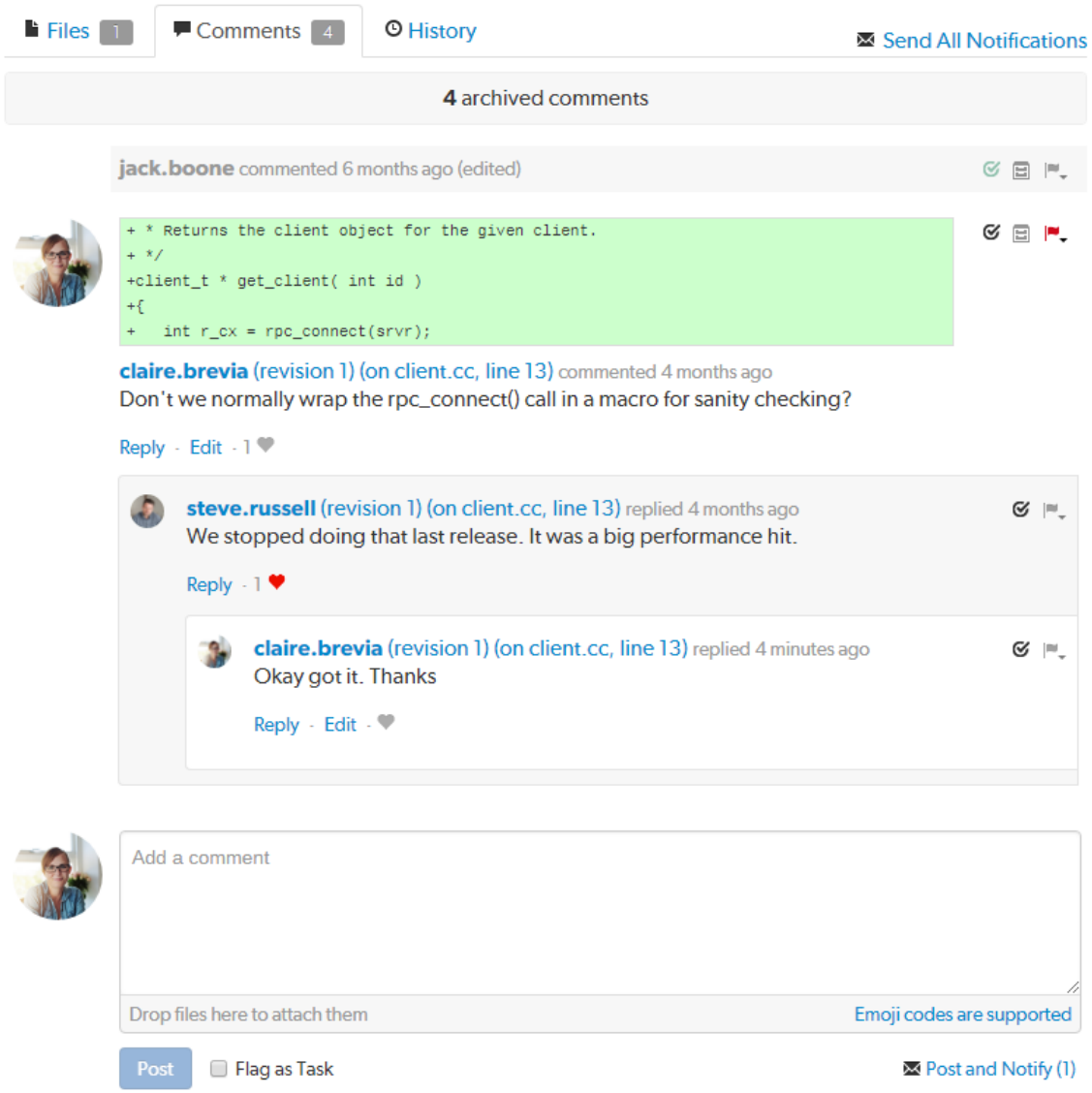
  Sometimes, the concise diff view needs to be expanded to fully understand the context of the change, use the **Show More** and **Show All** buttons to display extra lines around the change:

  > **Tip**
  > The following buttons are not available for stream specs.

  - **Show All Lines to Start of File**  button (only displayed for the first change in the file): click to show all of the lines up to the start of the file.
  - **Show More Lines for the Code Below** button: click to show 10 more lines above the change, the extra lines are displayed in the pane below the button.
  - **Show Entire Section**  button: click to show all of the lines between the changes that are above and below the button, the two changes and the lines between them are displayed in a single pane.
  - **Show More Lines for the Code Above** button: click to show 10 more lines below the change, the extra lines are displayed in the pane above the button.
  - **Show All Lines to End of File**  button (only displayed for the last change in the file): click to show all of the lines down to the end of the file.

Comments tab

The **Comments** tab is used to view all of the comments in the review.



Comment view

Comments are made up of the following elements (from left to right):

- **Commenter** avatar: the avatar of the user that made the comment, see "Avatars" on page 328.
- **Commenter** link: the username of the user that made the comment, click to see their user profile. For information about user profiles, see "Viewing another user's profile" on page 289.
- **Comment context** (only if the comment is made inline in a file): displays several lines of code before the line of code the comment is attached to. This helps makes sense of the comments should later changes remove those lines.

- **Revision** link (includes the file and line number if the comment is on a line in a file): click to go to the review revision the comment was made on. If the comment is made inline in a file the link will take you to that line in the file.
- **Mark comment as read** button (unread comments only): click to mark a comment as read, the comment will only be marked as read for you. For more information on marking a comment as read, see "Mark comments as read" on page 279.
- **Mark comment as unread** button (read comments only): click to mark a comment as unread, the comment will only be marked as unread for you. For more information on marking a comment as unread, see "Mark comments as unread" on page 279.
- **Archive** button (only available for root level comments): click to archive a comment and any replies to that comment. For more information about archiving and restoring comments, see "Archiving comments" on page 280 and "Restore comments" on page 281.
- **Tasks:** Flagging review comments as tasks is a lightweight workflow within a review that helps authors and reviewers prioritize review feedback. Any comment on a review can be flagged as a task, indicating to the review's author that the described issue needs to be addressed, and that the review is unlikely to be approved without a fix. For information about working with tasks, see "Tasks" on page 270.
  - **Not a Task** button: the comment has not been flagged as a task. Click to flag the comment as a task.
  - **Flagged as a Task** button: the comment has been flagged as a task. Click to confirm that the task has been addressed, or to remove the task flag from the comment.
  - **Task Addressed** button: the comment task has been addressed. Click to verify that the task has been addressed correctly, to verify and archive the task, or to reopen the task if it has not been addressed correctly.
  - **Task Verified** button: the comment task has been addressed and has been verified as correct. Click to reopen the task you think it has not been addressed correctly.
- **Comment** content: this can be text, a URL, or an attachment. For more information about the content of comments, see "Comment features" on page 273.
- **Reply** link: click to reply to the comment. For information about replying to comments, see "Reply to comments" on page 274.
- **Edit** link (only available for comments that you have made): click to edit the comment. For information about editing comments, see "Editing comments" on page 269.
- **Like** button (comments that you have not liked yet only): click to like the comment. The number to the left of the like button indicates how many users like the comment.
- **Unlike** button (liked comments only): click to unlike the comment.

Add a comment

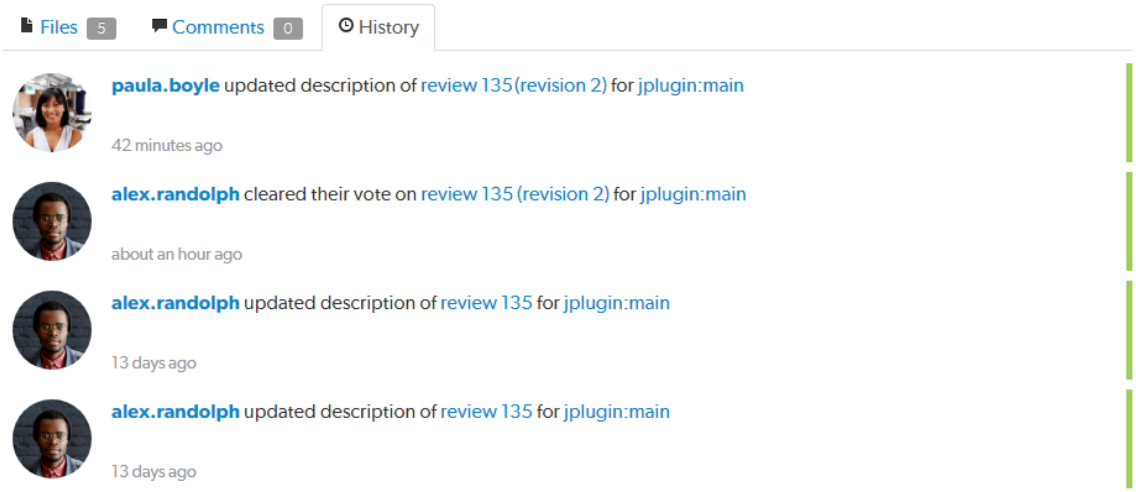
To add a comment, type your comment in the open comment text box at the bottom of the **Comments** tab page.

To reply to a comment, click the **Reply** button and type your comment in the comment text box.

- **Comment** text box: type your comment in the text box.
  - **URL links:** to add a URL link to a comment, type the URL into the comment text box. The URL is automatically made into a link. For information about how URL links are displayed in comments, see "Links in comments" on page 276.
  - **Attachments:** to attach a file to a comment, drag the file and drop it on the comment text box. For more information about attachments, see "Comment attachments" on page 276.
- **Flag as a Task** checkbox: select to flag the comment as a task. For more information about tasks, see "Flag a comment as a task" on page 270.
- **Post** button: click to post your comment. The comment is posted immediately but the comment notification is delayed, see "Comment notification delay" on page 273.
- **Post and Notify (x)** link: click to manually post your comment and send the comment notification immediately.
  Where **(X)** is the number of delayed comment notifications in the queue waiting to be sent, this number does not include the current comment you are working on.

History tab

The **History** tab presents a list of the events that affect this review.



Events include:

- When the review was started
- When a new reviewer joins the review

- When the review's state changes
- When the review's files are updated
- When a reviewer votes on the review
- When someone comments on the review, or one of its files
- When tests pass or fail, provided continuous integration is configured

Swarm user tasks

This section describes a number of the common Swarm user tasks:

Start a code review

Important
If your Helix server is configured as a commit-edge deployment, and your normal connection is to an edge server, Swarm refuses to start reviews for shelved changes that have not been promoted to the commit server.

Within Swarm, this means that the **Request Review** button does not appear for unpromoted shelved changes. Outside of Swarm, attempts to start reviews for unpromoted shelved changelists appear to do nothing. Ask your Helix server administrator for assistance if you cannot start a review.

An administrator of the Helix server can automatically promote shelved changes to the commit server by setting the configurable `dm.shelve.promote` to `1`.

Tip
If your changelist only contains a stream spec and its location in the Helix server is not associated with a Swarm project, the review that you create will not have any default reviewers or workflow rules. To associate the review with a project so that it has default reviewers and obeys the project workflow rules, include a file change from the project path in the changelist when you create the review.

1. Use the P4D command line or a P4D client to create the shelved or committed changelist.
2. Start a code review by using one of the following approaches:

Use Swarm:

1. Use Swarm to view a shelved or submitted changelist.

   > **Tip**
   > To view a shelved or submitted changelist, use a Quick URL. For example, if your change is **54321**, visit the URL: `https://myswarm.url/54321`.

2. Click the **Request Review** button to request a review of that changelist.

   Requesting a review on a shelved changelist uses the pre-commit model and requesting a review on a submitted changelist uses the post-commit model.

Use P4D command line:

When you are about to shelve or submit files:

1. Include `#review` within your changelist (separated from other text with whitespace, or on a separate line).

   Once the review begins, Swarm replaces `#review` with `#review-12345`, where `12345` is the review's identifier.

   > **Note**
   > The `#review` keyword is customizable. For details, see "Review keyword" on page 546.

2. At this time, you can add reviewers to the code review by using @mention for users, and @@mention for groups in the changelist description for each desired reviewer.

   If your `@mention` or `@@mention` includes an asterisk (`*`) before the *userid* or *groupid*, for example `@*userid`, that user or all of the group members become required reviewers. If your `@@mention` includes an exclamation mark (`!`) before the *groupid*, for example `@@!groupid`, the members of that group become required reviewers but only one member of the group is required to vote. See "Required reviewers" on page 380 for details.

3. Complete your shelve or submit operation.

   > **Warning**
   > If you shelve a changelist and subsequently edit the description to include `#review`, a review is **not started**. You must re-shelve the files after adding `#review`.

Use Git Fusion:

1. When you are using Git Fusion, you can start a review by pushing your changes to a target branch using the following command:

   ```
   $ git push origin task1:review/master/new
   ```

`task1` is the name of the current Git task branch, and ***`master`*** is the target branch that the proposed changes are intended for.

> **Important**
> The target branch must be mapped to a named Helix server branch in the Git Fusion repo configuration.
>
> See the Converting a lightweight branch into a fully-populated branch section of the *Git Fusion Guide* for details.

2. When the command completes, the output indicates the ***`review id`*** that has been created:

   ```
   remote: Perforce: Swarm review assigned: review/master/1234
   ```

   where ***`1234`*** is the review id that was just created.

   For more information on Git Fusion, see the *Git Fusion Guide*.

> **Tip**
> You can also start a Swarm review with P4V, P4VS, and P4Eclipse. See below for details:
>
> - **P4V:** see the Swarm integration features section of the *P4V User Guide*.
> - **P4VS:** see the Managing files chapter of the *P4VS User Guide*.
> - **P4Eclipse:** see the Reviewing changes chapter of the *P4Eclipse User Guide*.

> **Note**
> If you are using P4V and its Swarm integration, and you encounter the error **`Host requires authentication`**, ask your Helix Core server administrator for assistance. See "P4V Authentication" on page 477 for details.

Once a review has started

Wait for someone else to review your code, or see "Contribute comments or code changes to a code review" on page 67 More review activities are available.

Edit the reviewers on a review

A review author can edit the reviewers for a review by using the **Edit reviewers** button  on the review page. Reviewers are able to join or leave reviews, or to change whether their vote is required or optional.

> **Note**
> If the review includes content that is part of a project or branch with Retain default reviewers enabled, the following restrictions apply to the review:

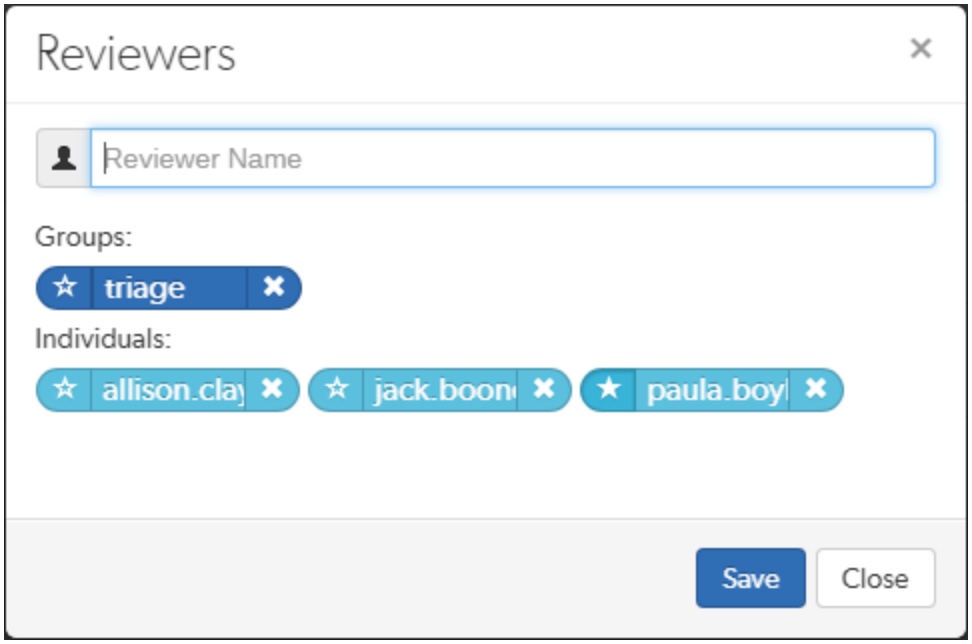
- The voting option for a retained default reviewer can only be changed to a stricter level, you cannot reduce the voting level.
- Retained default reviewers cannot be removed from the review.

Additionally, the following individuals can edit reviewers:

- Users with *admin* or *super* privileges.
- If the review is moderated, the moderators.
- If the review is part of a project, but not moderated, all project members.
- If the review is not part of a project, any authenticated user.

To edit reviewers for a review:

1. Navigate to a review.
2. Click the edit reviewers button , which appears just to the left of reviewer avatars.
   The **Reviewers** dialog is displayed.



3. Add or remove reviewers, or change the vote requirement.

   Use the reviewer search field to find users and groups by name, *userid*, *groupid*. The field auto-completes as you type, click on the user or group to add to the review.

   - **Groups:** click the star icon to the left of the *groupid* , and select whether the group is a required reviewer (one vote), a required reviewer (all votes), or an optional reviewer. A solid

star means that all group member votes are required to approve a review, a solid star with a 1 inside means at least one group member must vote up and no group members vote down to approve a review, and the outlined star means that the group vote is optional.

- **Users:** click the star icon to the left of the *userid* to toggle whether their vote is required or not. A solid star means that their vote is required to approve a review, whereas the outlined star means that their vote is optional.

Click the **X** icon to the right of the *userid* or *groupid* to remove that reviewer from the review.

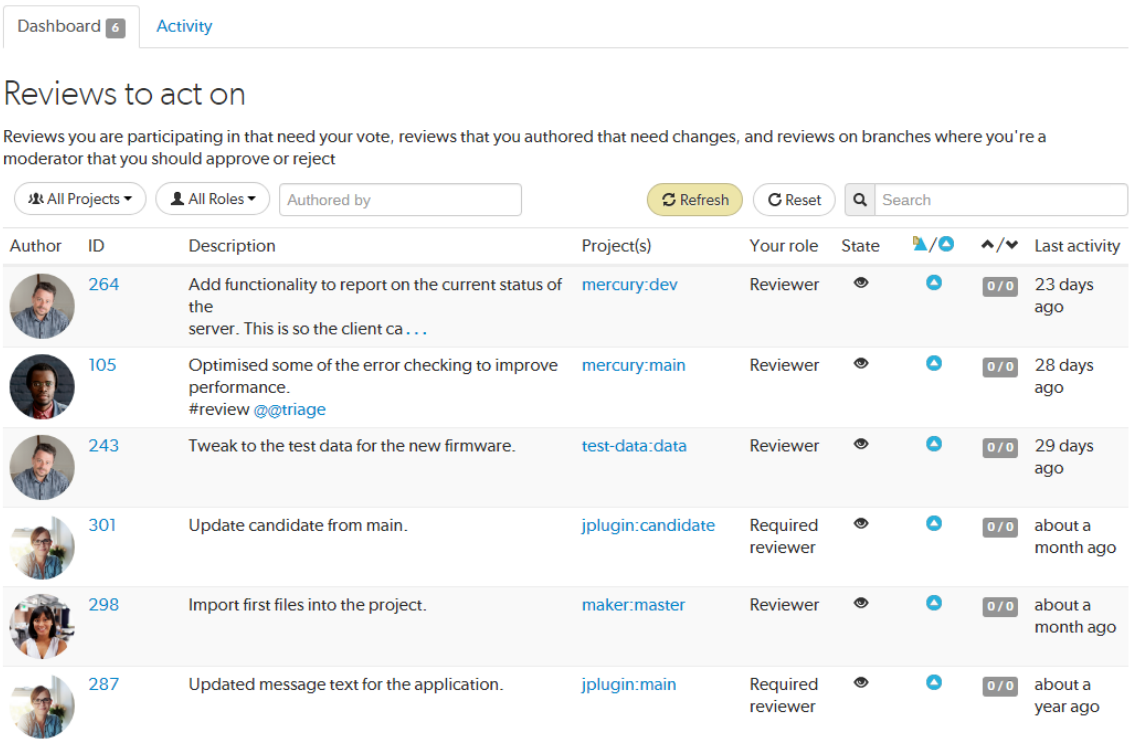
4. Click the **Save** button to save any changes made.

Check for code reviews you need to act on

Your dashboard is available by clicking the Swarm  icon above the main menu. Your dashboard displays a list of reviews that you may need to act on.

> **Note**
> Since it is tied to the logged in user, the dashboard is only populated if you are logged in.

1. Click the Swarm  icon above the main menu.
2. If the dashboard is not displayed, click the **Dashboard** tab.



3. By default all projects with reviews you need to act on are displayed.
4. To filter the reviews displayed by project, select **My Projects**, or a specific project from the **All Projects** drop down. The **All Projects** drop down will only show projects for which there are reviews in your dashboard.
5. By default all reviews you need to act on are displayed.
6. To filter by your role in a review, select **Author**, **Reviewer**, **Required reviewer**, or **Moderator** from the **All Roles** drop down. The **All Roles** drop down will only show roles for which there are reviews in your dashboard.
7. The **Refresh** button is displayed when new content is available for your dashboard. Click **Refresh** to update your dashboard with the new content.

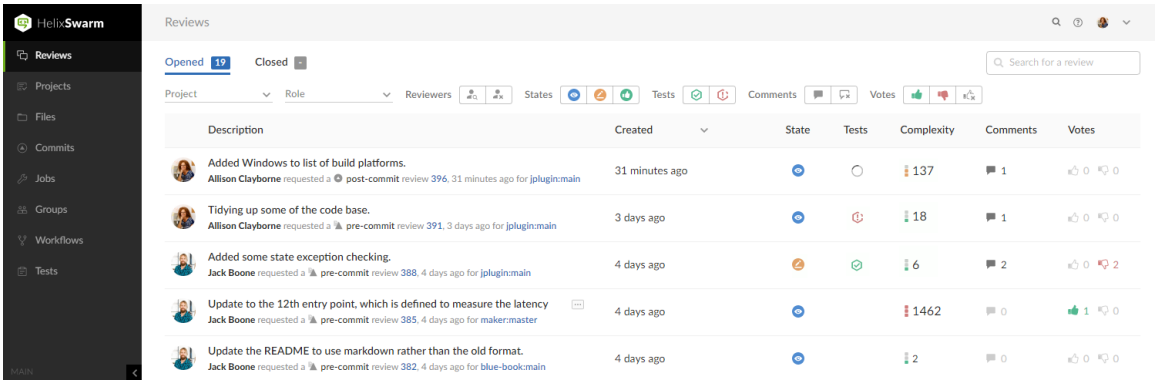
> **Note**
> By default, when your dashboard is open, Swarm checks for new content every five minutes.

Tip

- For more information on the dashboard page, see "Dashboard" on page 228.
- Click on a review **ID** number to display the code review.

List reviews you are involved with

1. Click **Reviews** in the main menu.



2. By default all projects are displayed.
3. To filter the reviews displayed by project, select **My projects**, or a specific project from the **Project** drop down.
4. By default all reviews are displayed.
5. To filter by your role in a review, select **Reviews I've authored**, **Reviews I'm participating in**, **Reviews I'm an individual reviewer of**, **Reviews I've authored or I'm participating in**, or search for reviews authored by a specific user from the **Role** drop down.
6. Click the **Opened** tab to view open reviews, click the **Closed** tab to view closed reviews. See "Reviews list" on page 342 for more information.

Contribute comments or code changes to a code review

You can contribute to a code review in the following ways:

- Comment on a review
- Comment on a review description
- Comment on a specific line in a file
- Comment on a file in a review
- Reply to a comment
- Edit files in a review
- Edit files in a review with Git Fusion

Comments

Comments are the primary feedback mechanism provided by Swarm. Review comments can be flagged as *tasks* for a lightweight workflow within a review that helps authors and reviewers prioritize review feedback. By default, comment notifications are delayed to allow you to add or edit comments as you progress through a review without sending a notification for each individual comment on the review. Comment notifications are rolled up into a single notification that you can either leave to be sent automatically, or you can send manually, see "Comment notification delay" on page 273.

> **Tip**
> For more information about comments and comment features, see "Comments" on page 267.

Comment on a review

1. From the review page: click **Comments** to view the **Comments** tab.
2. Add your comment in the text area.
3. **Optional:** select the **Flag as Task** checkbox to mark the comment as a task that needs to be addressed.
4. Click **Post**.

Comment on a review description

1. From the review page description area: click **Add a Comment** or **X Comments** where **X** is the number of description comments that already exist.
2. Add your comment in the text area.
3. **Optional:** select the **Flag as Task** checkbox to mark the comment as a task that needs to be addressed.
4. Click **Post**.

> **Tip**
> To hide the description comments, click **X Comments** (where **X** is the number of description comments that exist). To display the comments again, click **X Comments**.

Comment on a specific line in a file

1. From the review page: click **Files** to view the **Files** tab.
2. Click on the line you want to comment on.
3. Add your comment in the text area.
4. **Optional:** select the **Flag as Task** checkbox to mark the comment as a task that needs to be

addressed.

5. Click **Post**.

Comment on a file in a review

1. From the review page: click **Files** to view the **Files** tab.
2. If there are multiple files, click the file you want to comment on to expand its view.
3. Click the **Add a Comment** link in the footer of the file display.
4. Add your comment in the text area.
5. **Optional:** select the **Flag as Task** checkbox to mark the comment as a task that needs to be addressed.
6. Click **Post**.

Reply to a comment

1. From the review page: click **Comments** to view the **Comments** tab.
2. Navigate to the comment you are replying to.
3. Click the **Reply** link below the comment you are replying to.
4. Add your comment in the text area.
5. **Optional:** select the **Flag as Task** checkbox to mark your reply as a task that needs to be addressed.
6. Click **Post**.

Editing review files

Edit files in a review

1. Get a local copy of the review's files.

   > **Note**
   > If you are using Git Fusion, follow the steps in the next section.

2. Edit the files as required.
3. Prepare a changelist with the edited files and include `#review-1234` within the changelist's description (separated from other text with whitespace, or on a separate line), where `1234` is the review's identifier.

> **Warning**
> If you use an invalid review identifier, it will appear that nothing happens. Swarm is currently unable to notify you of this situation. If the review has not been correctly updated, use the **Add Change** button in the Swarm review heading to add the changelist to the review, see "Add a changelist to a review" on page 372.

4. Depending on the model of code review you are using, you would:
   - Shelve the files (for pre-commit).
   - Submit the files (for post-commit).

Edit files in a review with Git Fusion

> **Important**
> You can only update Git Fusion-initiated reviews using Git Fusion.

In the following example, the current local task branch is ***task1***, the target branch is ***master***, the review id is ***1234***, the Git Fusion hostname is ***gfserver***, and the remote repo name is ***p4gf_repo***.

1. Fetch the review's head version:

```
$ git fetch --prune origin
From gfserver:p4gf_repo
* [new_branch]      review/master/1234 ->
origin/review/master/1234
x [deleted]         (none)     -> origin/review/master/new
```

   The **--prune** option lets the local Git repo delete the unwanted review/master/new reference created by the initial **git push origin *task1*:review/*master*/new** command.

2. Check out the review's head version:

```
$ git checkout review/master/1234
```

3. Edit the files as required.
4. Add the edited files to the index of files, in preparation for the next commit.

   There are several ways to do this. For example, to add all modified files to the index, run:

```
$ git add -A
```

5. Commit the files in Git:

```
$ git commit -m "made some changes"
```

6. Push the Git changes to the review:

```
$ git push origin review/master/1234
```

> **Note**
> If you get review feedback that is better expressed as a Git rebase and cleaned up history, you can make your changes and push them as a new review.
>
> You cannot clean up history and then push your changes to the same review.

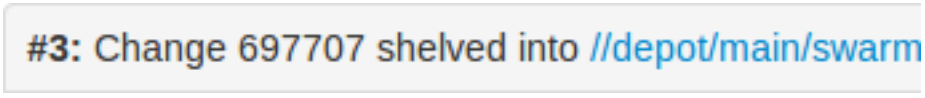
For more information on Git Fusion, see the *Git Fusion Guide*.

Get a local copy of the code in a review for evaluation

Swarm manages one or more changelists containing shelved copies of all of the files belonging to a specific review. You can unshelve the files to receive a copy of the review's code, or you can click the Download .zip button on the review to download a ZIP archive containing all of the review's files.

Determine the changelist that contains the files in the review

1. Visit the review's page.
2. The current review version's changelist appears in the file list heading.

   In this example, the changelist is **697707**. You use the identified changelist in place of *shelved changelist* below.

> **Note**
> Swarm can version file updates in reviews. For more information, see "Review display" on page 349.

Using P4

For a shelved changelist, use a command-line shell and type:

```
$ p4 unshelve -s shelved changelist
```

For a committed changelist, use a command-line shell and type:

```
$ p4 sync @committed changelist
```

> **Note**
> Your client's view mappings need to include the changelist's path.

Using P4V

For a shelved changelist:

1. Select **Search > Go To**.
2. Change the select box to **Pending Changelist**.
3. Type in the shelved changelist number and click **OK**.
4. Select the files in the **Shelved Files** area.
5. Right-click and select **Unshelve**.
6. Click **Unshelve**.

For a committed changelist:

1. Select **Search > Go To**.
2. Change the select box to **Submitted Changelist**.
3. Type in the submitted changelist number and click **OK**.
4. Select the files in the **Files** area.
5. Right-click and select **Get this Revision**.
6. Click **Close**.

Using Git Fusion

Git Fusion-initiated reviews include the Git logo beside the main review identifier. This indicator is important because Perforce users cannot update Git Fusion-initiated reviews.

In the following example, the current local task branch is `task1`, the target branch is `master`, the review id is `773273`, the Git Fusion hostname is `gfserver`, and the remote repo name is `p4gf_repo`.

1. Fetch the review's head version:

```
$ git fetch --prune origin
From gfserver:p4gf_repo
* [new_branch]      review/master/773273 ->
origin/review/master/773273
x [deleted]         (none)     -> origin/review/dev/new
```

   The `--prune` option lets the local Git repo delete the unwanted review/master/new reference created by the initial `git push origin task1:review/master/new` command.

2. Check out the review's head version:

```
$ git checkout review/master/773273
```

Important
You can only update Git Fusion-initiated reviews using Git Fusion.

For more information on Git Fusion, see the *Git Fusion Guide*.

Download files as a ZIP archive

The **Download .zip** button is used to download a ZIP archive that contains all of the files in the review. The version of the files downloaded is the most recent review revision selected in the review revision selector, see "Select review revisions to view" on page 361.

Note
The **Download .zip** button is not displayed if the zip command-line tool is not installed on the Swarm server. For information about installing, and configuring the zip command-line tool, see Zip archive.

When you click the **Download .zip** button, Swarm performs the following steps:

1. Scans the files/folders:
   - Checks that you have permission to access their contents, according to the Helix Core server protections.
   - Checks that the total file size is small enough to be processed by Swarm.
2. Syncs the file contents to the Swarm server from the Helix Core server.
3. Creates the ZIP archive by compressing the file content.
4. Starts a download of the generated ZIP archive.

Note
- You might not see all of the above steps; Swarm caches the resulting ZIP archives so that repeated requests to the same files/folders can skip the sync and compress steps whenever possible.
- If an error occurs while scanning, syncing, or compressing, Swarm indicates the error.

Fix errors that cannot be merged in a review

The problem can occur when you attempt to **Commit** or **Approve and Commit** via the Swarm UI and the shelved files are out of date.

Helix Swarm cannot currently help with resolving conflicts; you need to use a Helix server client such as p4 or P4V to resolve conflicts.

Resolve via P4

1. Acquire a local copy of the files.
2. Sync the files to the head version:

```
$ p4 sync
```

3. Begin resolving files with:

```
$ p4 resolve
```

4. Choose an appropriate option to resolve each file. For example:

```
$ p4 resolve
/home/bruno/bruno_ws/dev/main/jam/command.c - merging
//depot/dev/main/jam/command.c#9
Diff chunks: 4 yours + 2 theirs + 1 both + 1 conflicting
Accept(a) Edit(e) Diff(d) Merge (m) Skip(s) Help(?) e:
```

5. Re-shelve the resolved files with:

```
$ p4 shelve
```

> **Note**
> Ensure that the changelist description contains **`#review-12345`** (separated from other text by whitespace, or on a separate line), where **`12345`** is the identifier of the review you are updating.

> **Warning**
> If you use an invalid review identifier, it will appear that nothing happens. Swarm is currently unable to notify you of this situation. If the review has not been correctly updated, use the **Add Change** button in the Swarm review heading to add the changelist to the review, see "Add a changelist to a review" on page 372.

For more information, see "Resolve" in the *Helix Core Server User Guide*.

Resolve via P4V

1. Acquire a local copy of the files.
2. In P4V, right-click your workspace folder and select **Resolve Files**.

   The **Resolve** dialog appears.
3. Choose the appropriate options to resolve each file.
4. Right-click your workspace folder and select **Shelve Files**.

The **Shelve** dialog appears.

> **Note**
> Ensure that the changelist description contains **`#review-12345`** (separated from other text with whitespace, or on a separate line), where **`12345`** is the identifier of the review you are updating.

> **Warning**
> If you use an invalid review identifier, it will appear that nothing happens. Swarm is currently unable to notify you of this situation. If the review has not been correctly updated, use the **Add Change** button in the Swarm review heading to add the changelist to the review, see "Add a changelist to a review" on page 372.

5. Click **Shelve**.

For more information, see "Resolving Files" in the *P4V User Guide*.

View a list of Git-created reviews

You can view all reviews that were initiated in Git. First, you need to fetch any review branches that may exist:

```
$ git fetch --prune origin
```

Then you can list all branches, which includes any review branches, for the current Git Fusion repo:

```
$ git branch -a
task1
* master
remotes/origin/task1
remotes/origin/master
remotes/origin/review/task1/1234
remotes/origin/review/task1/1244
remotes/origin/review/task1/1347
remotes/origin/review/master/1235
remotes/origin/review/master/1236
remotes/origin/review/master/1358
```

> **Note**
> Git users cannot see Swarm reviews initiated in Helix server.

For more information on Git Fusion, see the *Git Fusion Guide*.

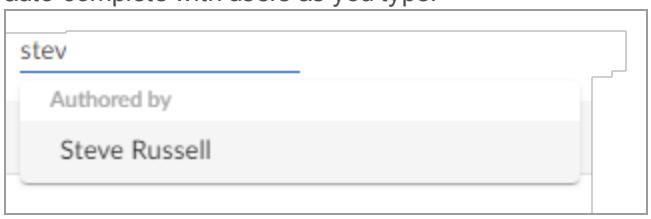
Change the author of a review

If the author of a review is no longer available, or ownership of a review is passed to a different developer, it is useful to be able to change the author of that review.

Note
By default you cannot change the author of a review, this option must be enabled by your Swarm administrator. See "Allow author change" on page 556 for details.

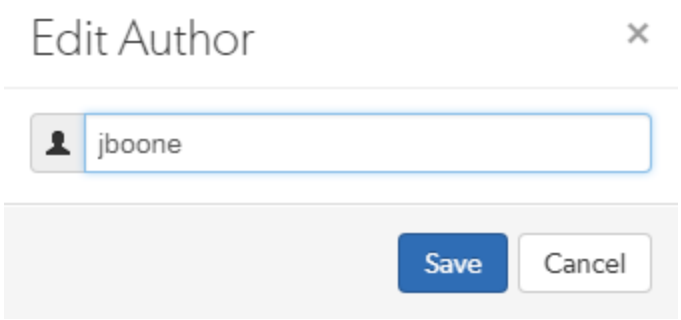
Tip
If you already have the review open in Swarm you can skip to step 5.

1. Click **Reviews** in the main menu.
2. Click the **Role** drop down and search for reviews by the author you want to replace. The list will auto-complete with users as you type.



3. Select the author from the list, reviews by that author are displayed.
4. Click the **Description** of the review you want to change the author on.
5. Click the **Author edit** icon and search for the new author name. The list will auto-complete with users as you type.

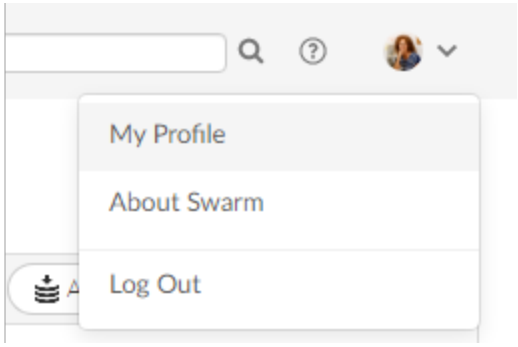
6. Select the new author name and click **Save**.



7. The review will display the new author's avatar in the author area of the review page.

Change your user notification settings

1. Click on your userid in the header and select **My Profile**.



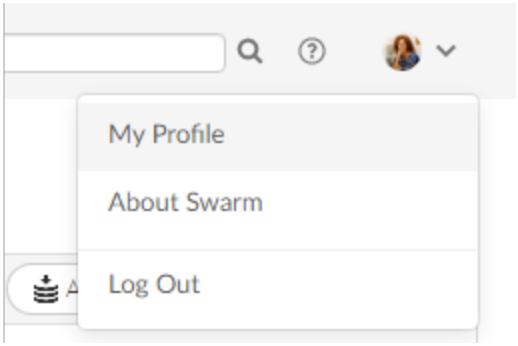
2. Click the **Notifications** tab to display your user notifications settings.
3. Configure which notifications you receive when events occur within Swarm, see the "Notifications tab" on page 287.

Unfollow all projects and users you are following

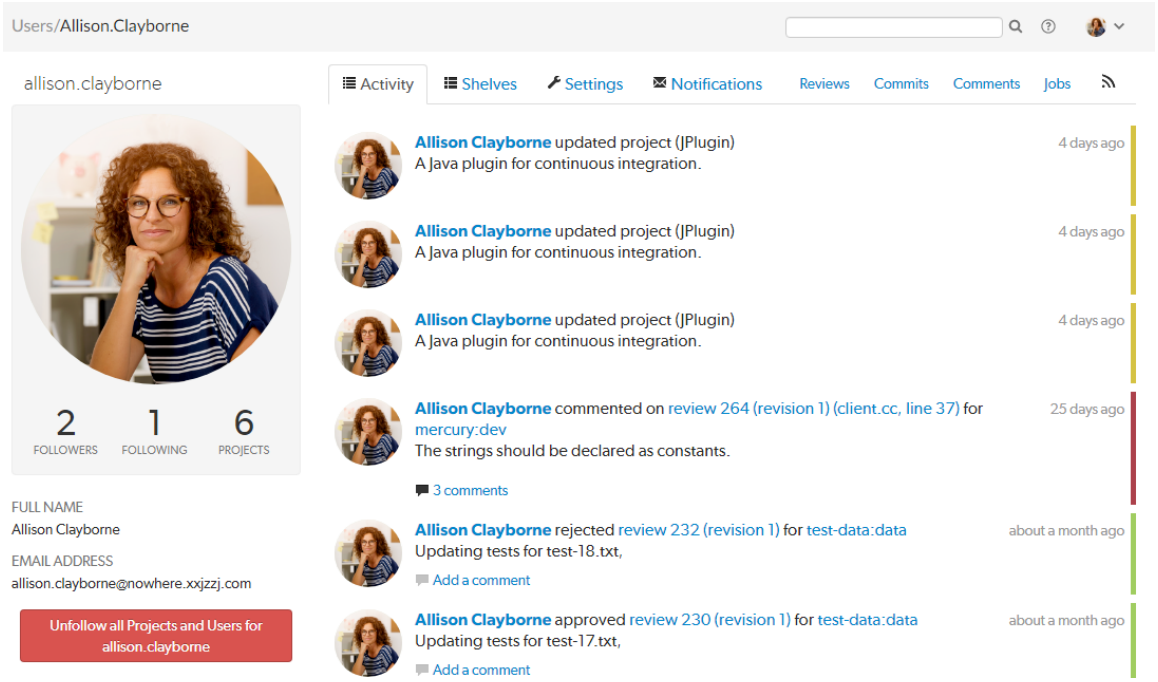
When you are viewing your own user profile, you can unfollow all projects and users that you are following.

> **Note**
> This action cannot be undone.

1. Display your own user profile when you are logged into Swarm by clicking on your *userid* in the header and selecting **My Profile**.



Your user page is displayed:



2. Click the **Unfollow all projects and users for *your-username*** button.
3. Click **OK** when the confirmation dialog is displayed to complete the unfollow action.
4. You will no longer be following any projects or users.

Swarm administration tasks

This section describes a number of the common Swarm administration tasks:

Change notification settings for a group

Important
You must be an owner of the group, have *super* privileges in Helix server (`p4d`), or have *admin* privileges in `p4d` version 2012.1 or later, to edit group notifications. If you do not have sufficient permissions, Swarm does not display the **Notifications** tab for the group.

1. Click **Groups** in the main menu.
2. Click the group you want to edit.
3. Click the group **Notifications** tab.

- If **Group mailing list** is **Disabled**, emails are sent to the group members individual email addresses and only the following notification options are available:
  - **Email members when a review is requested**: When any member of this group creates a review, this group will be notified.
  - **Email members when a change is committed**: When a change is committed into Perforce, if the owner of the changelist is a member of this group, this group will be notified.

    > **Tip**
    > When a user commits a changelist in Swarm, it is committed on behalf of the changelist owner. If the changelist owner is a member of this group, this group will be notified.

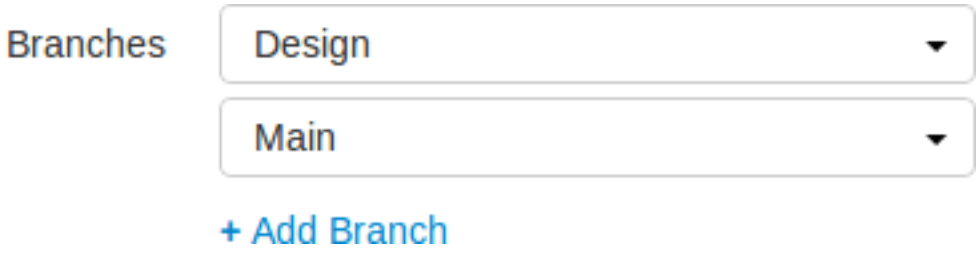
- If **Group mailing list** is **Enabled** notifications are sent to the group email address and additional notifications are available for the group, see group notifications for details.

Manage project branches

Initial steps:

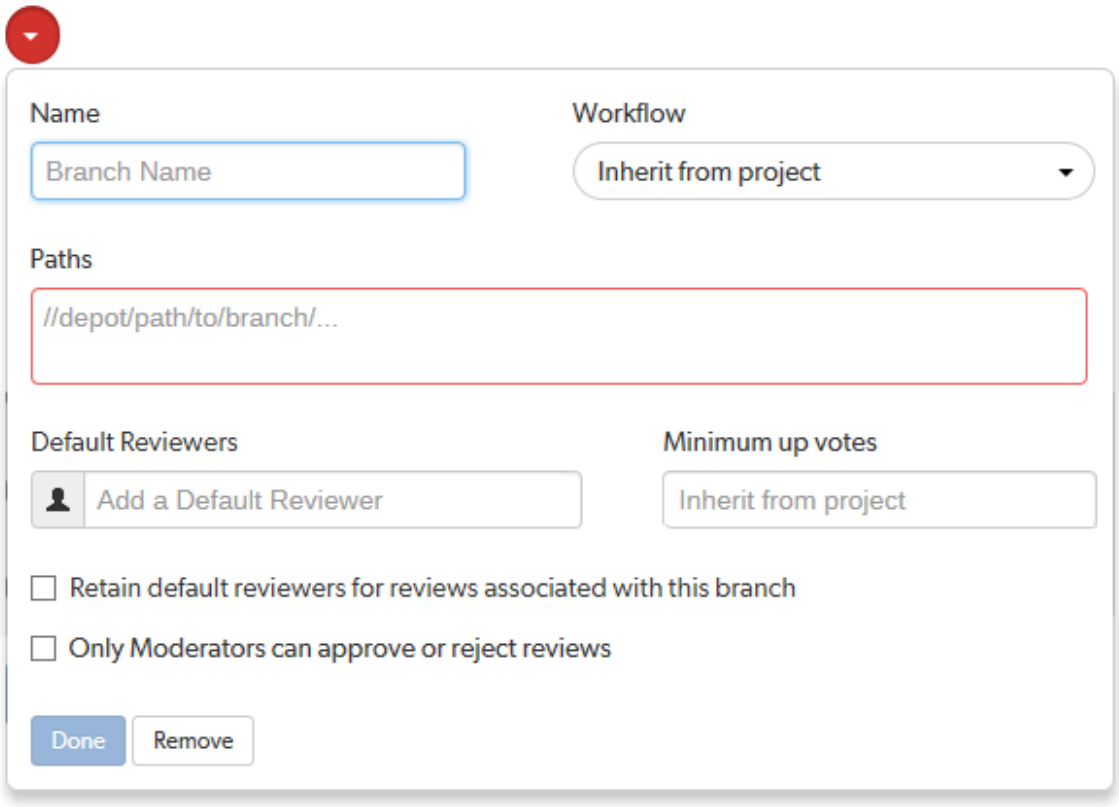
1. Visit your project page.
2. Click **Edit**.

   Next to the **Branches** label, a drop-down button for each branch is displayed, and **+ Add Branch**.



Adding a branch

1. Follow the initial steps.
2. Click **+ Add Branch** to display the branch drop-down dialog.



3. Enter a short **Name** for your branch.
4. **Optional** (not displayed if the Workflow feature is disabled): associate a workflow with the project branch.

   To use the parent project workflow, select **Inherit from project** from the **Workflow** dropdown list. This is the default when you create a new branch.

   To associate a workflow, select the workflow from the **Workflow** dropdown list, or enter the workflow name in the search field. The field auto-suggests workflows as you type. Only workflows that you own and shared workflows are shown in the dropdown list.

> **Tip**
> - When a workflow is associated with a project, the workflow is used for all of the branches in that project.

- When a project branch is associated with a workflow, the workflow of the parent project is ignored and the branch workflow is used.

For more information about workflows and how project workflows interact with branch workflows, see "Workflow basics" on page 425.

5. Enter one or more branch paths in the **Paths** field using depot syntax, one path per line.

Tip

- Branch paths, and files can be excluded by putting a minus symbol **-** at the start of the path.
- Branch paths are processed in order, starting with the first file path in the list.

For example:

```
//depot/main/swarm/...
-//depot/main/swarm/test/...
//depot/main/swarm/test/ResultSummary.html
```

The first path includes all of the directories and files under `//depot/main/swarm/` in the project branch.

The second path excludes all of the files in `-//depot/main/swarm/test/` from the project branch.

The third path includes the `ResultSummary.html` file from the previously excluded `//depot/main/swarm/test/` directory.

Note

- Wildcards should not be used; the only exception is that the branch path can end with the Helix server wildcard `...`
- Branch paths are case sensitive.
- Branch paths, and files are not checked to see if they are valid when you save the branch:
  - If you enter an invalid path ending in the wildcard `...`, the path will not be displayed in the project file browser until the path is created. This allows you to specify a path before it has been created.
  - If you enter a path that ends with a file that has not been committed, or a non-existent file, Swarm displays a 404 error when you navigate the path to the file with the project file browser.

Note
The Project Commits tab can fail to show some Helix server commits in the top level view. Individual branch views display the commits correctly:

The **Project Commits** tab top level client view is made up of all of the branches of the project.

For example, Project Alpha:

Branch: QA:
`//depot/alpha/dev/QA/...`

Branch: Dev :
`//depot/alpha/dev/...`
`-//depot/alpha/dev/QA/...`

The project commits tab view is generated by processing the branches in the order that they were created in and from top to bottom for the paths in each of those branches.
For the project Alpha example above:
The first path includes all of the directories and files under `//depot/alpha/dev/QA/` in the project commits tab top level view.
The second path includes all of the directories and files under `//depot/alpha/dev/` in the project commits tab top level view.
The third path excludes all of the directories and files in `//depot/alpha/dev/QA/` from the project commits tab top level view.

Result: commits made to `//depot/alpha/dev/QA/` that should be shown for the **QA** branch are not displayed in the **Project Commits** tab top level view.

When you have a simple branch structure this can be avoided by considering this issue when you create your branches. In the example above, creating the **Dev** branch first and then creating the **QA** branch avoids the problem because`//depot/alpha/dev/QA/` is not excluded from the **Project Commits** tab top level view.

6. **Default reviewers:**
   a. **Optional:** specify **Default Reviewers** for the project branch.

      This field auto-suggests users, and groups within Helix server as you type (up to a combined limit of 20 entries). Click on the user or group to add them as a default reviewer. Each time a new review is created, the default reviewers will be added to the review.

      - **Users:** click the star icon to the left of the *userid* to toggle whether their vote is required or not. A solid star means that their vote is required to approve a review, whereas the outlined star means that their vote is optional.

- **Groups:** click the star icon to the left of the *groupid* , and select whether the group is a required reviewer (one vote), a required reviewer (all votes), or an optional reviewer. A solid star means that all group member votes are required to approve a review, a solid star with a 1 inside means at least one group member must vote up and no group members vote down to approve a review, and the outlined star means that the group vote is optional.

Click the **X** icon to the right of the *userid* or *groupid* to remove that default reviewer from the default reviewers list.

Important
When a review is part of multiple projects/project branches:

- The default reviewer lists for all of the projects and project branches the review is part of are combined and added to the review.
- If a default reviewer has different reviewer options set on projects and project branches that the review is part of, the strictest reviewer option is used for the review.

  Example: A review is created and it is part of **Project A**, **Project B**, and **Project Branch b**.

  Project A: default reviewer X is an Optional reviewer

  Project B: default reviewer X is an Optional reviewer

  Project Branch b: default reviewer X is a Required reviewer

  Result: default reviewer X is added to the review as a Required reviewer

Note
If users or groups are @mentioned in a new changelist description that includes `#review`, they will be added to the review as reviewers. If any of these reviewers are already specified as default reviewers they will not be added to the review again, the reviewer's most restrictive reviewer option is used for the review.

Note
If a default reviewer is deleted from Helix server they will not be added to new reviews.

b. **Optional:** click the **Retain default reviewers for reviews associated with this branch** checkbox to prevent default reviewers being removed from reviews associated with this branch.

   For more information about retained default reviewers, see Retain default reviewers.

7. **Optional:** set the **Minimum up votes** required for reviews associated with this branch.

To inherit the parent project setting, leave **Minimum up votes** set to **Inherit from project**. This is the default when you create a new branch.

To use the branch setting and ignore the parent project setting, set **Minimum up votes** to **1** or more on the branch.

A review cannot be approved until all of the Required reviewers have voted up the review and the **Minimum up votes** specified has been satisfied.

- If a review spans projects/branches, the Minimum up votes for **each** of the projects and branches must be satisfied before you can approve the review.
- Required reviewers are included when up votes are counted.
- When **Count votes up from** is set to **Members** for a workflow associated with a project/branch, only the up votes of members of the project contribute to satisfying the **Minimum up votes** for a project/branch. For more information about the **Count votes up from** rule, see "Workflow rules" on page 423.

> **Important**
> If the Workflow feature is disabled, all votes are counted not just votes from project members.

> **Important**
> If the **Minimum up votes** required is set higher than the number of reviewers that exist for a review, approval will be blocked for that review. This is true even if all the reviewers on the review have voted up the review.

8. **Optional:** check the **Only Moderators can approve or reject reviews** checkbox.

   When checked, a field is displayed, allowing you to add a new moderator. The field auto-suggests groups and users within the Helix Core server as you type.

   If a group is specified as a moderator, all of the members of that group have the same moderator privileges for that project branch as if they were added individually.

   Once the branch specification is complete and the project has been saved, changing the state of any review associated with this moderated branch is restricted as follows:

   - Only moderators can approve or reject the review. Moderators can also transition a review to any other state.

   > **Important**
   > Moderators prevent the automatic approval of reviews, for more information about automatically approving reviews using workflow rules see "Workflow rules" on page 423.

> **Note**
> By default, when a review spans multiple branches that have different moderators, only one moderator from any one of the branches needs to approve the review.
>
> Swarm can be configured to require that one moderator from each branch must approve the review, this is a global setting. If a moderator belongs to more than one of the branches spanned by the review, their approval will count for each of the branches they belong to. For instructions on how to configure moderator behavior, see "Moderator behavior when a review spans multiple branches" on page 553.

- The review's author, when not a moderator, can change the review's state to **Needs Review**, **Needs Revision**, **Archived**, and can attach committed changelists.

  Normally, the review's author cannot change the review's state to **Approved** or **Rejected** on moderated branches. However, authors that are also moderators have moderator privileges, and may approve or reject their own review.

  When `disable_self_approve` is enabled, authors who are moderators (or even users with admin privileges) cannot approve their own reviews.
- Project members can change the review's state to **Needs Review** or **Needs Revision**, and can attach committed changelists. Project members cannot change the review's state to **Approved**, **Rejected**, or **Archived**.
- Users that are not project members, moderators, or the review's author cannot transition the review's state.
- For the review's author and project members, if a review is not in one of their permitted states, for example if the review's state is **Rejected**, they cannot transition the review to another state.

These restrictions have no effect on who can start a review.

9. Click **Done** to accept your branch specification.

   Once the branch definition has completed, if any moderators were specified, the number of moderators for that branch is displayed in the list of branches.
10. Click **Save** to save the branch changes to your project.

> **Note**
> The project name does not need to be included in the branch name; Swarm displays the project name with the branch name when appropriate.

Editing a branch

1. Follow the initial steps.
2. Click the branch drop-down button you want to edit.

3. Revise the **Name**, **Paths**, default reviewers, or moderators as required.
4. Click **Save**.

Removing a branch

1. Follow the initial steps.
2. Click the branch drop-down button you want to remove.
3. Click **Remove**.
4. Click **Save**.

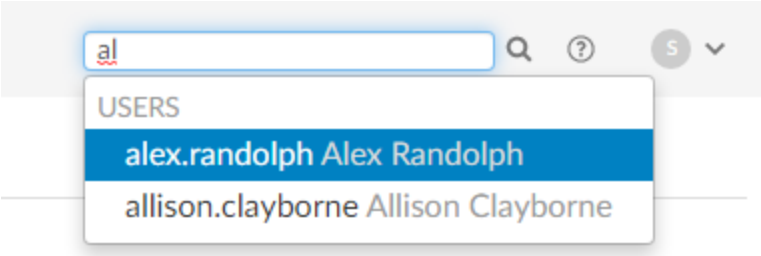
Unfollow all projects and users for another user

> **Note**
> You must be logged in as a user with *admin* or *super* user privileges and viewing the user's profile page to perform this action.

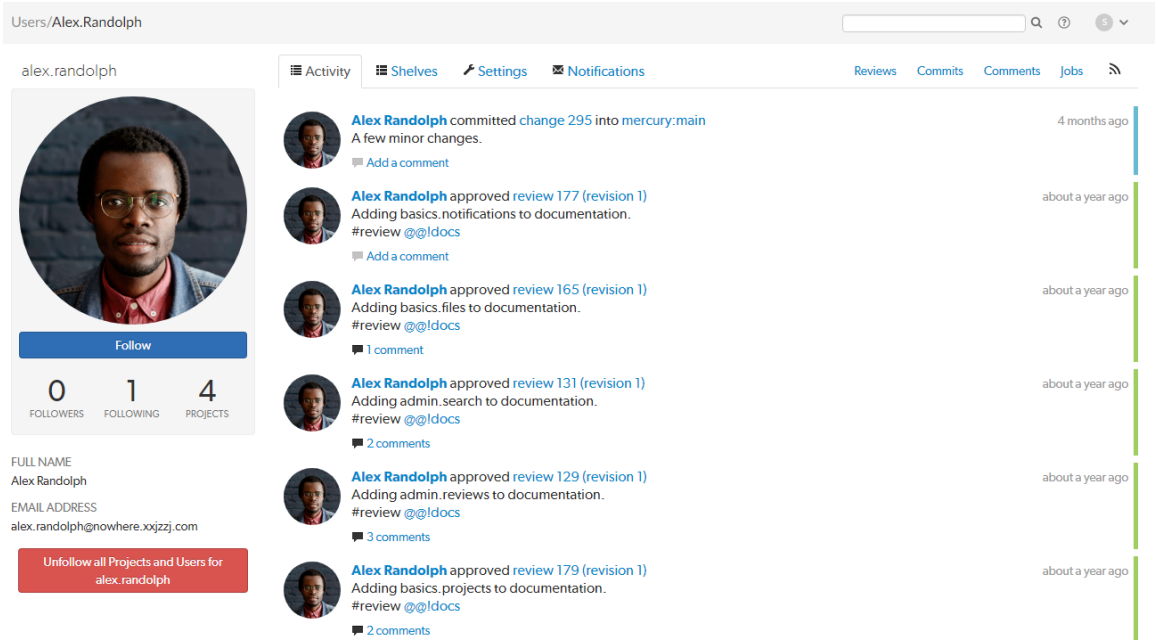
When a user has been removed from the Helix server but they are still following projects and users, it is useful to be able to remove all of their follows. This helps to keep the project and user follower lists up to date.

> **Note**
> This action cannot be undone.

1. Use the **Search** box in the header to search for the user, the search field will auto-complete as you type.



2. Select the user from the search results to display the user's profile:



3. Click the **Unfollow all projects and users for *username*** button located below the user's email address.
4. Click **OK** when the confirmation dialog is displayed to complete the unfollow action.
5. The user will no longer be following any projects or users.

Obliterate a review

> **Note**
> - By default, you must be a user with *admin* or *super* user rights to obliterate a review.
> - **Optional:** Swarm can be configured to allow users to obliterate reviews that they have authored. Configured by your Swarm administrator, see "Allow author obliterate review" on page 556.

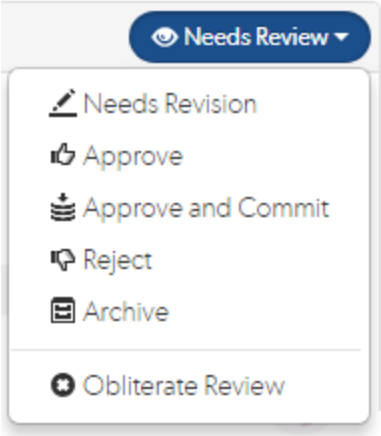
Obliterate is used to permanently delete reviews that have been created by mistake. For instance, if a review is associated with the wrong changelist, or a review contains sensitive information that should not be openly available.

For information on what happens to a review when it is obliterated, see "When you obliterate a review" on page 533.

> **Important**
> Obliterate must be used with care, the review and all of its associated metadata are permanently deleted. An obliterated review cannot be reinstated, not even by Perforce Support.

To obliterate a review:

1. Navigate to the review.
2. Click the **Review state** button and select **Obliterate Review**.



3. Click **Yes** on the confirmation dialog to complete the obliterate action.
4. The review is obliterated.

Change the logging level

Swarm logs various activities to the data/log file. Change the logging level to increase or decrease the volume of log data by editing a configuration file.

An example configuration, in the *SWARM_ROOT*`/data/config.php` file:

```
<?php
       // this block should be a peer of 'p4'
       'log' => array(
           'priority'  => 3, // 7 for max, defaults to 3
       ),
```

The maximum value for the log priority is 7; higher values do not result in increased logging. The minimum value is 0, which means no logging; lower values do not result in further logging reductions. For more information, see "Logging" on page 505.

Check the queue workers

Note

- Swarm package and OVA installations are automatically configured with a cron job to spawn workers.

- Swarm tarball installations should be setup with a cron job to ensure that workers are running to process events, this is part of the normal installation process. For information on setting up a cron job to spawn workers, see "Set up a recurring task to spawn workers" on page 174.

Helix Swarm uses a custom queue system to process events, provide notifications, and more. The queue system is required to handle the potentially large volume of events from a busy Helix server. A queue manager ensures that sufficient queue workers are available to process items.

You can check your queue workers in the following ways:

- If you do not have *admin* or *super* user permissions, see "HTTP request" below
- If you have *admin* or *super* user permissions, see "System information page" below

HTTP request

Check the status of the queue by making an HTTP request to **`/queue/status`**. The response is formatted in JSON and looks like this:

```
{"tasks":0,"futureTasks":1,"workers":3,"maxWorkers":3,"workerLifetime":"595s"}
```

This response indicates that the queue has no current tasks, there is 1 task scheduled for processing later, there are 3 queue workers available, at most 3 workers are created, and queue workers run for at most 10 minutes before self-terminating.

If the queue manager has stopped for some reason, start a new one by making an HTTP request to **`/queue/worker`**. No response is provided for this request.

System information page

The **Queue info** tab on the **System Information** page displays Swarm basic queue worker information. As well as displaying worker information you can also display the task queue, start a worker, and start a temporary worker.

To navigate to the **Queue Info** tab:

1. Click your **User id** avatar, to the right of the Swarm header.
2. Click **About Swarm**.
3. Click **System Information**.
4. Click the **Queue Info** tab.

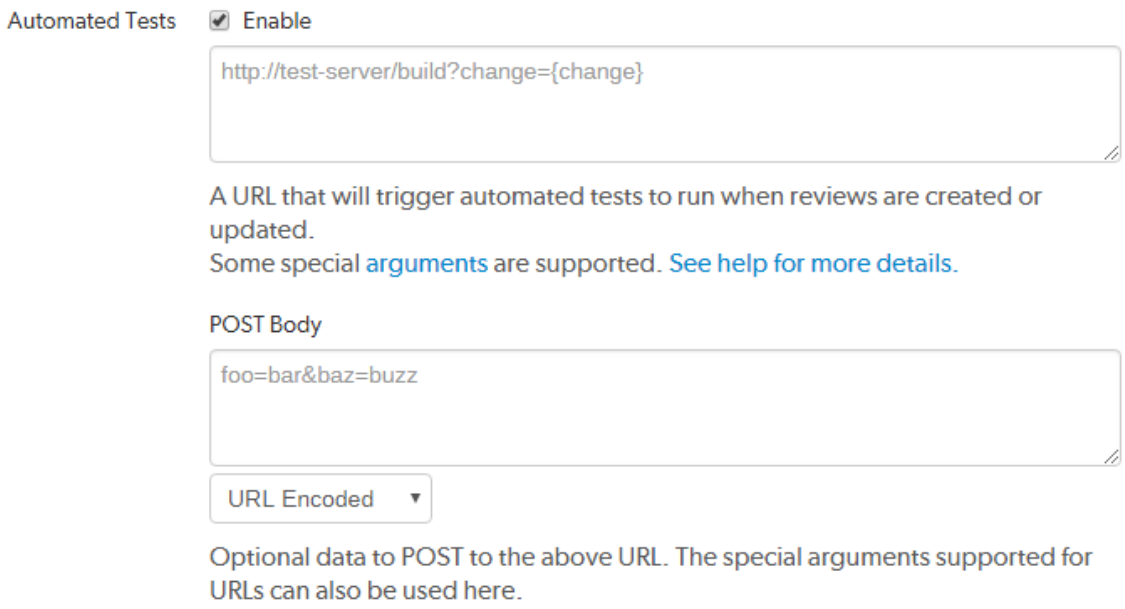
5. For information about the **Queue Info** tab, see "Queue Info" on page 583.

Integrate your test suite to inform review acceptance or rejection

> **Tip**
> From Swarm 2020.1, the preferred method for defining tests is on the Swarm "Tests" on page 440 page. This enables you to:
>
> - Associate a test with a workflow to ensure that the test is run when a review associated with that workflow is either created/updated or submitted.
> - Associate a test with the global workflow to ensure that the test is run whenever a Swarm review is either created/updated or submitted. This ensures that the global tests are enforced for all changes even if they are not part of a project.

Integrating Helix Swarm with a test suite involves enabling **Automated Tests** in your project's configuration and providing a *trigger URL*. When the *trigger URL* is requested, Swarm expects your test suite to be executed. When the tests complete, Swarm expects an *update callback URL*, a *pass callback URL*, or a *fail callback URL* to be requested by your test suite.

1. Navigate to the **Project** page.
2. Click the project **Settings** tab to display the **Project Settings** page.
3. Ensure that paths in each named branch configured for the project do not overlap with paths in other named branches.
4. **Automated tests** checkbox: select **Enable** to display the configuration fields:



5. Provide a URL that triggers your test suite execution.

   Special arguments are available to inform your test suite of various details from Swarm. Swarm automatically replaces the arguments with the relevant Swarm information when it calls the test:

> **Note**
> **Helix Plugin for Jenkins 1.10.11 and later:** Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Post Body** and select **URL Encoded**.

`{test}`
: The name of the test

`{testRunId}`
: The test run id

`{change}`
: The change number

`{status}`
: Status of the change, *shelved* or *submitted*

`{review}`

The review's identifier

{version}
The version of the review

{description}
The change description of the change used to generate this update. **{description}** cannot be used in the **URL**, it can only be used in the **POST Body**.

{project}
The project's identifier

{projectName}
The project's name

{branch}
The branch identifier(s) impacted by the review, comma-separated

{branchName}
The branch name(s) impacted by the review, comma-separated

{update}
The update callback URL. You can include any or all of the following when calling the update url to update the test run: status, messages, and a url in the body that links to the CI system for that run. They should be formatted in JSON in the body of the POST request. **Status:** valid status values are **running**, **pass**, and **fail**. **Messages:** you can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated. **{update}** is the preferred option for Swarm 2019.3. For more details, see the note below.

{pass}
Tests pass callback URL. From Swarm 2019.3, **{update}** is preferred. For more details, see the note below.

{fail}
Tests fail callback URL. From Swarm 2019.3, **{update}** is preferred. For more details, see the note below.

> **Note**
> - Swarm 2019.3 and later still supports **{pass}** and **{fail}**, however **{update}** is preferred because you can also include a message with the test status.
> - **{update}**, **{pass}**, and **{fail}** are composed automatically by Swarm. They include Swarm's own per-review authentication tokens.

6. **Optional:** specify any parameters that your automated tests require that must be sent via HTTP POST in the **POST Body** field. The POST parameters can include the special arguments listed above.

   Select the format of the POST parameters, either **URL Encoded** or **JSON Encoded**.

   - **URL Encoded:** POST parameters are parsed into name=value pairs.
   - **JSON Encoded:** parameters are passed raw in the POST body.

Configuring Jenkins for Swarm integration

Important
Your Jenkins host needs to be able to communicate with the Swarm host, and the Swarm host needs to be able to communicate with the Jenkins host. Ensure that the appropriate DNS/host configuration is in place, and that each server can reach the other via HTTP/HTTPS.

1. Install the p4-plugin for Jenkins:

   For instructions on installing the p4-plugin, see in the Installation chapter of the Helix Plugin for Jenkins Guide.

2. Configure a Jenkins project:

   Note
   Helix Plugin for Jenkins 1.10.11 and later: Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Post Body** and select **URL Encoded**.

   a. Specify the job name so that it matches the project identifier used in the trigger URL, as defined below.

      For example, the computed value of `{projectName}_{branchName}`.

      Or, edit the trigger URL to use the Jenkins job name you specify.

   b. Make the build parameterized to accept these parameters (note that these are named to match up with the script that is called):

      `{test}`
      The name of the test
      `{testRunId}`
      The test run id
      `{status}`
      Whether the changelist to be tested is *shelved* or *submitted*
      `{change}`
      Changelist # to run tests against
      `{review}`
      The review's identifier
      `{version}`
      The version of the review
      `{branchName}`
      The branch name(s) impacted by the review, comma-separated
      `{update}`
      The URL to POST to update the test run. You can include any or all of the following when calling the update url to update the test run: status, messages, and a url in the body that links to the CI system for that run. They should be formatted in JSON in the body of the POST request. **Status:** valid status values are **`running`**, **`pass`**, and

`fail`. **Messages:** you can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated. **`{update}`** is the recommended way of reporting test status.

Important
If your test system cannot POST to Swarm, you cannot use **`update`** and you must use **`pass`** and **`fail`** instead.

Note
When using {update}:

- If your build script has access to any messages related to the test execution, pass the messages to Swarm using the **`{update}`** URL. Swarm uses the provided message(s) to add to the test results.
- If your build script has access to the results of test execution, include a POST parameter called **`url`** when calling the update URL. Swarm uses the provided url to link reviews to the test results. Valid test status values are **`running`**, **`pass`**, and **`fail`**.
- The **`{update}`** callback url accepts a JSON body where you can specify any or all of the following: messages, url, and status.

```
{
    "messages" : ["My First Message", "My Second
Message"],
    "url" : "http://jenkins_host:8080/link_to_run",
    "status": "pass"
}
```

`{pass}`
The URL to wget if the build succeeds. From Swarm 2019.3, **`{update}`** is preferred. For more details, see the note below.

`{fail}`
The URL to wget if the build fails. From Swarm 2019.3, **`{update}`** is preferred. For more details, see the note below.

Note
When using `{pass}` and `{fail}`: if your build script has access to the results of test execution, include a GET or POST parameter called **`url`** when calling the pass or fail URLs. Swarm uses the provided url to link reviews to the test results.

> **Tip**
> Swarm 2019.3 and later still supports `{pass}` and `{fail}`, however `{update}` is preferred because you can also include a message with the test status.

c. Select **Perforce Software** for the **Source Code Management** section.

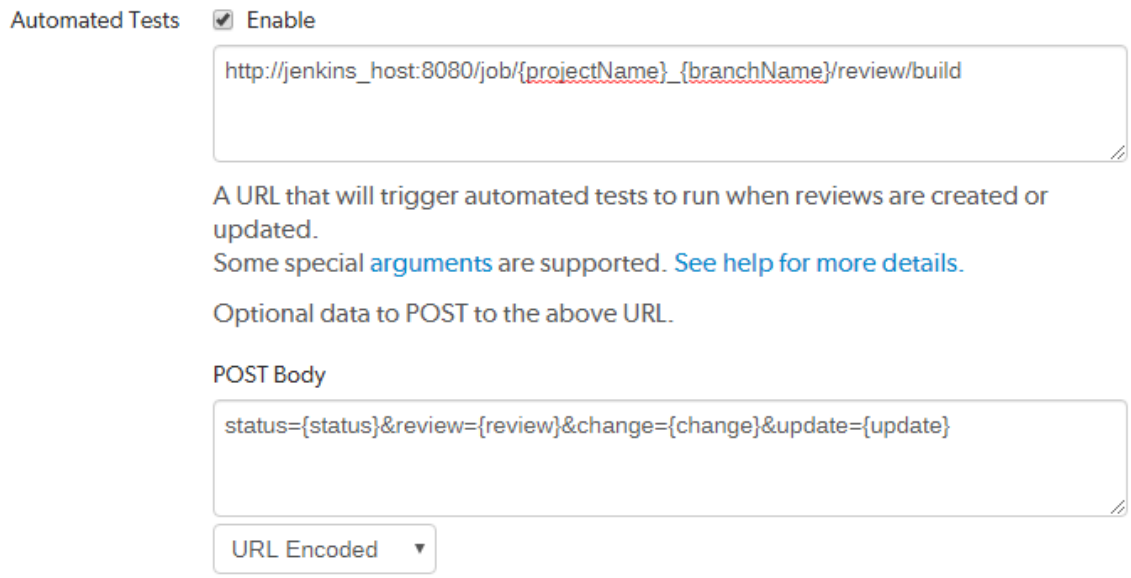
> **Important**
> You might see **Perforce** in the **Source Code Management** section. This represents an earlier community-provided Perforce plugin that does not include support for Swarm.

d. Set up credentials and workspace behavior as needed.

For instructions on configuring credentials and workspaces, see the Credentials and Workspaces sections of the Helix Plugin for Jenkins Guide.

> **Important**
> - If your Helix server is configured for Helix Authentication Service, the service user credentials used for automated testing must not use Helix Authentication Service.
> - The client workspace configured in Jenkins must have a view that includes the paths defined for that branch in Swarm.

3. Configure your Swarm project to run automated tests with a URL and parameters like this:



Important
For Jenkins, the job name needs to match the job identifier in the URL. In the example above, this is the computed value of **`{projectName}_{branchName}`**.

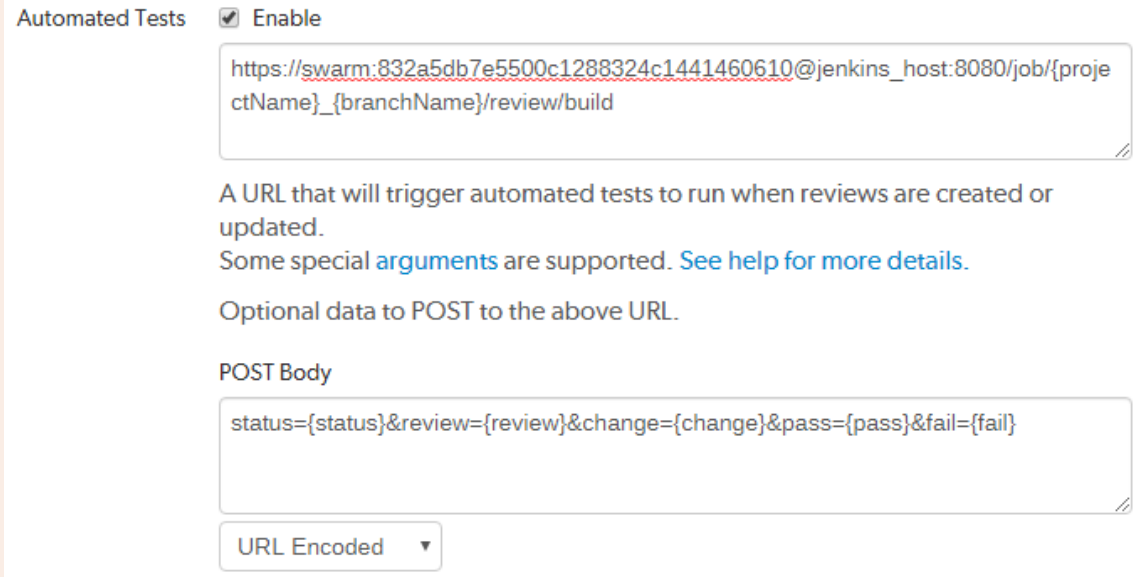
If you prefer a different naming scheme in Jenkins, replace **`{projectName}_{branchName}`** in the URL above with the project name actually defined in Jenkins.

Important
If security is enabled in Jenkins, the trigger URL needs to include credentials. Follow these steps:

a. Create a Jenkins user that will trigger Swarm builds. For example swarm.
b. Log into Jenkins as the new user.
c. Click on the user's username in the Jenkins toolbar.
d. Scroll down to **API Token**.
e. Click **Show API Token**.
f. Incorporate the value of the **API Token** into the Swarm trigger URL.

   For example, if the username is swarm and the API Token value is 832a5db7e5500c1288324c1441460610, the Swarm trigger URL and parameters should

be:

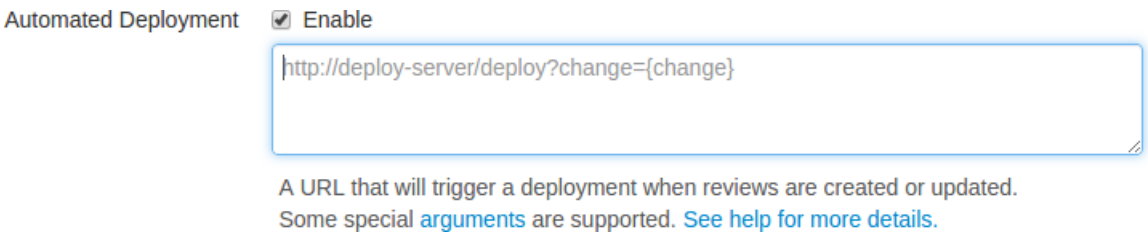


Automatically deploy code within a review

Deploying code in a code review automatically involves enabling **Automated Deployment** in your project's configuration and providing a *trigger URL*. When the *trigger URL* is requested, Swarm expects a deployment program to be executed.

When the deployment processing ends, Swarm expects either a *success callback URL* or *failure callback URL* to be requested by your deployment program. These callback URLs should include a url parameter (either via GET or POST); when a valid-looking URL is included, clicking the deployment status indicator directs the user to the specified URL. This is intended to facilitate easy viewing of the successfully deployed review, or a report indicating why the deployment failed. The url parameter is mandatory for successful deployments, but is optional for failures.

1. Navigate to the project page.
2. Click the project **Settings** tab to display the **Project Settings** page.
3. **Automated Deployment** checkbox: select **Enable** to display the configuration field:



4. Provide a URL that triggers your deployment execution.

   Special arguments are available to inform your deployment program of various details from Swarm:

   > **Note**
   > **Helix Plugin for Jenkins 1.10.11 and later:** Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Post Body** and select **URL Encoded**.

   `{change}`
   : The change number

   `{status}`
   : Status of the change, *shelved* or *submitted*

   `{review}`
   : The review's identifier

   `{project}`
   : The project's identifier

   `{projectName}`
   : The project's name

   `{branch}`
   : The branch identifier(s), comma-separated

   `{branchName}`
   : The branch name(s), comma-separated

   `{success}`
   : Deployment successful callback URL

   `{fail}`
   : Deployment failure callback URL

3 | Install and upgrade Swarm

This chapter covers the initial installation and configuration of Swarm as well as upgrading an existing Swarm installation.

Important restrictions

Do not prefix group names, project names, user names, or client-names with "*swarm-*", this is a reserved term used by Swarm. Prefixing a name with "*swarm-*" will result in unexpected and unwanted behavior in Swarm.

For example:

Prefixing a group name with "*swarm-project-*" will result in, but is not limited to, the following issues:

- Swarm notifications will not be processed correctly for the group.
- The group will not be visible in Swarm.

Review the runtime dependencies

First, review the runtime dependencies before you install Swarm, see "Runtime dependencies" on page 101.

Swarm and Helix server installation considerations

Before installing Swarm and Helix server you should consider the following:

- For a small system, you can run Swarm and Helix server on the same machine.
- For larger systems, we recommend that Swarm and Helix server are run on separate machines. The machines should be close to each other to maximize network performance.

  Triggers are installed on the Helix server machine and need to talk to Swarm. Trigger performance can be negatively affected if network lag between Swarm and the Helix server is high.
- The Swarm and Helix server machines do not need to have the same operating system. For example, Helix server could be on a Windows server and Swarm could be on a CentOS server.

> **Note**
> You cannot install Swarm on a Windows machine.

Choose the installation process

Once you have reviewed the "Runtime dependencies" on the facing page and know that you can satisfy them, there are a number of ways to install Swarm.

> **Note**
> We recommend the package installation method to install Swarm whenever possible, see "Install and configure Swarm from a package" on page 115. Package installs ensure that all of the Swarm dependencies are installed and this is the easiest way to install Swarm. For a list of recommended operating systems for Swarm, see "Recommended operating systems" on page 102.

Choose one of the following installation methods (we recommend the package installation method whenever possible):

- **Swarm RPM or Debian packages (recommended):**
  1. Follow the steps provided in "Install and configure Swarm from a package" on page 115.
  2. Establish a trigger token, see "Establish trigger token" on page 158.
  3. Configure Helix Core server for Swarm, see "Helix Core server configuration for Swarm" on page 158.
  4. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
  5. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.
  6. You are now all set to start using Swarm. Enjoy!

     > **Tip**
     > To get started with Swarm, see the "Quickstart" on page 29 chapter.

- **Swarm.ova (Open Virtualization Appliance):**
  1. Follow the steps provided in "Deploy and configure a Swarm VM from an OVA" on page 137.
  2. Establish a trigger token, see "Establish trigger token" on page 158.
  3. Configure Helix Core server for Swarm, see "Helix Core server configuration for Swarm" on page 158.
  4. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
  5. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.
  6. You are now all set to start using Swarm. Enjoy!

> **Tip**
> To get started with Swarm, see the "Quickstart" on page 29 chapter.

- **Swarm.tgz (Tarball):**
  1. Follow steps provided in the "Install and configure Swarm manually from a Tarball" on page 142.
  2. Configure Redis, see "Redis configuration" on page 144.
  3. Configure Apache, see "Apache configuration" on page 148.
  4. Configure PHP, see "PHP configuration" on page 151.
  5. Configure Swarm, see "Swarm configuration" on page 154.
  6. Establish a trigger token, see "Establish trigger token" on page 158.
  7. Configure Helix Core server for Swarm, see "Helix Core server configuration for Swarm" on page 158.
  8. Set up a recurring task to spawn workers, see "Set up a recurring task to spawn workers" on page 174.
  9. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
  10. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.
  11. You are now all set to start using Swarm. Enjoy!

> **Tip**
> To get started with Swarm, see the "Quickstart" on page 29 chapter.

Upgrade Swarm

If you already have a working Swarm installation and you want to upgrade Swarm to a newer release, see "Upgrade Swarm" on page 193.

Runtime dependencies

In order to successfully install, configure, and deploy Swarm, the following dependencies are required:

- A recommended operating system, see "Recommended operating systems" on the next page
- An Apache server with the modules required by Swarm installed, see "Apache web server" on page 103

- A supported version of PHP with the modules required by Swarm installed, see "PHP" on page 105.
- A supported Helix Core server deployment, and the ability to connect to it from the system hosting Swarm, see "Helix Core server requirements" on page 107

  > **Note**
  > "Helix Core server deployment" can refer to a running `p4d` or a proxy, replica, edge server, or commit server.

- `curl` or `wget` for Swarm worker operation, see "Worker dependencies" on page 110
- A supported version of perl to integrate with Helix Core server triggers, see "Trigger dependencies" on page 109
- If Security-enhanced Linux (SELinux) is installed it must be configured correctly, see "Security-enhanced Linux (SELinux)" on page 112

Optional dependencies:

- **LibreOffice:** required to view office-type documents. For more information, see "LibreOffice" on page 112.
- **zip:** command-line archiving tool: Required to download zip archives of files and folders. For more information, see "Zip" on page 112.
- **Helix Authentication Service SSO:** enables Swarm to authenticate with SSO when the Helix Core server is configured for Helix Authentication Service, see "Helix Authentication Service SSO" on page 112.
- **Sendmail or equivalent:** required to use Swarm email notifications. For more information, see "Swarm email notifications" on page 112.

Recommended operating systems

Perforce recommend that Swarm is installed on one of the following operating systems:

- **Ubuntu 16.04 LTS, 18.04 LTS, and Ubuntu 20.04 LTS**
- **CentOS/RHEL 7 and 8** latest point release with PHP libraries installed from the Remi repository, see "PHP" on page 105. At the time of the Swarm 2020.2 release, the latest point releases are 7.8 and 8.2.

> **Note**
> You cannot install Swarm on a Windows machine.

Other Linux distributions

> **Important**
> Swarm has not be tested on Linux distributions that not on our recommended list. Because of this, our Support team may not be able to help you to the same extent as they would for the recommended operating systems.

Swarm will probably run on any Linux distribution with kernel version 2.6+ Intel (x86_64) with glibc 2.11+. Swarm includes binary versions of P4PHP, the Perforce extension for PHP. This P4PHP dependency typically sets the limit to what Linux distributions you can install Swarm on.

You might be able to get Swarm running on another platform if you build P4PHP yourself and satisfy the other runtime dependencies. Instructions on how to obtain and build P4PHP from source can be found in the P4PHP Release notes.

> **Important**
> P4PHP does not support threaded operation. If you compile P4PHP from source, ensure that the version of PHP you compile for is non-threaded.

Apache web server

Swarm requires Apache HTTP Server 2.4 or newer:

- https://httpd.apache.org

> **Important**
> **CentOS/RHEL only:**
>
> - **CentOS 7:** Use the Remi repository configuration package (`remi-release-7.rpm`) to give Swarm access to PHP 7.x. The `sclo-php7x-php-pecl-redis` and `sclo-php7x-php-pecl-igbinary` extensions are not available from this RedHat repository and you must source them from elsewhere.
> - **RHEL 7:** Use the Remi repository configuration package (`remi-release-7.rpm`) to give Swarm access to PHP 7.x and most of the required PHP extensions. The `php7x-php-pecl-redis` and `php7x-php-pecl-igbinary` extensions are not available from this RedHat repository and you must source them from elsewhere.
> - **CentOS/RHEL 8:** Use the Remi repository configuration package (`remi-release-8.rpm`) to give Swarm access to PHP 7.x. Use the `epel-release-latest-8.noarch.rpm` repository configuration package to give Swarm access to EPEL packages.

- **Swarm 2019.1 to 2020.1:** these versions of Swarm used PHP packages from the SCL repositories for CentOS/RHEL 7. This was to provide access to more recent versions of PHP than are available as standard on CentOS/RHEL. This required the use of the **`httpd24-httpd`** package for Apache. These were all installed into **`/opt/rh`**

  Swarm 2020.2: this version of Swarm uses the Remi repository for CentOS/RHEL 7 and 8. This provides PHP 7.4 installed in the standard file system structure. This means that the old **`httpd24-httpd`** version of Apache is no longer needed, and the standard system version of Apache is being used again.

  The SCL Apache site configuration file was installed at this location for Swarm 2019.1 to 2020.1:

  `/opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf`

  If this exists when Swarm is upgraded to 2020.2, this file is copied to **`/etc/httpd/conf.d/perforce-swarm-site.conf`** if there is no file at the destination. It is also re-written to change references from **`/var/log/httpd24`** to **`/var/log/httpd`**

  If a site configuration file for Swarm already exists in **`/etc/httpd`**, the copy and re-write is not performed.

  After upgrade, **`httpd24-httpd`** is disabled.
- To avoid seeing the Apache HTTP server Linux test page when you start the Apache server, comment out the content of the **`welcome.conf`** file located in the **`/etc/httpd/conf.d/`** directory.

Swarm also requires the following Apache modules:

- **Ubuntu:** **`mod_phpx`** for interacting with PHP (usually installed with PHP)

  Where **`x`** is the version of PHP you are running, for example 7. For supported versions of PHP, see "PHP" on the facing page.
- **CentOS/RHEL:** **`php-fpm`** for interacting with PHP (usually installed with PHP)
- **`mod_rewrite`** URL rewriting engine

  http://httpd.apache.org/docs/2.4/mod/mod_rewrite.html

Important
Only the **`prefork`** MPM is supported. Use of the worker or event MPMs is not supported and is likely to cause problems because P4PHP does not support threaded operation.

For more information on the prefork MPM, see: https://httpd.apache.org/docs/2.4/mod/prefork.html

PHP

Swarm requires PHP 7.0, 7.1, 7.2, or 7.4:

- https://secure.php.net

> **Important**
> - PHP must be non-threaded because P4PHP does not support threaded operation.
> - **CentOS 7:** Use the Remi repository configuration package (**`remi-release-7.rpm`**) to give Swarm access to PHP 7.x. The **`sclo-php7x-php-pecl-redis`** and **`sclo-php7x-php-pecl-igbinary`** extensions are not available from this RedHat repository and you must source them from elsewhere.
> - **RHEL 7:** Use the Remi repository configuration package (**`remi-release-7.rpm`**) to give Swarm access to PHP 7.x and most of the required PHP extensions. The **`php7x-php-pecl-redis`** and **`php7x-php-pecl-igbinary`** extensions are not available from this RedHat repository and you must source them from elsewhere.
> - **CentOS/RHEL 8:** Use the Remi repository configuration package (**`remi-release-8.rpm`**) to give Swarm access to PHP 7.x. Use the **`epel-release-latest-8.noarch.rpm`** repository configuration package to give Swarm access to EPEL packages.

Required PHP extensions

Swarm requires the following PHP extensions:

> **Tip**
> If you install Swarm from a package, all of these PHP extensions are automatically pulled in as dependencies on all platforms.

- **`iconv`** (character encoding converter)
  https://secure.php.net/iconv

  This is typically enabled by default with most PHP distributions
- **`JSON`** (JavaScript Object Notation)
  https://secure.php.net/json

  This is typically enabled by default with most PHP distributions, although recent distributions are making this optional.
- **`Session`** (session handling)

  This is typically enabled by default with most PHP distributions

- **`P4PHP`** version 2019.1 or later (the Perforce PHP Extension)

  The latest P4PHP version is included with the Swarm package, OVA and tarball installations.

  > **Note**
  > **Swarm package, OVA and tarball installations:** 2 versions of P4PHP are supplied for each PHP 7 version supported by Swarm. They are located in the **`p4-bin/bin.linux26x86_64`** directory.
  >
  > - **`perforce-php7x.so`** compatible with systems using SSL 1.0.2
  > - **`perforce-php7x-ssl1.1.1.so`** compatible with systems using SSL 1.1.1 (by default Ubuntu 18.04 and Ubuntu 20.04 use SSL 1.1.1)
  >
  > Where **`x`** is the version of PHP 7.
  >
  > If the **`perforce.ini`** file is not pointing at the correct version of P4PHP and you connect to an SSL enabled Helix server:
  >
  > - The Swarm web-page will not load and you might see a **`Connection Reset`** error.
  > - There might be an **`undefined symbol: SSLeay`** message in the Apache error log

  Upgrading Swarm:

  - **Swarm package or OVA:** the latest P4PHP version is installed automatically.
  - **Swarm tarball installation:** you must configure Swarm to use the version of P4PHP in the new Swarm tarball. For swarm tarball upgrade instructions, see "Upgrade a Swarm tarball installation" on page 208.

- **`php-xml`** (DOM API for XML manipulation, the Swarm RSS feed will not work if it is not installed)

  Included with PHP on many operating systems. Must be manually installed on CentOS/RHEL unless you are installing Swarm from a package.

- **`php-mbstring`** (multi-byte character strings, the Swarm RSS feed will not work if it is not installed)

  Included with PHP on many operating systems. Must be manually installed on CentOS/RHEL unless you are installing Swarm from a package.

- **`php-redis`** (PHP extension for Redis, the Swarm cache will not work if it is not installed)

  Included with PHP on many operating systems. Must be manually installed on CentOS/RHEL unless you are installing Swarm from a package.

- **CentOS:** in addition to **`php-redis`** the following extensions must be manually installed unless you are installing Swarm from a package:

  - **`sclo-php74-php-pecl-redis`** (PHP extension for Redis, the Swarm cache will not work if it is not installed)

    - **`sclo-php74-php-pecl-igbinary`** (replacement PHP serializer for Redis, the Swarm cache will not work if it is not installed)
- **RHEL:** in addition to **`php-redis`** the following extensions must be manually installed unless you are installing Swarm from a package:
    - **`php74-php-pecl-redis`** (PHP extension for Redis, the Swarm cache will not work if it is not installed)
    - **`php74-php-pecl-igbinary`** (replacement PHP serializer for Redis, the Swarm cache will not work if it is not installed)

Recommended PHP extension

Swarm greatly benefits from the following PHP extension:

- **`Imagick`** (integrates ImageMagick into PHP) https://secure.php.net/imagick
  Installation instructions for Imagick.

Building PHP from source (not recommended)

If you build PHP from source, the following dependencies are required:

- **`openssl`**
- **`mcrypt`**
- **`zlib`**
- **`gettext`**
- **`curl`**
- **`apxs`** (Apache extension tool)

See the source PHP documentation for details on how to include these modules in your PHP build. There is normally a **`--with-xxxx`** option that defines where the dependency is loaded from.

For example for **`apxs`**, **`zlib`**, and **`openssl`**:

```
./configure --with-apxs2=<path_to_apxs> --with-zlib=<path_to_zlib_library> --with-openssl
```

Helix Core server requirements

Swarm works with any supported version of the Helix server (Standard Maintenance). The versions supported in this release of Swarm are:

- 2019.1, see note below.

  > **Note**
  > Helix server 2019.1 introduced the Private editing of streams feature that enables you to edit a stream spec in your workspace. The level of Swarm support depends on the version of Helix server 2019.1 installed:
  >
  > - **Helix server 2019.1 at 1832527 or earlier:** if you request a review from a changelist that contains a stream spec that was edited using this feature, it will not be displayed in the review. Other files in the changelist are displayed correctly in the review.
  > - **Helix server 2019.1 later than 1832527:** stream specs can be edited in your workspace using this feature and they are displayed in reviews.

- 2019.2
- 2020.1

  > **Note**
  > Helix server 2020.1 and later, permissions have changed for viewing and editing stream spec files in Swarm. To view and edit stream spec files in Swarm, the Swarm user must have *admin* permissions for the entire depot `//...`

- 2020.2

  > **Note**
  > **Helix server 2020.2 and later:** any new file being shelved that has the same content as an existing shelved file refers to the existing archive file instead of creating a duplicate archive file. No Helix server or Swarm configuration is required for this feature.
  >
  > This Helix server feature automatically reduces the space required for the Swarm-managed shelved review changelists. Swarm creates these changelists for its own internal use. Helix server only updates new shelves, it does not retrospectively update your existing shelves.
  >
  > For more information on the Swarm-managed changelists, see "Internal representation" on page 339.

Download Helix server from here: https://www.perforce.com/downloads/helix-core-p4d

> **Important**
> Swarm does not support Helix servers configured to use `P4AUTH`, see Centralized authentication server (P4AUTH) in the *Helix Core Server Administrator Guide*.

Helix server automated user requirements for Swarm

Swarm requires an automated user with at least admin privileges in the Helix Core server to enable Swarm to run against the Helix server. This can be an existing user, or a new user created specifically to support Swarm.

> **Important**
> If Helix Authentication Service is configured for your Helix server, the user account running Swarm must not use the Helix Authentication Service.

For more information about setting up Helix Core server, see Installing and upgrading the server in *Helix Core Server Administrator Guide*.

Swarm and Helix server installation considerations

Before installing Swarm and Helix server you should consider the following:

- For a small system, you can run Swarm and Helix server on the same machine.
- For larger systems, we recommend that Swarm and Helix server are run on separate machines. The machines should be close to each other to maximize network performance.

  Triggers are installed on the Helix server machine and need to talk to Swarm. Trigger performance can be negatively affected if network lag between Swarm and the Helix server is high.
- The Swarm and Helix server machines do not need to have the same operating system. For example, Helix server could be on a Windows server and Swarm could be on a CentOS server.

  > **Note**
  > You cannot install Swarm on a Windows machine.

Trigger dependencies

Swarm triggers must be installed on the Helix server to complete the Swarm installation.

The Swarm triggers require `perl 5.08+`:
https://www.perl.org/get.html

On the Windows platform, we have tested Swarm against Strawberry Perl. There are two Perl modules that are also required which may not be part of a minimal Perl installation.

- `HTTP::Tiny` is required to make calls to the Swarm server. If this is not present, then the trigger will attempt to use the command line curl program. This module is standard on Strawberry Perl on Windows, and available as a package with the version of Perl provided on CentOS 7, CentOS 8, Ubuntu 16.04, Ubuntu 18.04, and Ubuntu 20.04.

`IO::Socket::SSL` is required if the Swarm server is configured to use SSL and HTTP::Tiny is present. This is provided as standard by Strawberry Perl, and available on Linux.

> **Warning**
> If the `HTTP::Tiny` module is not available the triggers require the use of `curl`. This must be installed for the triggers to function. On CentOS, for example, this can be done using the `yum` package installer using `yum install curl`.

Swarm triggers also require the following perl modules to be installed:

- **Windows:**
  - `JSON` is required to exchange data between the browser and the server and is included by default with Strawberry Perl.
- **Ubuntu:**
  - `JSON` is required to exchange data between the browser and the server and must be installed.
  - `libjson-perl` is required to manipulate JSON formatted data and must be installed.
- **CentOS/RHEL:**
  - `JSON` is required to exchange data between the browser and the server and must be installed.
  - `perl-JSON` is required to manipulate JSON formatted data and must be installed.

Swarm package installation on the same machine as P4D

If Swarm is installed from a package on the same machine as P4D, the triggers require the following perl modules to be installed:

- **Ubuntu:**
  - `Perl 5.08+`
  - `libio-socket-ssl-perl` is required if the Swarm server is configured to use SSL.
  - `libjson-perl` is required to manipulate JSON formatted data and must be installed.
- **CentOS/RHEL:**
  - `Perl 5.08+`
  - `perl-IO-Socket-SSL` is required if the Swarm server is configured to use SSL.
  - `perl-JSON` is required to manipulate JSON formatted data and must be installed.

Worker dependencies

Swarm uses short-lived workers to process the Swarm queue, new workers are regularly spawned by a recurring task.

One of the following must be installed to ensure new workers are regularly spawned:

- **curl**, download from: https://curl.haxx.se/download.html
- **wget**, download from: https://ftp.gnu.org/gnu/wget/

For more information about Swarm workers, see "Set up a recurring task to spawn workers" on page 174.

Redis server

Swarm requires Redis to manage its caches. Swarm caches data from the Helix server to improve the performance of common searches in Swarm and to reduce the load on the Helix server.

Redis is included with the Swarm package, OVA, and Tarball installations:

- **Swarm package or OVA:** Redis is automatically installed on the Swarm machine and Swarm is automatically configured to use Redis.
- **Swarm Tarball installation:** see the Redis installation and configuration section for instructions about installing Redis.

> **Tip**
> When Swarm starts it verifies the Redis cache, during this time you cannot log in to Swarm. The time taken to verify the Redis cache depends on the number of users, groups, and projects Swarm has. Start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves in the `redis-server.conf` file, see "Redis server configuration file" on page 542.

Optional:

If you prefer to use your own Redis server, you must edit the Redis server connection configurable in Swarm. For information on setting your Redis server connection, see "Use your own Redis server" on page 189.

Supported web browsers

The following browsers are supported for use with Swarm:

- Apple Safari, latest stable version
- Google Chrome, latest stable version
- Mozilla Firefox, latest stable version
- Microsoft Edge, latest stable version
- Microsoft Internet Explorer, latest stable version

Other web browsers might also work, including prior, development or beta builds of the above web browsers, but are not officially supported.

Note

- JavaScript and cookies must be enabled in the web browser for Swarm to operate.
- Mobile web browsers are not supported by Swarm.

Optional dependencies

LibreOffice

Swarm can display previews of office-type documents when LibreOffice is installed on the Swarm server. Installation is not required, but when LibreOffice is installed Swarm automatically detects its presence.

For more information about LibreOffice, see "LibreOffice" on page 451.

Zip

You can download a ZIP archive of files/folders when the zip command-line tool is installed on the Swarm server.

For more information about installing and configuring Zip, see "Zip archive" on page 452.

Helix Authentication Service SSO

Swarm support for SSO using Helix Authentication Service with Helix server requires Helix server 2019.1 or later

For more information about configuring Swarm to authenticate with Helix Authentication Service, see "Single Sign-On PHP configuration" on page 574.

Swarm email notifications

Sendmail or an equivalent is required to use Swarm email notifications. By default, the configuration in `php.ini` relies on SendMail being installed. For more information about configuring email for Swarm, see "Email configuration" on page 489.

Security-enhanced Linux (SELinux)

Swarm supports SELinux on CentOS/RHEL 7 and CentOS/RHEL 8. SELinux is an advanced access control mechanism that improves security for Linux distributions.

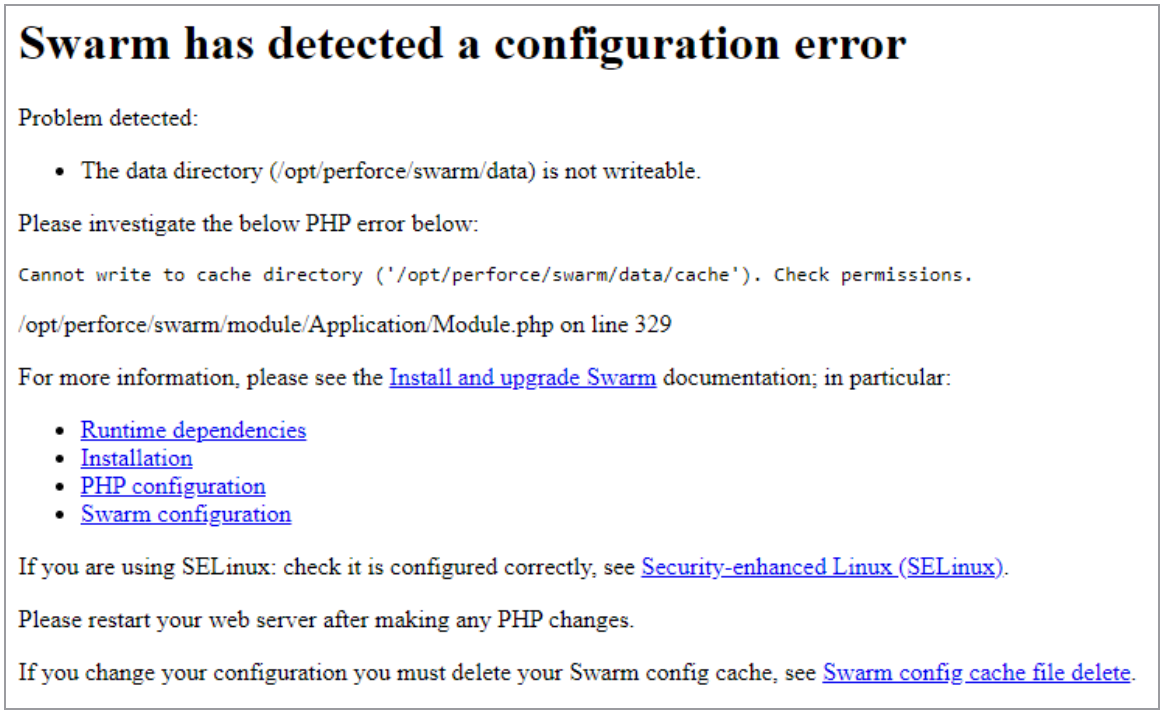
SELinux operates in one of three modes:

- `enforcing`: this mode blocks and logs any actions that do not match the defined security policy. This is the default mode for SELinux.

- **`permissive`**: this mode logs actions that do not match the defined security policy but these actions are not blocked.
- **`disabled`**: in this mode SELinux is off, actions are not blocked and are not logged.

See "SELinux on CentOS/RHEL configuration" on page 134 for details.

If you see a Swarm configuration error similar to the error shown below, SELinux has not been correctly configured for Swarm. Check you have configured SELinux on CentOS/REHL correctly.



Swarm has detected a configuration error

Problem detected:

- The data directory (/opt/perforce/swarm/data) is not writeable.

Please investigate the below PHP error below:

```
Cannot write to cache directory ('/opt/perforce/swarm/data/cache'). Check permissions.
```

/opt/perforce/swarm/module/Application/Module.php on line 329

For more information, please see the Install and upgrade Swarm documentation; in particular:

- Runtime dependencies
- Installation
- PHP configuration
- Swarm configuration

If you are using SELinux: check it is configured correctly, see Security-enhanced Linux (SELinux).

Please restart your web server after making any PHP changes.

If you change your configuration you must delete your Swarm config cache, see Swarm config cache file delete.

Choose the installation process

Once you have reviewed the "Runtime dependencies" on page 101 and know that you can satisfy them, there are a number of ways to install Swarm.

> **Note**
> We recommend the package installation method to install Swarm whenever possible, see "Install and configure Swarm from a package" on page 115. Package installs ensure that all of the Swarm dependencies are installed and this is the easiest way to install Swarm. For a list of recommended operating systems for Swarm, see "Recommended operating systems" on page 102.

Choose one of the following installation methods (we recommend the package installation method whenever possible):

- **Swarm RPM or Debian packages (recommended):**
    1. Follow the steps provided in "Install and configure Swarm from a package" on the facing page.
    2. Establish a trigger token, see "Establish trigger token" on page 158.
    3. Configure Helix Core server for Swarm, see "Helix Core server configuration for Swarm" on page 158.
    4. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
    5. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.
    6. You are now all set to start using Swarm. Enjoy!

    > **Tip**
    > To get started with Swarm, see the "Quickstart" on page 29 chapter.

- **Swarm.ova (Open Virtualization Appliance):**
    1. Follow the steps provided in "Deploy and configure a Swarm VM from an OVA" on page 137.
    2. Establish a trigger token, see "Establish trigger token" on page 158.
    3. Configure Helix Core server for Swarm, see "Helix Core server configuration for Swarm" on page 158.
    4. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
    5. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.
    6. You are now all set to start using Swarm. Enjoy!

    > **Tip**
    > To get started with Swarm, see the "Quickstart" on page 29 chapter.

- **Swarm.tgz (Tarball):**
    1. Follow steps provided in the "Install and configure Swarm manually from a Tarball" on page 142.
    2. Configure Redis, see "Redis configuration" on page 144.
    3. Configure Apache, see "Apache configuration" on page 148.
    4. Configure PHP, see "PHP configuration" on page 151.
    5. Configure Swarm, see "Swarm configuration" on page 154.

6. Establish a trigger token, see "Establish trigger token" on page 158.
7. Configure Helix Core server for Swarm, see "Helix Core server configuration for Swarm" on page 158.
8. Set up a recurring task to spawn workers, see "Set up a recurring task to spawn workers" on page 174.
9. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
10. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.
11. You are now all set to start using Swarm. Enjoy!

> **Tip**
> To get started with Swarm, see the "Quickstart" on page 29 chapter.

Install and configure Swarm from a package

Helix Swarm is available in two distribution package formats: Debian (`.deb`) for Ubuntu systems and RPM (`.rpm`) for CentOS and RedHat Enterprise Linux (RHEL).

Using distribution packages greatly simplifies the installation, updating, and removal of software, as the tools that manage these packages are aware of the dependencies for each package.

> **Important**
> Swarm does not support Helix servers configured to use `P4AUTH`, see Centralized authentication server (P4AUTH) in the *Helix Core Server Administrator Guide*.

> **Note**
> Swarm packages are available for Ubuntu 16.04 LTS, Ubuntu 18.04 LTS, Ubuntu 20.04 LTS, CentOS/RHEL 7, and CentOS/RHEL 8. For Red Hat based distributions, we recommend that the latest supported versions of those platforms are used. At the time of the Swarm 2020.2 release, the latest point releases are 7.8 and 8.2.
>
> While the packages should work on other compatible distributions, these have not been tested.

> **Note**
> - *Helix Core server* can refer to a Helix server machine (P4D), proxy, broker, replica, edge server, or commit server. For simplicity, the term *Helix server* is used to refer to any configuration of a Helix Core server machine.
> - Swarm can be connected to Helix servers (P4D) and commit servers.

To configure Swarm to connect to more than one Helix server (P4D), see "Multiple-Helix server instances" on page 519.

To configure Swarm to connect to a Helix server configured to use *commit-edge architecture*, see "Commit-edge deployment" on page 476.

- Swarm must not be connected to Helix Broker, Helix Proxy, Helix Edge, forwarding replica, or read-only replica servers.

Installation

Important

- Review the runtime dependencies before you install Swarm, see "Runtime dependencies" on page 101.
- Review the PHP requirements before you upgrade Swarm, see "PHP" on page 105.
- Review the Helix server requirements before you install Swarm, see "Helix Core server requirements" on page 107.

1. Configure the Perforce package repository, on the server to host Swarm and on the server hosting your Helix Core server.

   Important
   If the server hosting your Helix Core server cannot use packages, for example when it is running Windows, skip this step on that server.

   As root, follow the instructions for your OS distribution:

   Ubuntu

   Ubuntu 16.04:

   Create the file `/etc/apt/sources.list.d/perforce.list` with the following content:

   ```
   deb http://package.perforce.com/apt/ubuntu/ xenial release
   ```

   Ubuntu 18.04:

   Create the file `/etc/apt/sources.list.d/perforce.list` with the following content:

   ```
   deb http://package.perforce.com/apt/ubuntu/ bionic release
   ```

Ubuntu 20.04:

Create the file `/etc/apt/sources.list.d/perforce.list` with the following content:

```
deb http://package.perforce.com/apt/ubuntu/ focal release
```

CentOS/RHEL

CentOS/RHEL 7:

Create the file `/etc/yum.repos.d/perforce.repo` with the following content:

```
[Perforce]
name=Perforce
baseurl=http://package.perforce.com/yum/rhel/7/x86_64/
enabled=1
gpgcheck=1
```

CentOS/RHEL 8:

Create the file `/etc/yum.repos.d/perforce.repo` with the following content:

```
[Perforce]
name=Perforce
baseurl=http://package.perforce.com/yum/rhel/8/x86_64/
enabled=1
gpgcheck=1
```

2. Import the Perforce package signing key, on the server to host Swarm and the server hosting your Helix Core server.

> **Important**
> If the server hosting your Helix server cannot use packages, for example when it is running Windows, skip this step on that server.

Follow the instructions for your OS distribution:

Ubuntu:

```
wget -qO - https://package.perforce.com/perforce.pubkey | sudo apt-
key add -
sudo apt-get update
```

CentOS/RHEL (run this command as root):

```
rpm --import https://package.perforce.com/perforce.pubkey
```

For information about how to verify the authenticity of the signing key, see: https://www.perforce.com/perforce-packages

3. Install the main Swarm package on the server to host Swarm.

   Follow the instructions for your OS distribution:

   Ubuntu

   ```
   sudo apt-get install helix-swarm
   ```

CentOS

Follow the instructions for your CentOS distribution version:

CentOS 7.8 (run these commands as root):

a. Deploy the `epel-release-latest-7.noarch.rpm` repository configuration package to give Swarm access to EPEL packages:

```
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

```
yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

> **Tip**
> If you don't deploy the Remi repository you will see an error similar to the one shown below when you do the next steps:

```
--> Finished Dependency Resolution
Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
           Requires: httpd24
Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
           Requires: rh-php70-php-xml
Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
           Requires: rh-php70-php-mbstring
Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
           Requires: rh-php70-php-opcache
Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
           Requires: rh-php70-php
Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
           Requires: rh-php70
 You could try using --skip-broken to work around the problem
 You could try running: rpm -Va --nofiles --nodigest
```

c. Install the `yum-utils` package to give access to the `yum-config-manager` command:

```
yum install yum-utils
```

d. Install the **Default/Single version** of PHP:

   i. Disable `remi-php*`:

   ```
   yum-config-manager --disable 'remi-php*'
   ```

   ii. Enable PHP 7.4:

   ```
   yum-config-manager --enable remi-php74
   ```

iii. Run an upgrade for PHP:

```
yum update
```

e. Install Swarm and accept the prompts to import the GPG keys for Remi and EPEL when requested:

```
yum install helix-swarm
```

> **Note**
> The firewall configuration may need to be adjusted to allow access to the web server.
>
> ```
> sudo firewall-cmd --permanent --add-service=http
> sudo systemctl reload firewalld
> ```
>
> If you subsequently wish to enable HTTPS, run (as root):
>
> ```
> sudo firewall-cmd --permanent --add-service=https
> sudo systemctl reload firewalld
> ```

CentOS 8.2 (run these commands as root):

a. Deploy the `epel-release-latest-8.noarch.rpm` repository configuration package to give Swarm access to EPEL packages:

```
dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
```

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

```
dnf install https://rpms.remirepo.net/enterprise/remi-release-8.rpm
```

> **Tip**
> If you don't deploy the Remi repository you will see an error similar to the one shown below when you do the next steps:
>
> ```
> --> Finished Dependency Resolution
> Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
>           Requires: httpd24
> Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
>           Requires: rh-php70-php-xml
> Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
>           Requires: rh-php70-php-mbstring
> Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
>           Requires: rh-php70-php-opcache
> Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
>           Requires: rh-php70-php
> Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
>           Requires: rh-php70
>  You could try using --skip-broken to work around the problem
>  You could try running: rpm -Va --nofiles --nodigest
> ```

c. Install the `yum-utils` package to give access to the `yum-config-manager` command:

   ```
   dnf install yum-utils
   ```

d. Install the **Default/Single version** of PHP:

   i. Enable the module stream for PHP 7.4:

      ```
      dnf module reset php
      ```

   ii. Install PHP 7.4:

      ```
      dnf module install php:remi-7.4
      ```

   iii. Run an upgrade for PHP:

      ```
      dnf update
      ```

e. Install Swarm and accept the prompts to import the GPG keys for Remi and EPEL when requested:

   ```
   yum install helix-swarm
   ```

Note
The firewall configuration may need to be adjusted to allow access to the web server.

```
sudo firewall-cmd --permanent --add-service=http
sudo systemctl reload firewalld
```

If you subsequently wish to enable HTTPS, run (as root):

```
sudo firewall-cmd --permanent --add-service=https
sudo systemctl reload firewalld
```

RHEL

Follow the instructions for your RHEL distribution version:

RHEL 7.8 (run these commands as root):

a. Deploy the `epel-release-latest-7.noarch.rpm` repository configuration package to give Swarm access to EPEL packages:

   ```
   yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
   ```

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

   ```
   yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm
   ```

   > **Tip**
   > If you don't enable the repository you will see an error similar to the one shown below when you do the next step:
   >
   > ```
   > Error: Package: helix-swarm-2020.2-1845646.el6.x86_64
   > (perforce)
   >         Requires: rh-php74
   > ```

c. Install the `yum-utils` package to give access to the `yum-config-manager` command:

   ```
   yum install yum-utils
   ```

d. Enable the optional channel for some dependencies:

   ```
   subscription-manager repos --enable=rhel-7-server-optional-rpms
   ```

e. Install the **Default/Single version** of PHP:

   i. Disable `remi-php*`:

      ```
      yum-config-manager --disable 'remi-php*'
      ```

   ii. Enable PHP 7.4:

      ```
      yum-config-manager --enable remi-php74
      ```

   iii. Run an upgrade for PHP:

      ```
      yum update
      ```

f. Install Swarm and accept the prompts to import the GPG keys for Remi and EPEL when requested:

   ```
   yum install helix-swarm
   ```

> **Note**
> The firewall configuration may need to be adjusted to allow access to the web server.
>
> ```
> sudo firewall-cmd --permanent --add-service=http
> sudo systemctl reload firewalld
> ```
>
> If you subsequently wish to enable HTTPS, run (as root):
>
> ```
> sudo firewall-cmd --permanent --add-service=https
> sudo systemctl reload firewalld
> ```

RHEL 8.2 (run these commands as root):

a. Deploy the `epel-release-latest-8.noarch.rpm` repository configuration package to give Swarm access to EPEL packages:

   ```
   dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
   ```

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

   ```
   dnf install https://rpms.remirepo.net/enterprise/remi-release-8.rpm
   ```

   > **Tip**
   > If you don't enable the repository you will see an error similar to the one shown below when you do the next step:
   >
   > ```
   > Error: Package: helix-swarm-2020.2-1845646.el6.x86_64
   > (perforce)
   >         Requires: rh-php74
   > ```

c. Install the `yum-utils` package to give access to the `yum-config-manager` command:

   ```
   dnf install yum-utils
   ```

d. Install the **Default/Single version** of PHP:

   i. Enable the module stream for PHP 7.4:

      ```
      dnf module reset php
      ```

   ii. Install PHP 7.4:

      ```
      dnf module install php:remi-7.4
      ```

iii. Run an upgrade for PHP:

```
dnf update
```

e. Install Swarm and accept the prompts to import the GPG keys for Remi and EPEL when requested:

```
yum install helix-swarm
```

> **Note**
> The firewall configuration may need to be adjusted to allow access to the web server.
>
> ```
> sudo firewall-cmd --permanent --add-service=http
> sudo systemctl reload firewalld
> ```
>
> If you subsequently wish to enable HTTPS, run (as root):
>
> ```
> sudo firewall-cmd --permanent --add-service=https
> sudo systemctl reload firewalld
> ```

4. Install the Swarm triggers package on the server hosting your Helix Core server.

   Install this package on the server hosting your Helix Core server, which may be the same server that is hosting Swarm, or elsewhere on your network.

   > **Important**
   > If the server hosting your Helix server cannot use packages, for example when it is running Windows, you need to copy the appropriate Swarm trigger script from **`/opt/perforce/swarm/p4-bin/scripts`** to the server hosting your Helix server. The **`swarm-trigger.pl`** is for both Linux and Windows systems. Once copied, the trigger script needs to be configured. See "Helix Core server configuration for Swarm" on page 158 for details.

   Follow the instructions for your OS distribution:

   Ubuntu:

   ```
   sudo apt-get install helix-swarm-triggers
   ```

   CentOS/RHEL (run this command as root):

   ```
   yum install helix-swarm-triggers
   ```

   > **Important**
   > The package installs a config file at **`/opt/perforce/etc/swarm-trigger.conf`** that you will need to modify. See "Helix Core server configuration for Swarm" on page 158 for more details on configuring that file.

5. **Optional:** Install the Swarm optional package, on the server hosting Swarm.

   While not required, installing this package installs the dependencies required to use the ImageMagick and LibreOffice Swarm modules. These modules provide previews of a variety of image and office documents.

   Follow the instructions for your OS distribution:

   Ubuntu:

   ```
   sudo apt-get install helix-swarm-optional
   ```

   CentOS/RHEL (run this command as root):

   ```
   yum install helix-swarm-optional
   ```

6. Complete the "Post-installation configuration" below steps.

Post-installation configuration

Important
CentOS/RHEL only:

- **Swarm 2019.1 to 2020.1:** these versions of Swarm used PHP packages from the SCL repositories for CentOS/RHEL 7. This was to provide access to more recent versions of PHP than are available as standard on CentOS/RHEL. This required the use of the **`httpd24-httpd`** package for Apache. These were all installed into **`/opt/rh`**

  Swarm 2020.2: this version of Swarm uses the Remi repository for CentOS/RHEL 7 and 8. This provides PHP 7.4 installed in the standard file system structure. This means that the old **`httpd24-httpd`** version of Apache is no longer needed, and the standard system version of Apache is being used again.

  The SCL Apache site configuration file was installed at this location for Swarm 2019.1 to 2020.1:

  `/opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf`

  If this exists when Swarm is upgraded to 2020.2, this file is copied to **`/etc/httpd/conf.d/perforce-swarm-site.conf`** if there is no file at the destination. It is also re-written to change references from **`/var/log/httpd24`** to **`/var/log/httpd`**

  If a site configuration file for Swarm already exists in **`/etc/httpd`**, the copy and re-write is not performed.

  After upgrade, **`httpd24-httpd`** is disabled.

- To avoid seeing the Apache HTTP server Linux test page when you start the Apache server, comment out the content of the **`welcome.conf`** file located in the **`/etc/httpd/conf.d/`** directory.

Once the helix-swarm package has been installed, additional configuration is required. Perform the following steps:

1. Use the Swarm configuration script to setup Swarm, on the server hosting Swarm.

   Important
   If your Helix server is configured for Helix Authentication Service, the Helix server must be temporarily configured to allow fall-back to passwords while you establish a connection to the Helix server. Run the following command on the Helix server to enable fall-back to passwords:

   ```
   p4 configure set auth.sso.allow.passwd=1
   ```

   Note
   The Swarm configuration script can be used in a few different ways. The steps below outline the most straightforward configuration using an interactive install, but you can review the options by running:

   ```
   sudo /opt/perforce/swarm/sbin/configure-swarm.sh -h
   ```

   Run an interactive install:

   ```
   sudo /opt/perforce/swarm/sbin/configure-swarm.sh
   ```

   The configuration script displays the following summary:

   ```
   ------------------------------------------------------------
   configure-swarm.sh: Thu Aug 23 11:29:49 PDT 2018: commencing
   configuration of Swarm
   Summary of arguments passed:
   Interactive?       [yes]
   Force?             [no]
   P4PORT             [(not specified)]
   Swarm user         [(not specified, will suggest swarm)]
   Swarm password     [(not specified)]
   Email host         [(not specified)]
   Swarm host         [(not specified, will suggest myhost)]
   Swarm port         [80]
   Swarm base URL     [(default (empty))]
   Create Swarm user? [no]
   Super user         [(not specified)] * not needed
   Super password     [(not specified)] * not needed
   ```

2. Provide information to the configuration script.

After the summary, the configuration script prompts for the following information:

a. Specify a value for **`P4PORT`** in the form: *my-helix-core-server:1666*

```
No P4PORT specified

Swarm requires a connection to a Helix Core Server.
Please supply the P4PORT to connect to.

Helix Core Server address (P4PORT):
```

Specify the hostname and port for your Helix server. If defined, the value for P4PORT is used as the default. The configuration script verifies that it can connect:

```
-response: [myp4host:1666]

Checking P4PORT [myp4host:1666]...
-P4 command line to use: [/opt/perforce/bin/p4 -p myp4host:1666]
Attempting connection to [myp4host:1666]...
-connection successful:
  Server address: myp4host:1666
  Server version: P4D/LINUX26X86_64/2017.2/1622831 (2017/10/24)
  Server license: 10000 users (support ends 2019/05/16)
  Server license-ip: 192.168.0.1
```

Important
If your Helix Core server is deployed using the commit-edge architecture, ensure that the Swarm port value points to the commit server.

For more information, see the Commit-edge chapter in the *Helix Core Server Administrator Guide*.

b. Specify the userid and password of a normal user with admin-level privileges in the Helix Core server.

```
Checking Swarm user credentials...
No Swarm user specified

Swarm requires a Helix user account with 'admin' rights.
Please provide a username and password for this account.
If this account does not have 'admin' rights, it will
```

```
be set for this user.

Helix username for the Swarm user [swarm]:
```

Enter the userid. The default is *swarm*.

> **Note**
> If the Helix server user account is given 'super' rights, then this allows a user to clean up a review created by another user when the review is committed. See "Review cleanup" on page 544.

```
-response: [swarm]

Helix password or login ticket for the Swarm user (typing
hidden):
```

Enter the login ticket, or password, for the userid.

> **Important**
> If your Helix server is configured for Helix Authentication Service, you must use a long-lived login ticket for the Swarm user.

> **Note**
> You can obtain a login ticket by running (in another shell):
>
> `$ p4 -p myp4host:1666 -u userid login -p`
>
> If the login ticket you provide would expire in less than a year, you will receive a warning.

```
Checking Swarm user credentials...
-checking if user [swarm] exists in [myp4host:1666]...
-user exists
Obtaining Helix login ticket for [swarm] in [myp4host:1666]...
-login ticket obtained
Checking user [swarm]'s ticket against [myp4host:1666]...
-login ticket is good
Checking user [swarm] has at least access level [admin]...
-user has maximum access level [admin]
-user meets minimum access level [admin]
```

c. Specify the hostname for the Swarm UI.

```
Swarm needs a distinct hostname that users can enter into their
browsers to
access Swarm. Ideally, this is a fully-qualified domain name,
e.g.
'swarm.company.com', but it can be just a hostname, e.g. 'swarm'.

Whatever hostname you provide should be Swarm-specific and not
shared with
any other web service on this host.

Note that the hostname you specify typically requires
configuration in your
network's DNS service. If you are merely testing Swarm, you can
add a
hostname->IP mapping entry to your computer's hosts
configuration.

Hostname for this Swarm server [myhost]:
```

> **Note**
> The default is the current hostname. The configuration script does not verify that the hostname actually works (DNS configuration may not exist yet).

d. Specify a mail relay host.

```
Swarm requires an mail relay host to send email notifications.

Mail relay host (e.g.: mx.yourdomain.com):
```

> **Note**
> The configuration script does not verify that the mail relay host you provide actually accepts SMTP connections.

Once this information has been provided, the configuration script performs the following steps (some of the detail depends on the version of PHP and Apache that is installed):

```
Configuring Cron...
`/opt/perforce/etc/swarm-cron-hosts.conf.new' -&gt;
`/opt/perforce/etc/swarm-cron-hosts.conf'
-updated cron configuration file with supplied Swarm host
Configuring Swarm installation...
-composed new Swarm config file contents
`/opt/perforce/swarm/data/config.php.new' -&gt;
`/opt/perforce/swarm/data/config.php'
-wrote new Swarm config file to reflect new configuration
-identified Apache user:group: [www-data:www-data]
-setting permissions on the Swarm data directory...
-ensured file permissions are set properly
Configuring Apache...
-identified Swarm virtual host config file: [/etc/apache2/sites-
available/perforce-swarm-site.conf]
-identified Apache log directory: [/var/log/apache2]
-updated the vhost file to set Apache log directory
-updated the vhost file to reflect Swarm host
-checking Apache modules...
Enabling module rewrite.
Module php7 already enabled
To activate the new configuration, you need to run:
  service apache2 restart
-proper Apache modules are enabled
-enabling Swarm Apache site...
Enabling site perforce-swarm-site.conf.
To activate the new configuration, you need to run:
  service apache2 reload
-Swarm Apache site enabled
-restarting Apache...
-Apache restarted
configure-swarm.sh: Thu Aug 23 11:31:36 PDT 2018: completed
configuration of Helix Swarm
```

```
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::
::
::  Swarm is now configured and available at:
::
::      http://myhost/
::
::  You may login as the Swarm user [swarm] using the password
::  you specified.
::
::  Please ensure you install the following package on the server
::  hosting your Helix Core Server.
::
::      helix-swarm-triggers
::
::  (If your Helix Core Server is hosted on an OS and
::  platform that is not compatible with the above package, you
can
::  also install the trigger script manually.)
::
::  You will need to configure the triggers, as covered in the
Swarm
::  documentation:
::
::
http://www.perforce.com/perforce/doc.current/manuals/swarm/setup.
perforce.html
::
::  Documentation for optional post-install configuration, such
as
::  configuring Swarm to use HTTPS, operate in a sub-folder, or
```

```
on a
::  custom port, is available:
::
::
https://www.perforce.com/perforce/doc.current/manuals/swarm/setup
.post.html
::

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::
```

> **Note**
> If you have installed Swarm on a host that does not provide other web services, you may wish to disable Apache's default site configuration. Doing so means that regardless of the hostname a user might use to reach the web server hosting Swarm, Swarm would be presented.
>
> **Be aware that disabling Apache's default site configuration could disable existing web services or content.**
>
> Disabling Apache's default site configuration on Ubuntu hosts is easy. Run:
>
> ```
> $ sudo a2dissite 000-default
> ```
>
> For CentOS hosts, or for non-standard Apache installations, you would need to manually adjust the Apache configuration. Such changes require familiarity with Apache configuration; for more details, see: https://httpd.apache.org/docs/2.4/configuring.html

3. **CentOS/RHEL only:** Configure SELinux on CentOS/RHEL for Swarm, see "SELinux on CentOS/RHEL configuration" on the next page.
4. The basic Swarm configuration is now complete.

> **Important**
> If your Helix server is configured for Helix Authentication Service, you can force all of your users to authenticate via your Identity Provider (IdP) by disabling fall-back to passwords. To disable fall-back to passwords on the Helix server, run the following command:
>
> ```
> p4 configure set auth.sso.allow.passwd=0
> ```

5. Establish a trigger token, see "Establish trigger token" on page 158.

SELinux on CentOS/RHEL configuration

Swarm supports SELinux on CentOS/RHEL 7 and CentOS/RHEL 8. SELinux is an advanced access control mechanism that improves security for Linux distributions.

SELinux operates in one of three modes:

- `enforcing`: this mode blocks and logs any actions that do not match the defined security policy. This is the default mode for SELinux.
- `permissive`: this mode logs actions that do not match the defined security policy but these actions are not blocked.
- `disabled`: in this mode SELinux is off, actions are not blocked and are not logged.

SELinux must be configured to enable it to work correctly with Swarm, these configuration steps are shown below.

> **Note**
> You must complete the Helix Swarm package "Installation" on page 116 steps, and the "Post-installation configuration" on page 126 steps before configuring SELinux.

Configure SELinux to enforcing mode

Run the following commands as root:

1. Install the package that contains the **semanage** configuration tool, this is used to configure SELinux:

   Follow the instructions for your OS distribution:

   - **CentOS/RHEL 7:** Install the `policycoreutils-python` package:

     ```
     root $ yum install policycoreutils-python
     ```

   - **CentOS/RHEL 8:** Install the `policycoreutils-python-utils` package:

     ```
     root $ yum install policycoreutils-python-utils
     ```

2. Check the current SELinux mode:

   ```
   root $ getenforce
   ```

3. SELinux will report its mode as; `enforcing`, `permissive`, or `disabled`.

   a. If the mode is not set correctly edit the `/etc/selinux/config` file with vi or a similar editor.

      ```
      root $ vi /etc/selinux/config
      ```

   b. Edit the config file so that `SELinux=` is set to `enforcing` .

   c. Save the config file.

   d. Reboot the server to complete the SELinux mode change.

4. Define the context of the **`/opt/perforce/swarm`** directory and the files in it to **`httpd_sys_rw_content_t`**:

```
root $ semanage fcontext -a -t httpd_sys_rw_content_t
"/opt/perforce/swarm(/.*)?"
root $ restorecon -R /opt/perforce/swarm
```

5. Set the SELinux Boolean value to **`httpd_can_network_connect 1`** to allow Swarm to connect to p4d and other services:

```
root $ setsebool -P httpd_can_network_connect 1
```

6. Define the context of the **`/opt/perforce/swarm/p4-bin`** directory and the files in it to **`httpd_sys_script_exec_t`**

```
root $ semanage fcontext -a -t httpd_sys_script_exec_t
'/opt/perforce/swarm/p4-bin(/.*)?'
root $ restorecon -R -v /opt/perforce/swarm/p4-bin
```

7. Restart the system:

```
root $ systemctl restart httpd
```

8. Check that you can log in to Swarm.
9. **Only if required:** Relabel your filesystem, see note before relabeling:

> **Important**
> Relabeling your file system can be a time consuming process, it is recommended that you only do this if you need to. This depends entirely on your SELinux setup, Perforce cannot give you advice on this.

```
root $ touch /.autorelabel
```

10. Reboot the server.
11. Check that you can log in to Swarm.
12. SELinux is now configured for Swarm.

> **Note**
> If you can not log in to Swarm it is possible that SELinux is blocking Swarm because its configuration is incorrect. You will need to troubleshoot the SELinux configuration to find any issues.
>
> Install the **`setroubleshoot`** package, this contains **`sealert`** which is used when troubleshooting SELinux:
>
> **`root $ yum install setroubleshoot`**
>
> **`sealert`** helps you to interpret the contents of the **`audit.log`**. Run the following command:
>
> **`root $ sealert -a /var/log/audit/audit.log`**

Error message: If you see an error message with a title similar to the message below, it may be because you are running CentOS/RHEL on a Virtual Machine (VM).

```
root $ SELinux is preventing /usr/sbin/ldconfig from write access on the directory etc.
```

Install `open-vm-tools` on the VM and reboot the VM.

```
root $ yum install open-vm-tools
```

Configure SELinux permissive or disabled mode

Run the following as root:

1. Check the current SELinux mode:

   ```
   root $ getenforce
   ```

2. SELinux will report its mode as; `enforcing`, `permissive`, or `disabled`.
   a. If the mode is not set correctly edit the `/etc/selinux/config` file with vi or a similar editor.

      ```
      root $ vi /etc/selinux/config
      ```

   b. Edit the config file so that `SELinux=` is set to `permissive` or `disabled` as required.
   c. Save the config file.
   d. Reboot the server to complete the SELinux mode change.
3. Check that you can log in to Swarm.
4. SELinux is now configured for Swarm.

Upgrading

See "Upgrade a Swarm package installation" on page 194 for details.

Uninstall

1. Remove the Swarm triggers from your Helix server.
2. Remove the Swarm trigger scripts from the server hosting your Helix server.

   > **Important**
   > If you manually installed the trigger script, perhaps because the server hosting your Helix server cannot use packages (e.g. Windows), manually remove the script.

   Follow the instructions for your OS distribution, the following examples assume that the packages are all installed on the same server:

Ubuntu, uninstalling packages:

```
sudo apt-get remove helix-swarm helix-swarm-triggers helix-swarm-optional
```

CentOS/RHEL, uninstalling packages (run this command as root):

```
yum remove helix-swarm helix-swarm-triggers helix-swarm-optional
```

Deploy and configure a Swarm VM from an OVA

Swarm is available as an OVA (Open Virtualization Appliance) that requires minimal configuration.

Use the Swarm OVA if you want to:

- Simplify the installation and configuration steps
- Experiment with Swarm without using additional hardware
- Install Swarm without having a Linux-based server available

Important
Swarm does not support Helix servers configured to use **P4AUTH**, see Centralized authentication server (P4AUTH) in the *Helix Core Server Administrator Guide*.

Note

- *Helix Core server* can refer to a Helix server machine (P4D), proxy, broker, replica, edge server, or commit server. For simplicity, the term *Helix server* is used to refer to any configuration of a Helix Core server machine.
- Swarm can be connected to Helix servers (P4D) and commit servers.

  To configure Swarm to connect to more than one Helix server (P4D), see "Multiple-Helix server instances" on page 519.

  To configure Swarm to connect to a Helix server configured to use *commit-edge architecture*, see "Commit-edge deployment" on page 476.
- Swarm must not be connected to Helix Broker, Helix Proxy, Helix Edge, forwarding replica, or read-only replica servers.

Before you begin

Important
Review the Helix server requirements before you install Swarm, see "Helix Core server requirements" on page 107.

You will need the following information to deploy the Swarm Virtual Machine (VM) from the OVA:

- The Helix Core server port (P4PORT) in the form: *my-helix-core-server:1666*

> **Important**
> If your Helix Core server is deployed using the commit-edge architecture, ensure that the Swarm port value points to the commit server.
>
> For more information, see the Commit-edge chapter in the *Helix Core Server Administrator Guide*.

- The *Userid* of a normal user in the Helix server with *admin* privileges.
- The Ticket, or password of the *admin-level* Perforce user.

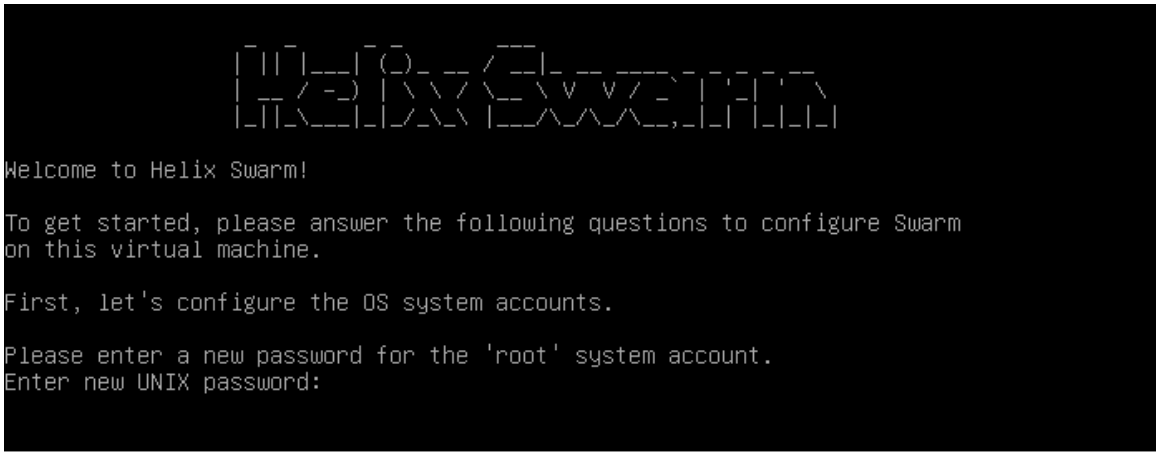
> **Important**
> If your Helix server is configured for Helix Authentication Service, you must use a long-lived login ticket for the Swarm user.

- The Mail relay host address.

Deploy the Swarm OVA

To deploy the Swarm OVA, follow the instructions on this page.

1. Download the Swarm OVA.
2. Import the OVA into your virtualization environment, see "VMWare OVA import" on page 142 or "Oracle VirtualBox import" on page 142 for details.
3. Start the VM, diagnostic and boot information is displayed as the VM starts. The configuration welcome screen is displayed when the boot is complete:



4. Follow the configuration script prompts to configure the Swarm VM passwords and hostname:

a. Set a password for the root user.

b. Set a password for the system swarm user.

c. Set a hostname for the Swarm VM.

5. The configuration script displays the following prompt:

```
Do you want to proceed with Swarm configuration (Y/n)?
```

You now have two choices:

- Proceed with VM configuration, see "Proceed with the Swarm VM configuration script" below for details.

  or

- Exit the Swarm VM configuration script to make changes from the VM operating system and then return to the Swarm configuration script. For example, you may need to edit the VM network configuration or install P4D on the same VM. See "Exit the Swarm VM configuration script" below for details.

Proceed with the Swarm VM configuration script

a. To proceed with the Swarm VM configuration script, press the **Enter** key.

b. Press the **Enter** key again to restart the Swarm VM configuration script.

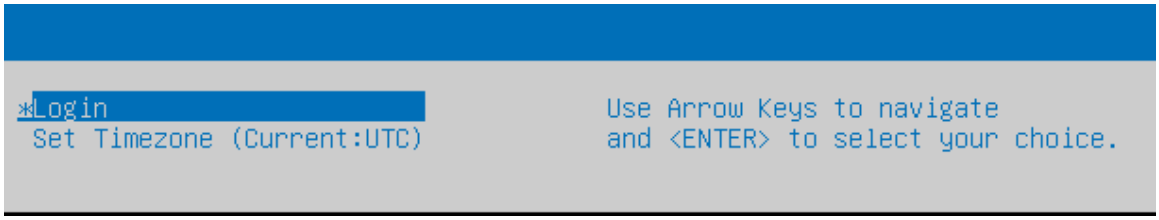
c. Finalize Swarm VM configuration, see "Finalize Swarm configuration" on the next page for details.

Exit the Swarm VM configuration script

a. To exit the Swarm VM configuration script, type **n** and press the **Enter** key

b. The Swarm configuration script exits and the welcome screen is displayed:

c. Select **Login** and press the **Enter** key.



d. Login to the VM as the *root* user.

e. Make changes to the VM as required.

f. When you have finished making changes to the VM, rerun the `configure-swarm.sh` script from the `/opt/perforce/swarm/sbin` directory:

```
root@swarm:~# /opt/perforce/swarm/sbin/configure-swarm.sh
```

g. Finalize Swarm VM configuration, see "Finalize Swarm configuration" below for details.

Finalize Swarm configuration

Follow the Swarm configuration script prompts to finalize Swarm VM configuration:

Important
If your Helix server is configured for Helix Authentication Service, the Helix server must be temporarily configured to allow fall-back to passwords while you establish a connection to the Helix server. Run the following command on the Helix server to enable fall-back to passwords:

```
p4 configure set auth.sso.allow.passwd=1
```

1. Enter the Helix Core server port (P4PORT) number.

   Important
   If your Helix Core server is deployed using the commit-edge architecture, ensure that the Swarm port value points to the commit server.

   For more information, see the Commit-edge chapter in the *Helix Core Server Administrator Guide*.

2. Enter the *Userid* of a normal user with *admin* privileges in the Helix server.
3. Enter the Ticket, or password, of the *admin-level* Perforce user.

   Important
   If your Helix server is configured for Helix Authentication Service, you must use a long-lived login ticket for the Swarm user.

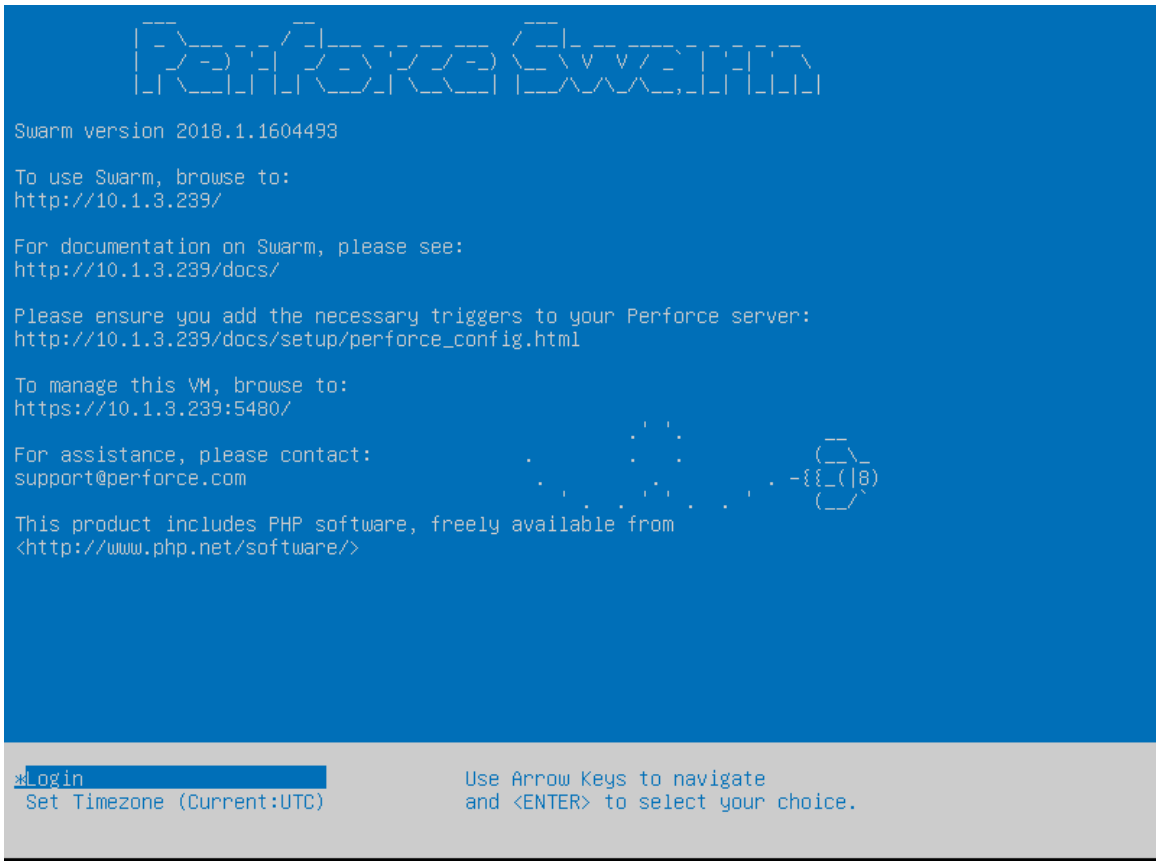
   Note
   You can obtain a login ticket by running (in another shell):

   ```
   $ p4 -p myp4host:1666 -u userid login -p
   ```

   If the login ticket you provide would expire in less than a year, you will receive a warning.

4. Enter the Mail relay host address.
5. The Swarm configuration script completes the configuration and the welcome screen is displayed.

   The welcome screen lists the links to access Swarm, its documentation, trigger configuration information, the VM management console, Perforce support, and the PHP website:



6. Update the VM with security updates and bug fixes:
   a. Use **`ssh`** to log into the VM as the *root* user.
   b. Enter the following commands to update the VM packages list and to apply any available upgrades.

      ```
      $ apt-get update
      $ apt-get upgrade
      ```

      See OVA Management for more details.

> **Note**
> Now that the Swarm VM is configured and running, you can edit the configuration by using **`ssh`** to connect to the VM as the system *swarm* user and editing the "Swarm configuration" on page 154 file **`/opt/perforce/swarm/data/config.php`**. The Swarm installation folder is **`/opt/perforce/swarm/`**.

7. The basic Swarm configuration is now complete.

> **Important**
> If your Helix server is configured for Helix Authentication Service, you can force all of your users to authenticate via your Identity Provider (IdP) by disabling fall-back to passwords. To disable fall-back to passwords on the Helix server, run the following command:
>
> ```
> p4 configure set auth.sso.allow.passwd=0
> ```

8. Establish a trigger token, see "Establish trigger token" on page 158.

VMWare OVA import

The Swarm OVA works with several VMWare virtualization products, such as Player, Workstation, and Fusion.

1. In the VMWare product, select **File > Open**.
2. Browse to the **`swarm.ova`** file and click **Open**.
3. Type a name for the virtual machine, such as *Swarm*, and click **Import**.

Oracle VirtualBox import

The Swarm OVA works with Oracle VirtualBox, version 4.x+.

1. In VirtualBox, select **File > Import Appliance**.
2. Browse to the **`swarm.ova`** file and click **Open**.
3. Click **Next** (this might be **Continue** for some versions of VirtualBox)
4. Click **Import**.

Install and configure Swarm manually from a Tarball

> **Important**
> - Review the runtime dependencies before you install Swarm, see "Runtime dependencies" on page 101.
> - **CentOS/RHEL:**
>   - **CentOS 7:** Use the Remi repository configuration package (**`remi-release-7.rpm`**) to give Swarm access to PHP 7.x. The **`sclo-php7x-php-pecl-redis`** and **`sclo-php7x-php-pecl-igbinary`** extensions are not available from this RedHat repository and you must source them from elsewhere.

- **CentOS/RHEL 8:** Use the Remi repository configuration package (**`remi-release-8.rpm`**) to give Swarm access to PHP 7.x. Use the **`epel-release-latest-8.noarch.rpm`** repository configuration package to give Swarm access to EPEL packages.
- **RHEL 7:** Use the Remi repository configuration package (**`remi-release-7.rpm`**) to give Swarm access to PHP 7.x and most of the required PHP extensions. The **`php7x-php-pecl-redis`** and **`php7x-php-pecl-igbinary`** extensions are not available from this RedHat repository and you must source them from elsewhere.

1. Download the Swarm tarball.
2. Expand the Swarm package (a *compressed tarball*).

   From the command line, expand it via the tar command:

   ```
   $ tar -zxf swarm.tgz
   ```

   The contents of the Swarm package are expanded into a top-level folder named **`swarm-version`**, where ***`version`*** corresponds to the version downloaded.

   > **Tip**
   > Many graphical file manager applications (Nautilus on Linux, Finder on Mac, etc.) can automatically expand the tarball package by simply double-clicking it.

3. Move the contents of the Swarm package to the correct location.

   Identify a location for the Swarm files; this should correspond to a location associated to the virtual host configured under Apache (see "Apache configuration" on page 148).

   ```
   $ mv /path/to/swarm-version /path/to/vhosts/swarm
   ```

4. Assign correct ownership and permission for the Swarm files.

   The data top-level folder in the Swarm distribution needs to be writeable by the web server. To achieve this effect, simply change ownership of the data folder to the web user:

   ```
   $ sudo chown -R www /path/to/vhosts/swarm/data
   ```

   The **`www`** user above is an example of what the web server user name might be. Depending on your distribution, this could be **`_www`**, **`web`**, **`nobody`** or something else entirely.

   If your web server is already running, you can discover the user with:

   ```
   $ ps aux | grep -E 'apache|httpd'
   root      3592  0.0  0.5 405240 20708 ?        Ss   May03
   4:32 /usr/sbin/apache2 -k start
   www      20016  0.0  0.2 405264  9796 ?        S    07:45
   0:00 /usr/sbin/apache2 -k start
   ```

In this example, **www** is the user Apache is running as.

From a security perspective, we recommend that the minimum file permissions should be granted to the user/group under which the web server runs against the Swarm distribution.

5. Configure Redis, see "Redis configuration" below.

Redis configuration

Swarm requires Redis to manage its caches and by default Swarm uses its own Redis server on the Swarm machine. Swarm caches data from the Helix server to improve the performance of common searches in Swarm and to reduce the load on the Helix server. Redis is included in the Swarm Tarball installation.

This section describes how to configure the Swarm Redis service on the Swarm machine. Do not change the default values of the following configuration files if you are using the Swarm Redis server.

> **Tip**
> If required, you can use your own Redis server instead of the Swarm Redis server. For instructions on how to configure Swarm to use your own Redis server, see "Use your own Redis server" on page 189.

Swarm has two Redis binaries in ***SWARM_ROOT*/p4-bin/bin.linux26x86_64/**:

- **redis-server-swarm**
- **redis-cli-swarm**

1. Configure the Redis cache database, enter the following details in the **redis-server.conf** file in **/opt/perforce/etc/**:

```
bind 127.0.0.1
port 7379
supervised auto
save ""
dir /opt/perforce/swarm/redis
```

Default values:

> **Tip**
> - The default settings are shown below, the **redis-server.conf** file contains more detailed information about the Redis configuration for Swarm.
> - On Swarm systems with a large number of users, groups, and projects, start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves, see the **redis-server.conf** file comments for detailed information.

- `bind 127.0.0.1` - Redis server IP address (loopback interface)
- `port 7379` - Redis server port number
- `supervised auto` - detects the use of **upstart** or **systemd** automatically to signal that the process is ready to use the supervisors
- `save ""` - background saves disabled, recommended.
- `dir /opt/perforce/swarm/redis` - the directory the Redis cache database is stored in.

> **Tip**
> To fine-tune your Redis server settings, make your changes in the Laminas Redis adapter. For information about the Laminas Redis adapter, see the Laminas Redis Adapter documentation.

2. The Redis cache database must be running before Swarm starts so we recommend setting it up as a service so that it starts automatically at boot time.

   The following example script will:

   - run the Redis service as the Perforce user
   - use the configuration file **in /opt/perforce/etc/redis-server.conf**
   - assume the database server is installed in **/opt/perforce/swarm/p4-bin/bin.linux26x86_64/**

   To configure Redis as a service, add a script with the following content in **/etc/systemd/system**

   redis-server-swarm.service

```
[Unit]
Description=Redis Server for Swarm
After=network.target

[Service]
ExecStart=/opt/perforce/swarm/p4-bin/bin.linux26x86_64/redis-server-
swarm /opt/perforce/etc/redis-server.conf
ExecStop=/opt/perforce/swarm/p4-bin/bin.linux26x86_64/redis-cli-swarm
shutdown
Restart=always
RestartSec=10
StandardOutput=syslog
StandardError=syslog
SyslogIndentifier=redis
User=perforce
       Group=perforce

[Install]
WantedBy=multi-user.target
```

3. Create the *SWARM_ROOT*`/data/config.php` file if it does not already exist. The `redis` block of the *SWARM_ROOT*`/data/config.php` file contains the `password`, `namespace`, `host` and `port` details of the Swarm Redis server:

```
<?php
    // this block should be a peer of 'p4'
    'redis' => array(
        'options' => array(
            'password' => null, // Defaults to null
            'namespace' => 'Swarm',
            'server' => array(
                'host' => 'localhost', // Defaults to 'localhost' or
enter your Redis server hostname
                'port' => '7379', // Defaults to '7379 or enter your
Redis server port
            ),
        ),
```

```
        'items_batch_size' => 100000,
        'check_integrity' => '03:00', // Defaults to '03:00' Use one
of the following two formats:
                                      // 1) The time of day that the
integrity check starts each day. Set in 24 hour format with leading
zeros and a : separator
                                      // 2) The number of seconds
between each integrity check. Set as a positive integer. Specify '0'
to disable the integrity check.
        'population_lock_timeout' => 300, // Timeout for initial
cache population. Defaults to 300 seconds.
    ),
```

Configurables:

- **`password`**: Redis server password. Defaults to **`null`** and should be left at default if using the Swarm Redis server.
- **`namespace`**: the prefix used for key values in the Redis cache. Defaults to **`Swarm`** and should be left at default if using the Swarm Redis server.

  > **Note**
  > If you have multiple-Swarm instances running against a single Redis server, each Swarm server must use a different Redis **`namespace`**. This enables the cache data for the individual Swarm instances to be identified. The **`namespace`** is limited to ≤ 128 characters.
  >
  > If one or more of your Swarm instances is connected to multiple-Helix servers, the Redis **`namespace`** includes the **`server label`** and the character limit is reduced to ≤ 127 characters, see "Multiple-Helix server instances" on page 519.

- **`host`**: Redis server hostname. Defaults to **`localhost`** and should be left at default if using the Swarm Redis server.
- **`port`**: Redis server port number. Defaults to **`7379`**. Swarm uses port **`7379`** as its default to avoid clashing with other Redis servers that might be on your network. It should be left at default if using the Swarm Redis server. The default port for a non-Swarm Redis server is **`6379`**.
- **`items_batch_size`**: Maximum number of key/value pairs allowed in an **`mset`** call to Redis. Sets exceeding this will be batched according to this maximum for efficiency. Defaults to **`100000`**.

> **Note**
> The default value of `100000` was chosen to strike a balance between efficiency and project data complexity. This value should not normally need to be changed, contact support before making a change to this value.

- `check_integrity`: In some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server. Swarm can run a regular integrity check to make sure that the Redis caches and Helix server are in sync. If an integrity check finds an out of sync cache file, Swarm automatically updates the data in that cache.

  The `check_integrity` configurable specifies when the Redis cache integrity check is run. Set as a specific time of day (24 hour format with leading zeros) or a number of seconds (positive integer) between checks. Disable the integrity check with `'0'`. Defaults to `'03:00'`.

- `population_lock_timeout`: specifies the timeout, in seconds, for initial cache population. If you have a large Swarm system, increase this time if the initial cache population times out. Defaults to `300` seconds.

Configure Apache, see "Apache configuration" below.

Apache configuration

The configuration of the Apache HTTP Server (Apache) can vary between OS distributions; see the documentation specific to your installation of Apache. For example, on Mac OS X, you may have to enable Web Sharing within the Sharing control panel in System Preferences.

> **Important**
> **CentOS/RHEL only:**
>
> - **Swarm 2019.1 to 2020.1:** these versions of Swarm used PHP packages from the SCL repositories for CentOS/RHEL 7. This was to provide access to more recent versions of PHP than are available as standard on CentOS/RHEL. This required the use of the `httpd24-httpd` package for Apache. These were all installed into `/opt/rh`
>
> **Swarm 2020.2:** this version of Swarm uses the Remi repository for CentOS/RHEL 7 and 8. This provides PHP 7.4 installed in the standard file system structure. This means that the old `httpd24-httpd` version of Apache is no longer needed, and the standard system version of Apache is being used again.
>
> The SCL Apache site configuration file was installed at this location for Swarm 2019.1 to 2020.1:
>
> `/opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf`

If this exists when Swarm is upgraded to 2020.2, this file is copied to **/etc/httpd/conf.d/perforce-swarm-site.conf** if there is no file at the destination. It is also re-written to change references from **/var/log/httpd24** to **/var/log/httpd**

If a site configuration file for Swarm already exists in **/etc/httpd**, the copy and re-write is not performed.

After upgrade, **httpd24-httpd** is disabled.

- To avoid seeing the Apache HTTP server Linux test page when you start the Apache server, comment out the content of the **welcome.conf** file located in the **/etc/httpd/conf.d/** directory.

1. Locate your system's Apache configuration.

   Common configuration directories include:

   - **/etc/httpd/conf/**
   - **/etc/apache2/**
   - **/Applications/XAMPP/etc/**

   Within the configuration path, the main Apache configuration file is usually named one of the following:

   - **httpd.conf**
   - **apache2.conf**

   Tip
   A longer discussion on the possible locations and names of Apache configuration files is available here: https://wiki.apache.org/httpd/DistrosDefaultLayout

2. Set up an Apache virtual host (vhost) for your installation.

   If your Apache configuration directory contains the directories **sites-available** and **sites-enabled**:

   a. Copy the appropriate virtual host definition below into the file **sites-available/swarm**.

   b. Enable the Swarm virtual host definition.

   ```
   $ sudo a2ensite swarm
   ```

   Otherwise, copy the virtual host definition below into the bottom of the main Apache configuration file, **httpd.conf** or **apache2.conf**.

Virtual host definition example for Apache 2.4:

```
<VirtualHost *:80>
    ServerName myswarm.host
    ServerAlias myswarm
    ErrorLog "/path/to/apache/logs/myswarm.error_log"
    CustomLog "/path/to/apache/logs/myswarm.access_log" common
    DocumentRoot "/path/to/swarm/public"
    <Directory "/path/to/swarm/public">
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

> **Note**
> See Apache's virtual host documentation for details:
> https://httpd.apache.org/docs/2.4/vhosts/

3. Customize the virtual host definition.
   a. Replace ***myswarm.host*** with the hostname for Swarm on your network. This may require adjusting the DNS configuration on your network.
   b. Replace ***myswarm*** with the name of the subdomain hosting Swarm. Many administrators choose swarm.

      > **Note**
      > The string ***myswarm*** in the log file paths: this should match the subdomain name and prefix for the log files, to help coordinate the active host with the log files for that host. Doing this is particularly useful when your Apache server hosts multiple instances of Swarm.

   c. Replace ***/path/to/apache/logs*** with the path where your Apache stores its log files. Apache's log files are typically named **access_log** and **error_log**.
   d. Replace ***/path/to/swarm*** with the path to the Swarm directory.
4. Verify that the correct Apache modules are enabled.
   - To query whether the PHP and Rewrite modules are active, use the apachectl utility to list all of the active modules (this may be named **apache2ctl** on your system):

     ```
     $ apachectl -t -D DUMP_MODULES
     ```

- Simply look for **`php7_module`** and **`rewrite_module`** in the output. If you see them, skip ahead to step 5.
- If the Apache utility **`a2enmod`** is installed, use it to enable the PHP and Rewrite modules:

```
$ sudo a2enmod php7 rewrite
```

- Without the **`a2enmod`** utility, edit the Apache configuration file by hand. Locate your Apache configuration file for modules and either uncomment or add the following lines:

```
LoadModule  php7_module     libexec/apache2/libphp7.so
LoadModule  rewrite_module  libexec/apache2/mod_rewrite.so
```

- Note that your Apache installation may have different paths for the location of its modules (the .so files).

5. Restart your web server.
   - To ensure that the Apache configuration changes you made become active, restart the web server.

```
$ sudo apachectl restart
```

   - Query Apache's active virtual hosts and modules to confirm your changes are in effect:

```
$ apachectl -t -D DUMP_VHOSTS
$ apachectl -t -D DUMP_MODULES
```

> **Important**
> Apache must be configured to use the prefork MPM because P4PHP does not support threaded operation.
>
> The prefork ***MPM*** is the default for Linux and OSX Apache installations, so you may not have to do anything.
>
> For more information on Apache MPMs and configuration, see: https://httpd.apache.org/docs/2.4/mpm.html

6. Configure PHP, see "PHP configuration" below.

PHP configuration

PHP can vary between OS distributions; see the documentation specific to your installation of PHP.

1. First determine which PHP `.ini` file is in use by the PHP Apache module.

   > **Note**
   > It may not necessarily be the same `.ini` file that is in use when calling PHP from the command line (running `php --ini` from the command line reports this).

   If you are having trouble determining which `.ini` file the PHP Apache module is using, create a PHP file that can be served through Apache with the following contents:

   ```
   <?php phpinfo();?>
   ```

   Point your browser to this file and look for this table row in the resulting table:

   ```
   Loaded Configuration File
   ```

2. Ensure that `date.timezone` is set correctly for your system.

   Some distributions do not make a default timezone available to PHP, so the best practice to set the timezone for PHP explicitly. See the list of supported timezones.

   An example `date.timezone` setting in `php.ini`:

   ```
   date.timezone = America/Vancouver
   ```

3. Verify that the iconv, json, and session extensions are present.

   They are usually enabled by default, although you may have to install packages for them through your OS distribution. Verify they are present by searching for their respective names in the `phpinfo` output above.

4. Enable P4PHP, the Perforce extension for PHP:

   For Swarm to communicate with the Helix Core server, it needs the P4PHP extension. P4PHP variants are available for each PHP 7 version supported by Swarm and each variant is available as either openssl 1.02 or openssl 1.1.1. All of the P4PHP variants are supplied with the Swarm package, OVA and tarball installations.

   > **Note**
   > - For Linux, the default variants are compiled with glibc 2.11.
   > - P4PHP 2019.1 or later is required for Swarm 2019.1 and later.
   >
   >   **Swarm package, OVA and tarball installations:** 2 versions of P4PHP are supplied for each PHP 7 version supported by Swarm. They are located in the `p4-bin/bin.linux26x86_64` directory.
   >
   >   - `perforce-php7x.so` compatible with systems using SSL 1.0.2
   >   - `perforce-php7x-ssl1.1.1.so` compatible with systems using SSL 1.1.1 (by default Ubuntu 18.04 and Ubuntu 20.04 use SSL 1.1.1)
   >
   >   Where `x` is the version of PHP 7.

> If the **perforce.ini** file is not pointing at the correct version of P4PHP and you connect to an SSL enabled Helix server:
>
> - The Swarm web-page will not load and you might see a **Connection Reset** error.
> - There might be an **undefined symbol: SSLeay** message in the Apache error log

To enable P4PHP, edit the web server's **php.ini** file and add the following line:

```
extension=/path/to/swarm/p4-bin/bin.<platform>/perforce-<variant>.so
```

Example 1: for a 64-bit Linux system running PHP 7.0 and SSL 1.0.2:

```
extension=/path/to/swarm/p4-bin/bin.linux26x86_64/perforce-php70.so
```

Example 2: for a 64-bit Linux system running PHP 7.1 and SSL 1.0.2:

```
extension=/path/to/swarm/p4-bin/bin.linux26x86_64/perforce-php71.so
```

Example 3: for a 64-bit Linux system running PHP 7.2 and SSL 1.0.2:

```
extension=/path/to/swarm/p4-bin/bin.linux26x86_64/perforce-php72.so
```

Example 4: for a 64-bit Linux system running PHP 7.4 and SSL 1.1.1:

```
extension=/path/to/swarm/p4-bin/bin.linux26x86_64/perforce-php74-ssl1.1.1.so
```

Alternatively, copy the extension file to the default location for PHP extensions, and then just add this line instead:

```
extension=perforce-<variant>.so
```

5. Restart Apache for the changes to become active.
6. To verify that P4PHP is active, navigate to the **phpinfo** file you created above. You should then see a perforce section (search for "Perforce Module"). It should report that the module is enabled and display the version information.

> **Note**
> Be aware that any operating system upgrades on the machine hosting Swarm may involve updates to PHP. If this occurs, the PHP **.ini** file needs to be updated to point to the correct ***variant*** of P4PHP to match the version of PHP that the upgraded operating system is using.

7. In addition, Swarm greatly benefits from the following optional extension: ImageMagick extension for PHP to improve Swarm's ability to preview graphic formats that web browsers typically cannot display, see "ImageMagick (imagick) extension for PHP" below.
8. Configure Swarm, see "Swarm configuration" below.

ImageMagick (imagick) extension for PHP

Imagick is a PHP extension that integrates the ImageMagick graphics library's API for the creation and manipulation of images. Enabling Imagick improves Swarm's ability to preview graphics formats that web browsers typically cannot display.

For more information, see:

- https://secure.php.net/imagick
- https://pecl.php.net/package/imagick

1. We recommend that you install Imagick from your OS distribution, via `apt-get`, `yum`, etc. If your distribution does not offer the imagick package for PHP, install it via PECL (although you may have to resolve system dependencies):

   ```
   $ sudo pecl install imagick
   ```

2. Verify that imagick is enabled in your PHP Apache module's PHP `.ini` file (as determined in the section above for P4PHP). You may need to add the following line:

   ```
   extension=imagick.so
   ```

3. Restart Apache for the changes to become active.
4. To verify that imagick is active, navigate to the `phpinfo` file you created earlier. You should then see an imagick section. It should report its version information and a table for its directives, supported image file formats, and more.

> **Warning**
> Once you have completed installing and enabling P4PHP and imagick, we recommend that you remove the `phpinfo` file you created to avoid disclosing information about your installation.

Swarm configuration

Now that Swarm is ready for use, you need to configure it to work in your environment.

> **Important**
> Swarm does not support Helix servers configured to use `P4AUTH`, see Centralized authentication server (P4AUTH) in the *Helix Core Server Administrator Guide*.

Note

- *Helix Core server* can refer to a Helix server machine (P4D), proxy, broker, replica, edge server, or commit server. For simplicity, the term *Helix server* is used to refer to any configuration of a Helix Core server machine.
- Swarm can be connected to Helix servers (P4D) and commit servers.

  To configure Swarm to connect to more than one Helix server (P4D), see "Multiple-Helix server instances" on page 519.

  To configure Swarm to connect to a Helix server configured to use *commit-edge architecture*, see "Commit-edge deployment" on page 476.
- Swarm must not be connected to Helix Broker, Helix Proxy, Helix Edge, forwarding replica, or read-only replica servers.

Swarm configuration file

Important

- If your Helix server is configured for Helix Authentication Service, the Helix server must be temporarily configured to allow fall-back to passwords while you establish a connection to the Helix server. Run the following command on the Helix server to enable fall-back to passwords:

  `p4 configure set auth.sso.allow.passwd=1`

Edit the *SWARM_ROOT*`/data/config.php` file so that it contains the following configuration block:

```
<?php
    return array(
        'p4' => array(
            'port'      => 'my-helix-core-server:1666',
            'user'      => 'admin_userid',
            'password'  => 'admin user ticket or password',
            'sso_enabled' => false, // defaults to false
        ),
        'log' => array(
            'priority'  => 3, // 7 for max, defaults to 3
        ),
        'mail' => array(
            'transport' => array(
                'host' => 'my.mx.host',
            ),
```

```
        ),
    );
```

- For the port value, replace *my-helix-core-server:1666* with the P4PORT value used to connect to your Helix server.

  Important
  If your Helix server is deployed using the commit-edge architecture, ensure that Swarm's port value points to the commit server.

  For more information, see Commit-edge in the *Helix Core Server Administrator Guide*.

  Warning
  If the port points to a Helix Broker, ensure that the broker does not delegate commands to different replicas, edge servers, or proxies. Such delegation can cause odd problems or outright failures in Swarm.

  Swarm needs to have a consistent, current view of the state of Helix server, and works best when it connects to a central/commit server.

- For the user value, replace ***admin_userid*** with a normal Helix server userid that has ***admin***-level access to Helix server.

  Note
  - This user is used by Swarm to communicate with the Helix server. It should not be used to perform everyday Swarm reviewing tasks.
  - Helix server 2020.1 and later, permissions have changed for viewing and editing stream spec files in Swarm. To view and edit stream spec files in Swarm, the Swarm user must have *admin* permissions for the entire depot **//...**

- For the password value, while a plain-text password works, we recommend that you use a ticket value instead. Obtain the ticket value for the ***admin_userid*** during login with this command:

  ```
  $ p4 -p my-helix-core-server:1666 -u admin_userid login -p
  ```

  Important
  If your Helix server authentication is configured in the one of the following ways, ticket-based authentication is required:
  - Authentication configured with security level 3.
  - Authentication configured for LDAP.
  - Authentication configured for Helix Authentication Service.

  You can determine when the admin userid's ticket will expire with:

```
$ p4 -p my-helix-core-server:1666 -u admin_userid -P ticket_value
login -s
```

For more information about tickets, see the section Ticket-based authentication in the *Helix Core Server Administrator Guide*.

- Swarm can be configured for the Helix Authentication Service when the Helix server is also configured for Helix Authentication Service.

  To enable Helix Authentication Service in Swarm, set **`sso_enabled`** to **`true`**.

- For the host value, replace ***`my.mx.host`*** with the hostname of the mail exchanger service that Swarm should use to send its email notifications.

  > **Note**
  > Since this configuration file contains the credentials for a Helix server *admin*-level user, we recommend that this file's ownership and permissions be adjusted such that only the web server user can read the file, and that no user can write the file.

- You can now configure some additional functionality for Swarm:
  - **Optional:** JIRA integration, see "JIRA" on page 447.
  - **Optional:** LibreOffice integration, see "LibreOffice" on page 451.
  - **Optional:** Manually specify the Swarm hostname, see "Swarm hostname" below.
- Establish a trigger token, see "Establish trigger token" on the next page.

Optional additional Swarm configuration

Swarm provides optional functionality that could be enabled at this time:

- JIRA integration
- LibreOffice

Swarm hostname

Swarm normally auto-detects the hostname it operates under. In some system configuration, the auto-detection logic might not choose the correct hostname, such as when there are multiple virtual hosts configured for a single Swarm instance. When auto-detection chooses the wrong hostname, email notifications, worker start up, and more could be affected.

If you need to specify the Swarm hostname, see "hostname" on page 496 for details.

Common installation and post-installation steps

This chapter covers the installation and post-installation steps that are common for all installation processes.

Establish trigger token

Trigger tokens prevent unwanted events from influencing Swarm operations; trigger requests to Swarm without a valid token are ignored.

1. Log in to Swarm as a *super* user.
2. Click your userid, found at the right of the main toolbar.
3. Select **About Swarm**.

   The **About Swarm** dialog appears and Swarm generates an API token if it doesn't exist.
4. Make a note of the trigger token value displayed at the bottom of the dialog, you will need the trigger token when you configure Helix server for Swarm.

   Tip
   Click on the trigger token to select it and then copy it to your clipboard. You can paste it into the `swarm-trigger.conf` file when you configure Helix server for Swarm.
5. Configure the Swarm triggers, see "Helix Core server configuration for Swarm" below.

Helix Core server configuration for Swarm

Now that you have a configured instance of Swarm, the last piece is to configure your Helix server to tell Swarm about interesting events. This is accomplished through the use of triggers.

For more information about Helix server triggers, see Using triggers to customize behavior in *Helix Core Server Administrator Guide*.

Important
Swarm does not support Helix servers configured to use `P4AUTH`, see Centralized authentication server (P4AUTH) in the *Helix Core Server Administrator Guide*.

Note

- *Helix Core server* can refer to a Helix server machine (P4D), proxy, broker, replica, edge server, or commit server. For simplicity, the term *Helix server* is used to refer to any configuration of a Helix Core server machine.
- Swarm can be connected to Helix servers (P4D) and commit servers.

  To configure Swarm to connect to more than one Helix server (P4D), see "Multiple-Helix server instances" on page 519.

  To configure Swarm to connect to a Helix server configured to use *commit-edge architecture*, see "Commit-edge deployment" on page 476.

- Swarm must not be connected to Helix Broker, Helix Proxy, Helix Edge, forwarding replica, or read-only replica servers.

Using triggers to push events to Swarm

Helix server provides a facility called *triggers* to customize the operation of the server, or to invoke additional processing for specific kinds of versioning operations. Swarm provides a trigger script written in Perl that notifies Swarm about activity within the Helix server.

See "Trigger options" on page 589 for more information on configuring the Perl trigger.

Set up Swarm triggers with a Windows or Linux-hosted Helix server

Tip
Many of the steps in this procedure are common to both Windows and Linux-hosted Helix servers. When there is a difference between the Windows and Linux operating systems, this is indicated in the procedure.

1. Ensure that the required "Trigger dependencies" on page 109 have been installed on the machine hosting the Helix server.
2. Copy the Swarm trigger to the Helix server
   - If your Helix server is version 2014.1 (or later), we recommend submitting the trigger script, **`p4-bin/scripts/swarm-trigger.pl`**, to Helix server and running it from the depot. The recommended depot location is **`//.swarm/triggers/swarm-trigger.pl`**, especially if you have already setup "Comment attachments" on page 469.
   - If your Helix server is older than version 2014.1, or you prefer that the trigger file exist in the filesystem, you must copy the **`p4-bin/scripts/swarm-trigger.pl`** script to the server hosting Helix server. If your Helix server deployment uses the commit-edge architecture, the script must also be copied to all edge servers, and it must exist in the same path on all servers.

   Note
   If you are using the Swarm OVA, the full path to the trigger script within the OVA's filesystem is: **`/opt/perforce/swarm/p4-bin/scripts/swarm-trigger.pl`**.
3. Configure the trigger.

   You need to use the API token established in the "Establish trigger token" on the previous page.

   `swarm-trigger.pl` can be configured directly, but the preferred approach is to create a configuration file called **`swarm-trigger.conf`**. Using the configuration file greatly simplifies upgrades.

- If you are using the Swarm triggers package described in "Install and configure Swarm from a package" on page 115, the configuration file is available at **/opt/perforce/etc/swarm-trigger.conf**
  - If you submitted the trigger script to the depot in the previous step, you should similarly submit the configuration file to the depot. The recommended depot location is **//.swarm/triggers/swarm-trigger.conf**.
  - If you copied the trigger script to the commit server and all edge servers in the previous step, also copy the configuration file to the commit server and all edge servers, making sure that they exist in the same path on all servers.
- If you are not using the Swarm trigger package, create **swarm-trigger.conf** in the same directory as **swarm-trigger.pl**.
  - If you submitted the trigger script to the depot in the previous step, you should similarly submit the configuration file to the depot. The recommended depot location is **//.swarm/triggers/swarm-trigger.conf**.
  - If you copied the trigger script to the commit server and all edge servers in the previous step, also copy the configuration file to the commit server and all edge servers, making sure that they exist in the same path on all servers.

The following is a sample of what your **swarm-trigger.conf** should contain:

```
# SWARM_HOST (required)
# Hostname of your Swarm instance, with leading "http://" or
"https://".
SWARM_HOST="http://my-swarm-host"

# SWARM_TOKEN (required)
# The token used when talking to Swarm to offer some security. To
obtain the
# value, log in to Swarm as a super user and select 'About Swarm' to
see the
# token value.
SWARM_TOKEN="MY-UUID-STYLE-TOKEN"

# ADMIN_USER (optional) Do not use if the Workflow feature is
enabled (default)
# For enforcing reviewed changes, optionally specify the normal
Perforce user
# with admin privileges (to read keys); if not set, will use whatever
```

```
Perforce
# user is set in environment.
ADMIN_USER=

# ADMIN_TICKET_FILE (optional) Do not use if the Workflow
feature is enabled (default)
# For enforcing reviewed changes, optionally specify the location of
the
# p4tickets file if different from the default ($HOME/.p4tickets).
# Ensure this user is a member of a group with an 'unlimited' or very
long
# timeout; then, manually login as this user from the Perforce server
machine to
# set the ticket.
ADMIN_TICKET_FILE=

# VERIFY_SSL (optional)
# If HTTPS is being used on the Swarm web server, then this controls
whether
# the SSL certificate is validated or not. By default this is set to
1, which
# means any SSL certificates must be valid. If the web server is
using a self
# signed certificate, then this must be set to 0.
VERIFY_SSL=1
```

Modify the **`swarm-trigger.conf`** configuration file to set the **`SWARM_HOST`** and the **`SWARM_TOKEN`** variables appropriately.

> **Note**
> **Windows only:**
>
> You may need to edit the trigger script to specify the full path to **`curl.exe`** if the **`HTTP::Tiny`** module is not installed.

> **Note**
> **Linux only:**

swarm-trigger.pl looks for configuration in the following files. Variables defined in the later files will override the earlier defined variables of the same name:

- Variables set inside the **swarm-trigger.pl** script itself
- **/etc/perforce/swarm-trigger.conf**
- **/opt/perforce/etc/swarm-trigger.conf**
- The **swarm-trigger.conf** file stored in the same directory as **swarm-trigger.pl**.
- Any file passed to the **swarm-trigger.pl** script using the -c command line argument

Important
Windows and Linux:

If you specify **ADMIN_USER**, the ticket contained in **%USERPROFILE%/p4tickets.txt** (Windows), **$HOME/.p4tickets** (Linux), or the optional ticket file specified with the **ADMIN_TICKET_FILE** must use the port that was used to start Helix server. For example, if **p4d** is started with:

```
C:\> p4d -p my-helix-core-server:1666 ...
```

then the ticket for the admin user specified with ADMIN_USER should be established with:

```
C:\> p4 -p my-helix-core-server:1666 -u admin_userid login
```

If the ticket was established using the wrong port, the error message you encounter includes the port that the trigger is attempting to use:

```
'swarm.strict.1' validation failed: Invalid login credentials to
[port] within this trigger script; please contact your
administrator.
```

4. **Linux only**: Ensure that the script has execute permissions.

   Important
   Skip this step if you have committed the script to the Helix server.

   ```
   $ chmod +x /path/to/swarm-trigger.pl
   ```

5. Verify that the trigger script executes correctly.
   - **For Windows:**

     Run:

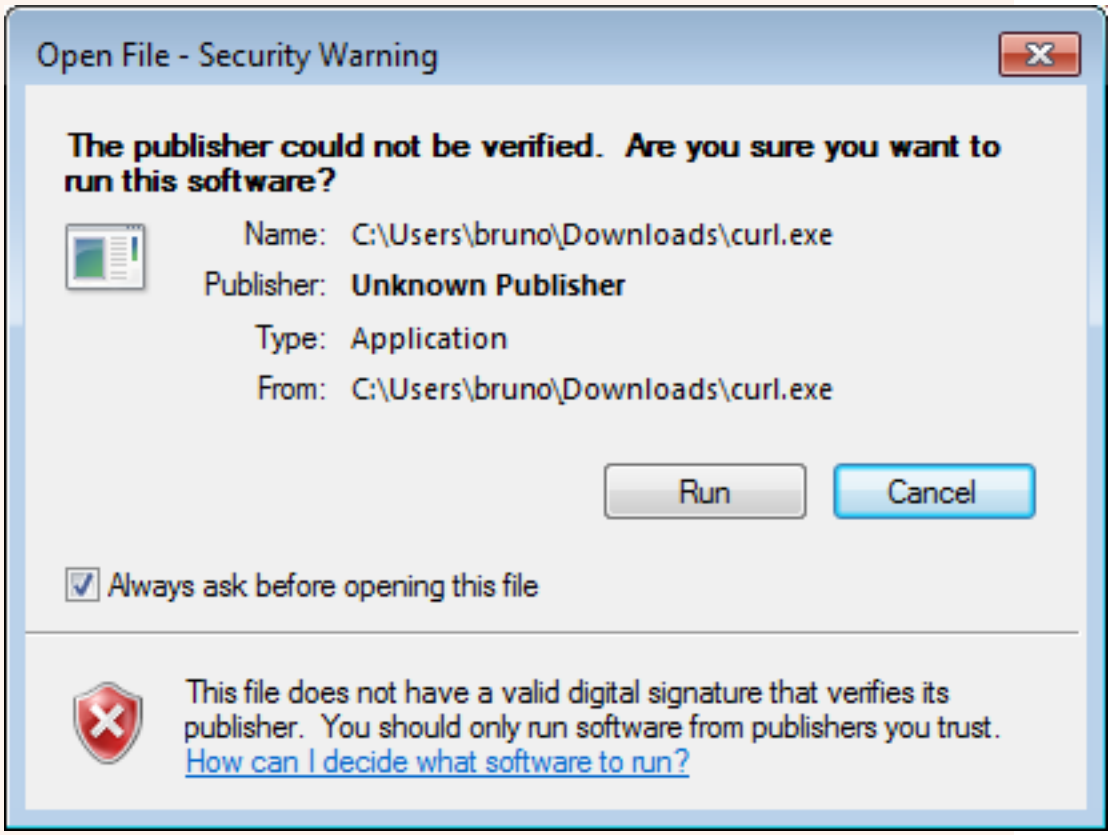
```
C:\>  perl "C:\path\to\swarm-trigger.pl" -t ping -v 0
```

Use the full path to `perl` if it is not available in your command path.

You should expect to see no output. If the trigger is misconfigured, such as using an invalid trigger token, you would see an error.

Warning
Only if using Curl:

Installation of the triggers may cause a security warning dialog to appear when `curl.exe` executes:



If this occurs, the triggers hang, creating zombie Perl processes. Due to the way triggers are invoked by Helix server, the dialog is normally not visible even though Windows is waiting on interaction.

To resolve this:

a. Clear the **Always ask before opening this file** check box and click **Run**.

b. Right-click `curl.exe`, select **Properties**, and click **Unblock**.

- **For Linux:**

  > **Important**
  > Skip this step if you have committed the script to the Helix server.

  ```
  $ /path/to/swarm-trigger.pl -t ping -v 0
  ```

  You should expect to see no output. If the trigger is misconfigured, such as using an invalid trigger token, you would see an error.

  > **Note**
  > Run the trigger script without any arguments to see additional usage information.

6. Update the Helix server triggers table to run the trigger script.

> **Warning**
> - The **`swarm.shelvedel shelve-delete`** trigger line was added to Swarm in version 2018.1 and updated in version 2020.1.
>   - **New installation or upgrading from Swarm 2017.4 and earlier:** add the **`swarm.shelvedel shelve-delete`** trigger line to the Helix server trigger table if it is not already present.
>   - **Upgrading from Swarm 2018.x and 2019.x:** replace the existing **`swarm.shelvedel shelve-delete`** trigger line in the Helix server trigger table with the one supplied in the Swarm version you are upgrading to.
> - **Workflow feature:**

The Workflow feature is enabled by default in Swarm 2019.2 and later. The trigger lines required when workflow is enabled are different to those required when workflow is disabled:

- **Workflow feature** enabled **(default):**
  - Comment out the **`swarm.enforce.1`**, **`swarm.enforce.2`**, **`swarm.strict.1`**, and **`swarm.strict.2`** trigger lines in the Helix server trigger table if they are present.
  - Add the **`swarm.enforce change-submit`**, **`swarm.strict change-content`**, and **`swarm.shelvesub shelve-submit`** trigger lines to the Helix server trigger table if they are not already present.
- **Workflow feature** disabled**:**
  - Comment out the **`swarm.enforce change-submit`**, **`swarm.strict change-content`**, and **`swarm.shelvesub shelve-submit`** trigger lines in the Helix server trigger table if they are present.
  - The **`swarm.enforce.1`**, **`swarm.enforce.2`**, **`swarm.strict.1`**, and **`swarm.strict.2`** trigger lines are commented out because they are optional. They require that the **`DEPOT_PATH1`** and **`DEPOT_PATH2`** values are configured appropriately. Support for these triggers will be dropped in a later release.
    - The first two lines configure the **`enforce`** feature, which rejects any submitted changes that are not tied to an approved review.
    - The second two lines configure the **`strict`** feature, which rejects any submitted changes when the contents of the changelist do not match the contents of its associated approved review.
  - If you need to apply **`enforce`** or **`strict`** to more depot paths, copy the lines and tweak the depot paths as necessary.

Tip

The trigger script can provide the list of trigger lines that should work, with little to no adjustment, by executing it with the **`-o`** option:

Windows

```
C:\> perl "C:\path\to\swarm-trigger.pl" -o
```

Linux

```
$ /path/to/swarm-trigger.pl -o
```

Tip

- The **`%quote%c:\path\to\perl.exe%quote%`** entry is only required for Windows-hosted systems.
- An initial tabbed indent is required for each trigger line.
- Commented out trigger lines are not stored.
- If you disable the workflow feature in the Swarm **`config.php`** file, workflow will not be processed by Swarm but a small overhead is still incurred by the Helix server each time it runs a workflow trigger script. This overhead can be eliminated by commenting out the **`swarm.enforce change-submit`**, **`swarm.strict change-content`**, and **`swarm.shelvesub shelve-submit`** workflow triggers.

As a Helix server user with *super* privileges, edit the Helix server trigger table by running the **`p4 triggers`** command and add the following lines (including the initial tab character). Update the **`perl.exe`**, trigger script, and configuration file paths in each line below to reflect the actual paths on your Helix server:

a. If you have committed both the trigger script and the configuration file to the Helix server:

```
	swarm.job	form-commit	job
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t job
  -v %formname%"
	swarm.user	form-commit	user
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t user
  -v %formname%"
	swarm.userdel	form-delete	user
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t userdel
  -v %formname%"
	swarm.group	form-commit	group
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t group
  -v %formname%"
	swarm.groupdel	form-delete	group
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t groupdel
```

```
  -v %formname%"
       swarm.changesave form-save     change
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t
changesave -v %formname%"
       swarm.shelve     shelve-commit  //...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t shelve
  -v %change%"
       swarm.commit     change-commit  //...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t commit
  -v %change%"
       swarm.shelvedel  shelve-delete  //...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t
shelvedel  -v %change% -w %client% -u %user% -d
%quote%%clientcwd%^^^%quote% -a %quote%%argsQuoted%%quote% -s
%quote%%serverVersion%%quote%"
#      The following three triggers are used by workflow. If workflow
is disabled in the Swarm
#      configuration then they should be disabled.
       swarm.enforce    change-submit  //...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t
checkenforced -v %change% -u %user%"
       swarm.strict     change-content //...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t
checkstrict   -v %change% -u %user%"
       swarm.shelvesub  shelve-submit  //...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t
```

```
checkshelve   -v %change% -u %user%"
#      The following triggers are only used to prevent a commit
without an approved review.
#      They predate the workflow functionality and should only be used
if workflow is disabled.
#      Support for these will be dropped in a later release.
#      swarm.enforce.1 change-submit  //DEPOT_PATH1/...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t enforce
-v %change% -p %serverport%"
#      swarm.enforce.2 change-submit  //DEPOT_PATH2/...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t enforce
-v %change% -p %serverport%"
#      swarm.strict.1  change-content //DEPOT_PATH1/...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t strict -
v %change% -p %serverport%"
#      swarm.strict.2  change-content //DEPOT_PATH2/...
"%quote%c:\path\to\perl.exe%quote% %//.swarm/triggers/swarm-
trigger.pl% -c %//.swarm/triggers/swarm-trigger.conf% -t strict -
v %change% -p %serverport%"
```

b. If you have copied the trigger script and configuration file to common paths on all servers:

```
       swarm.job        form-commit   job
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
job        -v %formname%"
       swarm.user       form-commit   user
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
user       -v %formname%"
       swarm.userdel    form-delete   user
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
userdel    -v %formname%"
       swarm.group      form-commit   group
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
group      -v %formname%"
       swarm.groupdel   form-delete   group
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
groupdel   -v %formname%"
       swarm.changesave form-save     change
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
changesave -v %formname%"
       swarm.shelve     shelve-commit  //...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
shelve     -v %change%"
       swarm.commit     change-commit  //...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
commit     -v %change%"
       swarm.shelvedel  shelve-delete  //...
```

```
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
shelvedel  -v %change% -w %client% -u %user% -d
%quote%%clientcwd%^^^%quote% -a %quote%%argsQuoted%%quote% -s
%quote%%serverVersion%%quote%"
#      The following three triggers are used by workflow. If workflow
is disabled in the Swarm
#      configuration then they should be disabled.
       swarm.enforce    change-submit  //...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
checkenforced -v %change% -u %user%"
       swarm.strict     change-content //...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
checkstrict   -v %change% -u %user%"
       swarm.shelvesub  shelve-submit  //...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
checkshelve   -v %change% -u %user%"
#      The following triggers are only used to prevent a commit
without an approved review.
#      They predate the workflow functionality and should only be used
if workflow is disabled.
#      Support for these will be dropped in a later release.
#      swarm.enforce.1 change-submit  //DEPOT_PATH1/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
enforce -v %change% -p %serverport%"
#      swarm.enforce.2 change-submit  //DEPOT_PATH2/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
enforce -v %change% -p %serverport%"
```

```
#      swarm.strict.1  change-content //DEPOT_PATH1/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
strict -v %change% -p %serverport%"
#      swarm.strict.2  change-content //DEPOT_PATH2/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
strict -v %change% -p %serverport%"
```

Warning
Windows and Linux:

The use of **`%quote%`** is not supported on 2010.2 servers (it is harmless though); if you are using this version, ensure that you do not have any spaces in the path to **`perl.exe`** or the script's path.

Important
Workflow feature disabled**:**

If your Helix server has SSL enabled and is older than the 2014.1 release, the **`%serverport%`** trigger variable does not include the necessary transport indicator, which can cause the **`enforce`** and **`strict`** triggers to fail.

To solve this problem, add **`ssl:`** immediately before **`%serverport%`** in the trigger lines. For example:

```
#       The following triggers are only used to prevent a commit without
an approved review.
#       They predate the workflow functionality and should only be used
if workflow is disabled.
#       Support for these will be dropped in a later release.
#       swarm.enforce.1 change-submit  //DEPOT_PATH1/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
enforce -v %change% -p ssl:%serverport%"
#       swarm.enforce.2 change-submit  //DEPOT_PATH2/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
enforce -v %change% -p ssl:%serverport%"
#       swarm.strict.1  change-content //DEPOT_PATH1/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
strict -v %change% -p ssl:%serverport%"
#       swarm.strict.2  change-content //DEPOT_PATH2/...
"%quote%C:\path\to\perl.exe%quote% %quote%/path/to/swarm-
trigger.pl%quote% -c %quote%/path/to/swarm-trigger.conf%quote% -t
strict -v %change% -p ssl:%serverport%"
```

7. Configure the Helix server to promote all shelved changes.

   When this configurable is set, Swarm has access to all shelved changelists, which is a requirement for pre-commit reviews. When it is not set, users connected to an edge server must remember to use the `-p` option when shelving files to promote their shelves to the commit server when initiating a pre-commit review.

   ```
   p4 configure set dm.shelve.promote=1
   ```

8. Optionally forward logins to the commit server.

   If you intend to use P4V and its Swarm integration, you should consider forwarding logins to the commit server. See "P4V Authentication" on page 477 for details.

9. You can now configure additional functionality for Swarm:
   - **Optional:** Hide Swarm storage from regular users, see "Hiding Swarm storage from regular users" below.
   - **Optional:** Handle exclusive locks, see "Handling Exclusive Locks" on the next page.

10. Depending on your installation type, do one of the following:
    - **Swarm package installation:** Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
    - **Swarm OVA installation:** Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" on page 178.
    - **Swarm tarball installation:** Set up a recurring task to spawn workers, see "Set up a recurring task to spawn workers" on the next page.

Hiding Swarm storage from regular users

Swarm information storage uses Helix server's ***keys*** facility. By default, users with *list*-level access can search keys and potentially obtain information they would not otherwise have access to, and users with *review*-level access can write or modify keys potentially corrupting or destroying data.

We recommend that you set the **`dm.keys.hide`** configurable to 2 to require *admin*-level access for searching and modifying keys. Note that **`dm.keys.hide`** is available in Helix server versions 2013.1 and newer.

When **`dm.keys.hide`** is set to 2, both the **`p4 keys`** and **`p4 key`** commands require *admin*-level access in the Helix server. When **`dm.keys.hide`** is set to 1, only the **`p4 keys`** command requires *admin*-level access in the Helix server. When **`dm.keys.hide`** is set to 1, or is not set, users who know (or can deduce) key names can read values (if they have *list*-level access) or write values (if they have *review*-level access) with the **`p4 key`** command.

To set **`dm.keys.hide`**:

```
$ p4 configure set dm.keys.hide=2
```

To confirm the current value of **`dm.keys.hide`**:

```
$ p4 configure show dm.keys.hide
```

To unset **dm.keys.hide**:

```
$ p4 configure unset dm.keys.hide
```

Handling Exclusive Locks

Swarm takes copies of files when it is creating reviews. Some of the files managed by Helix server may be limited to 'exclusive open' by having the filetype modifier '+l' set. This file-level setting ensures only one user at a time can open the file for editing.

To allow Swarm to work with these 'exclusive open' files, you must enable **filetype.bypasslock** in the Helix server configuration.

To set **filetype.bypasslock**:

```
$ p4 configure set filetype.bypasslock=1
```

To confirm the current value of **filetype.bypasslock**:

```
$ p4 configure show filetype.bypasslock
```

To unset **filetype.bypasslock**:

```
$ p4 configure unset filetype.bypasslock
```

If this setting is not enabled in Helix server, Swarm will report exceptions when working with exclusively opened files similar to "Cannot unshelve review (x). One or more files are exclusively open", and noting that you must have the **filetype.bypasslock** configurable enabled.

Set up a recurring task to spawn workers

To ensure that incoming Helix server events are automatically processed by Swarm, it is important to set up a cron job to do this. We recommend that the cron job is installed on the Swarm host machine.

Install curl or wget

You must install either **curl** or **wget** on the Swarm server machine to run the recurring task that spawns the Swarm workers:

- To install **curl**, see "Install curl" below.
- To install **wget**, see "Install wget" on the facing page.

Install curl

1. Download **curl** from: https://curl.haxx.se/download.html
2. Install **curl**.
3. Verify that **curl** is installed, see "Verify that curl or wget is installed" on the facing page.

Warning
If **`curl`** cannot execute as expected, trigger execution may block or fail. Prior to configuring the triggers, verify that **`curl`** executes, run:

```
$ curl -h
```

The start of the output should be similar to:

```
Usage: curl [options...] <url>
Options: (H) means HTTP/HTTPS only, (F) means FTP only
     --anyauth       Pick "any" authentication method (H)
 -a, --append        Append to target file when uploading (F/SFTP)
     --cacert FILE   CA certificate to verify peer against (SSL)
     --capath DIR    CA directory to verify peer against (SSL)
...[truncated for brevity]...
```

For a more thorough test that actually fetches content over a network, try the following test:

```
$ curl https://www.perforce.com/
```

The output should look like HTML.

Install wget

1. Download **wget** from: https://ftp.gnu.org/gnu/wget/
2. Install **wget**.
3. Verify that **wget** is installed, see "Verify that curl or wget is installed" below.

Verify that curl or wget is installed

Warning
`curl` or **`wget`** must be installed or workers do not spawn and Swarm cannot process any events.

Verify that curl or wget is installed:

1. Use the **`which`** command.

   For example to verify that **curl** is installed:

   ```
   $ which curl
   ```

2. If you see any output, the referenced command is installed.

Create a recurring task to spawn workers

> **Note**
> **Swarm package installation** and **Swarm OVA installation**:
>
> - The **helix-swarm**, **swarm-cron-hosts.conf**, and **swarm-cron.sh** files are automatically created with the correct content including the correct URL for Swarm. They do not need to be edited.
> - If you want to run Swarm in a sub-folder of an existing website, or with a custom port, the **swarm-cron-host.conf** file must be edited. For instructions on how to configure Swarm to run in a sub-folder, or with a custom port, see Run Swarm in a sub-folder of an existing web site, or Run Swarm's virtual host on a custom port.

helix-swarm cron file

The **helix-swarm** file is located in the **/etc/cron.d** and points to the **swarm-cron.sh** script file.

1. Create a file named **helix-swarm** in **/etc/cron.d** if it does not already exist.
2. Edit the **helix-swarm** file so that it has the following content:

```
#
# Cron job to start Swarm workers every minute
#
* * * * * nobody [ -x /opt/perforce/swarm/p4-bin/scripts/swarm-
cron.sh ] && /opt/perforce/swarm/p4-bin/scripts/swarm-cron.sh
```

3. Save the **helix-swarm** file.

swarm-cron.sh script

The **swarm-cron.sh** file in **/opt/perforce/swarm/p4-bin/scripts** ensures that a worker is fired up every minute.

By default, the **DEFAULT_CONFIG_FILE=** configuration line is set to **/opt/perforce/etc/swarm-cron-hosts.conf**. The **swarm-cron.sh** script takes the Swarm hostname from the **swarm-cron-hosts.conf** file.

The **swarm-cron.sh** script contains the correct content for Swarm installations using **wget**, or **curl**.

> **Tip**
> You must leave the **wget**, and **curl** configuration lines in the script. The script is written so that it can be used with Swarm installations that use **wget**, and Swarm installations that use **curl**. The **wget** configuration line is tried first, if that fails the **curl** configuration line is tried.

- **`wget`** configuration line for HTTP, and HTTPS:

```
wget --quiet --no-check-certificate --output-document /dev/null --
timeout 5 "${SWARM_HOST}/queue/worker"
```

- **`curl`** configuration line for HTTP, and HTTPS:

```
curl --silent --insecure --output /dev/null --max-time 5 "${SWARM_
HOST}/queue/worker"
```

In the **`wget`** and **`curl`** configuration lines above, where you see **`--timeout 5`** or **`--max-time 5`**, the **`5`** is the number of seconds that the cron task will wait for a response from the Swarm host. When the cron task is installed on the Swarm host, such as in the Swarm OVA, that value could be reduced to **`1`** second (e.g. **`--timeout 1`** or **`--max-time 1`**).

> **Note**
> If you configure Swarm to use HTTPS, and you install a self-signed certificate, the cron jobs avoid the certificate validity test which could cause silent failures to process events. The command to avoid the certificate validity test is included by default for **`wget`**, and **`curl`**:
>
> - wget command: **`--no-check-certificate`**
> - curl command: **`--insecure`**
>
> If you have configured an HTTP connection to Swarm, the avoid validity check command is ignored.

swarm-cron-hosts.conf configuration file

The **`swarm-cron-hosts.conf`** file in **`/opt/perforce/etc`** specifies the connection type (HTTP or HTTPS), hostname, and port number for Swarm cron jobs.

1. Edit the **`swarm-cron-hosts.conf`** file so that is contains the actual Swarm hostname, and port you have configured for Swarm (which may include a sub-folder or a custom port). The following format is used:

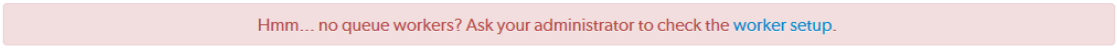
   ```
   [http[s]://]<swarm-host>[:<port>]
   ```

   Default if value not specified:

   - **`[http[s]://]`** http
   - **`<swarm-host>`** must be specified
   - **`:<port>`** 80

2. Save the edited file.

> **Tip**
> To check or modify Swarm worker configuration, see "Workers" on page 603.

> **Note**
> If the recurring task is disabled, or stops functioning for any reason, logged-in users see the following error message when Swarm detects that no workers are running:
>
> 
>

3. The basic Swarm configuration is now complete.

> **Important**
> If your Helix server is configured for Helix Authentication Service, you can force all of your users to authenticate via your Identity Provider (IdP) by disabling fall-back to passwords. To disable fall-back to passwords on the Helix server, run the following command:
>
> ```
> p4 configure set auth.sso.allow.passwd=0
> ```

4. Review the post-install configuration options to customize your Swarm installation, see "Post-install configuration options" below.

Post-install configuration options

There are a few options for customizing your Swarm installation's operation. This section covers the options that are officially supported.

This section contains:

No customization required

If you do not need to customize your Swarm installation, your Swarm installation is complete but you must check that it is working correctly before using it in production. For instructions on how to check your Swarm installation, see "Validate your Swarm installation" on page 192.

Back up your Swarm virtual host first

Before undertaking any of the following customization options, ensure that you have backed up your Swarm virtual host configuration. Choose the most appropriate option:

- If your Apache configuration directory contains the directories `sites-available` and `sites-enabled`:

```
$ cd /path/to/apache/configuration/..
$ cp -a sites-available sites-available.bak
```

> **Important**
> If the `sites-enabled` directory contains files, and not just symbolic links, you need to backup this folder as well:
>
> ```
> $ cd /path/to/apache/configuration/..
> $ cp -a sites-enabled sites-enabled.bak
> ```

- For CentOS/RHEL systems, if you used the Swarm packages to install Swarm:

```
$ cd /path/to/apache/configuration/..
$ cp -a conf.d conf.d.bak
```

- Otherwise, back up your Apache configuration.

HTTPS

> **Important**
> Back up your Swarm virtual host configuration before customizing your Swarm installation, see "Back up your Swarm virtual host first" on the previous page.

This section describes how to make your Swarm installation more secure by using HTTPS.

Before you begin the following procedure, locate your system's Apache configuration. Common configuration directories include:

- `/etc/httpd/conf/`
- `/etc/apache2/`
- `/Applications/XAMPP/etc/`

Within the Apache configuration path, the main Apache configuration file is usually named one of the following:

- `httpd.conf`
- `apache2.conf`

A longer discussion on the possible locations and names of Apache configuration files is available here: https://wiki.apache.org/httpd/DistrosDefaultLayout

1. Enable SSL in Apache.

   If the Apache utility **a2enmod** is installed:

   ```
   $ sudo a2enmod ssl
   ```

   Without the **a2enmod** utility, edit the Apache configuration file by hand. Locate your Apache configuration file for modules and either uncomment or add the following lines:

   ```
   LoadModule  ssl_module  libexec/apache2/mod_ssl.so
   ```

2. Create a directory to store certificates.

   ```
   $ sudo mkdir -p /etc/apache2/ssl
   ```

3. Create a certificate/key pair.

   ```
   $ cd /etc/apache2/ssl
   $ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
   keyout apache.key -out apache.crt
   ```

   This command generates a private key and a certificate. To form the certificate, **openssl** prompts you for several details:

   ```
   Generating a 2048 bit RSA private key
   ...................+++
   ...................................+++
   writing new private key to 'apache.key'
   -----
   You are about to be asked to enter information that will be
   incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or
   a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:CA
   State or Province Name (full name) [Some-State]:British Columbia
   Locality Name (eg, city) []:Victoria
   Organization Name (eg, company) [Internet Widgits Pty Ltd]:Perforce
   ```

```
Software
Organizational Unit Name (eg, section) []:Swarm development team
Common Name (e.g. server FQDN or YOUR name) []:myswarm.host
Email Address []:admin@myswarm.host
```

The output above includes some example details. You should replace anything in italics with your own details. Since the certificate request details that can help users determine whether your certificate is valid, enter legitimate information whenever possible.

> **Important**
> The Common Name field must match the hostname for your Swarm installation exactly.

4. Secure the certificate directory.

```
$ sudo chmod 600 /etc/apache2/ssl
```

5. Edit the virtual host configuration.

> **Note**
> The virtual host configuration should be in the file you backed up initially.

Edit the virtual host configuration to match:

```
<VirtualHost *:80>
	ServerName myswarm
	ServerAlias myswarm.host
	ErrorLog "/path/to/apache/logs/myswarm.error_log"
	CustomLog "/path/to/apache/logs/myswarm.access_log" common
	DocumentRoot "/path/to/swarm/public"
	<Directory "/path/to/swarm/public">
		AllowOverride All
		Require all granted
	</Directory>

	Redirect / https://myswarm.host/
</VirtualHost>

<VirtualHost *:443>
	SSLEngine on
	SSLCertificateFile /etc/apache2/ssl/apache.crt
	SSLCertificateKeyFile /etc/apache2/ssl/apache.key

	ServerName myswarm.host
	ServerAlias myswarm
	ErrorLog "/path/to/apache/logs/myswarm.error_log"
	CustomLog "/path/to/apache/logs/myswarm.access_log" common
	DocumentRoot "/path/to/swarm/public"
	<Directory "/path/to/swarm/public">
		AllowOverride All
		Require all granted
	</Directory>
</VirtualHost>
```

See Apache's virtual host documentation for details: https://httpd.apache.org/docs/2.4/vhosts/

6. Customize the virtual host definition.

a. Replace ***myswarm.host*** with the hostname for Swarm on your network.

b. Replace ***myswarm*** with the name of the subdomain hosting Swarm. Many administrators choose swarm.

> **Note**
> The string **myswarm** in the log file paths: this should match the subdomain name and prefix for the log files, to help coordinate the active host with the log files for that host. Doing this is particularly useful when your Apache server hosts multiple instances of Swarm.

c. Replace ***/path/to/apache/logs*** with the path where your Apache store its log files. Apache's log files are typically named **access_log** and **error_log**.

d. Replace ***/path/to/swarm*** with the path to the Swarm directory.

7. Restart your web server.

```
$ sudo apachectl restart
```

8. Adjust your firewall configuration to allow connections to the standard SSL port for web servers.
   - For CentOS/RHEL:

```
$ sudo firewall-cmd --zone=public --add-port=443/tcp --
permanent
$ sudo systemctl reload firewalld
```

   - For other distributions, consult with your network administrator or operating system documentation to determine how to adjust your firewall configuration.

9. Test your HTTPS URL from a web browser.

> **Important**
> If the ***myswarm.host*** value in the virtual host configuration and the certificate do not match, the P4V integration with Swarm fails with the message **SSL handshake failed**.
>
> Also, when a reverse DNS lookup is performed, **myswarm.host** should be the answer when querying for the Swarm server's IP address.

10. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.

Run Swarm in a sub-folder of an existing web site

> **Warning**
> **Multiple-Helix server instances on a single Swarm instance:** You cannot run Swarm in a sub-folder if Swarm is connected to multiple-Helix servers.

Issue: Swarm will lose connection to all of the Helix servers if you edit the base_url configurable value in the **environment** block of **<swarm_root>/data/config.php**. This will stop your system working.

Fix: Remove the **base_url** configurable from the **environment** block of **<swarm_root>/data/config.php**.

Important
Back up your Swarm virtual host configuration before customizing your Swarm installation, see "Back up your Swarm virtual host first" on page 178.

If you cannot run Swarm in its own virtual host, which might be necessary when you do not control the hostname to be used with Swarm, installing Swarm in a sub-folder of an existing virtual host configuration can be a good solution.

Installing Swarm in a sub-folder requires modification of the previous installation steps covered in this chapter:

- The "Apache configuration" on page 148 is entirely different; instead of establishing a new virtual host, you need to modify an existing virtual host configuration. Often, this would be Apache's default site.
- Swarm's configuration file requires an extra item.

The following sections cover the specifics of sub-folder installation.

See "base_url" on page 497 for more details.

Important
If you used the Swarm OVA or Swarm packages to install Swarm, you can adjust Swarm's configuration using the package configuration script **/opt/perforce/swarm/sbin/configure-swarm.sh**.

configure-swarm.sh does not read any existing Swarm configuration; you must provide all of the configuration details each time you execute **configure-swarm.sh**:

```
$ sudo /opt/perforce/swarm/sbin/configure-swarm.sh -n -p myp4host:1666
-u swarm -w password -e mx.example.com -H myhost -B /swarm
```

In the example above, the **-B** option is used to specify the name of the sub-folder.

If you use **configure-swarm.sh** to adjust the Swarm configuration, you only need to follow the "Apache configuration" on the facing page steps described below; all of the changes listed in the "Swarm configuration" on page 186 section below have been completed by **configure-swarm.sh**.

Apache configuration

1. Ensure that the ***SWARM_ROOT*** is not within the document root of the intended virtual host.

   This step ensures that Swarm's source code and configuration is impossible to browse, preventing access to important details such as stored credentials, and active sessions and workspaces.

2. Adjust the virtual host configuration that you are already using.

   > **Note**
   > Depending on the method used to install Swarm, the filename for virtual host configuration you need to edit is:
   >
   > - For Swarm OVA or Swarm package installations, edit **`perforce-swarm-site.conf`**.
   > - For manual installations following Swarm's recommended "Apache configuration" on page 148 edit **`swarm`**.
   > - For other installations, you may have to edit **`httpd.conf`** or nearby files.

   Add the following lines to the virtual host definition:

   ```
   Alias "/swarm" "SWARM_ROOT/public"

   DocumentRoot "/var/www/html"

   <Directory "SWARM_ROOT/public">
     AllowOverride All
     Require all granted
   </Directory>
   ```

   The **`Alias`** line configures Apache to respond to requests to **`https://myhost/swarm`** with content from Swarm's **`public`** folder. You can change the **`/swarm`** portion of the Alias line to anything you want.

   The **`DocumentRoot`** line configures the Apache DocumentRoot directory location.

   The **`<Directory>`** block grants access to everything within Swarm's **`public`** folder. Replace **`SWARM_ROOT`** with the actual path to Swarm.

3. Restart your web server.

   ```
   $ sudo apachectl restart
   ```

Swarm configuration

To successfully operate within a sub-folder, the "swarm_root" on page 580**/data/config.php** file needs to be adjusted to contain the following lines (as a peer of the p4 item):

```
'environment' => array(
        'base_url' => '/swarm'
 ),
```

Ensure that **/swarm** matches the first item in the **Alias** line in the virtual host configuration.

See "Environment" on page 495 for more details.

Swarm trigger configuration

The **swarm-trigger.conf** file must be updated to include the **base_url** in the **SWARM_HOST** path.

> **Tip**
> The location of the **swarm-trigger.conf** depends on your installation. For information on the location of the **swarm-trigger.conf** file, see "Set up Swarm triggers with a Windows or Linux-hosted Helix server" on page 159.

1. Edit the **swarm-trigger.conf** file.
2. Replace:

   SWARM_HOST="http://my-swarm-host"

   With:

   SWARM_HOST="http://website-hostname/swarm"
3. Save the file.

Cron configuration

Swarm's recurring task configuration must be updated to reflect the sub-folder that you have configured in Apache's and Swarm's configurations. This is configured in the **swarm-cron-hosts.conf** file.

1. Edit **/opt/perforce/etc/swarm-cron-hosts.conf**.
2. Replace:

   ```
   https://swarm-host
   ```

   with:

   ```
   https://swarm-host/swarm/
   ```

Where ***swarm-host*** is the hostname of your Swarm installation, and ***swarm*** is the sub-folder you want to use.

3. Save the edited file.

   New workers should be started at the start of the next minute.

4. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.

Run the Swarm virtual host on a custom port

Important
Back up your Swarm virtual host configuration before customizing your Swarm installation, see "Back up your Swarm virtual host first" on page 178.

If you cannot run Swarm on port 80 (or port 443 for HTTPS), perhaps because you do not have root access, it is possible to run Swarm on a custom port.

Installing Swarm to use a custom port requires modification of the previous installation steps covered in this chapter: The Apache configuration is slightly different, requiring modification of Swarm's virtual host definition.

The following section covers the specifics of the custom port configuration.

Note
In addition to the following instructions, you may also need to apply the **external_url** item described in the "Environment" on page 495 section if your Swarm is behind a proxy, or you have multiple Swarm instances connected to Helix server.

Important
If you used the Swarm OVA or Swarm packages to install Swarm, you can adjust Swarm's configuration using the package configuration script **/opt/perforce/swarm/sbin/configure-swarm.sh**.

configure-swarm.sh does not read any existing Swarm configuration; you must provide all of the configuration details each time you execute **configure-swarm.sh**:

```
$ sudo /opt/perforce/swarm/sbin/configure-swarm.sh -n -p
myp4host:1666 -u swarm -w password -e mx.example.com -H myhost
-P 8080
```

In the example above, the **-P** option is used to specify the custom port that Swarm should use.

If you use **configure-swarm.sh** to adjust Swarm's configuration, follow the additional steps that it describes. Once those steps are complete, do not perform any of the steps described below.

Apache configuration

1. Edit the virtual host configuration.

   > **Note**
   > Depending on the method used to install Swarm, the filename for virtual host configuration you need to edit is:
   >
   > - For Swarm OVA or Swarm package installations, edit **`perforce-swarm-site.conf`**.
   > - For manual installations following Swarm's recommended Apache configuration, edit **`swarm`**.
   > - For other installations, you may have to edit **`httpd.conf`** or nearby files.

   a. Add the following line *outside* of the **`<VirtualHost>`** block:

   ```
   Listen 8080
   ```

   b. Edit the **`<VirtualHost *:80>`** line to read:

   ```
   <VirtualHost *:8080>
   ```

   For both lines, replace ***`8080`*** with the custom port you wish to use.

   > **Important**
   > If you choose a port that is already in use, Apache refuses to start.

2. Restart your web server.

   ```
   $ sudo apachectl restart
   ```

3. Adjust your firewall configuration to allow connections to the custom port.
   - For CentOS/RHEL:

     ```
     $ sudo firewall-cmd --zone=public --add-port=8080/tcp --permanent
     $ sudo systemctl reload firewalld
     ```

     Replace ***`8080`*** with the custom port you wish to use.
   - For other distributions, consult with your network administrator or operating system documentation to determine how to just your firewall configuration.

Cron configuration

Swarm's recurring task configuration must be updated to reflect the custom port that you have configured in Apache's configuration. This is configured in the **`swarm-cron-hosts.conf`** file.

1. Edit **/opt/perforce/etc/swarm-cron-hosts.conf**.
2. Replace:

   ```
   https://swarm-host:80
   ```

   with:

   ```
   https://swarm-host:8080
   ```

   Where ***swarm-host*** is the hostname of your Swarm installation, and ***8080*** is the custom port you want to use.
3. Save the edited file.

   New workers should be started at the start of the next minute.
4. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.

Use your own Redis server

By default Swarm uses its own Redis server on the Swarm machine.

This section describes how to connect Swarm to your own Redis server.

To use your own Redis server:

1. Add the **redis** block to the *SWARM_ROOT*/**data/config.php** file, it contains the **password**, **namespace**, **host**, and **port** details of your Redis server

```
<?php
    // this block should be a peer of 'p4'
    'redis' => array(
        'options' => array(
            'password' => null, // Defaults to null
            'namespace' => 'Swarm',
            'server' => array(
                'host' => 'MyRedisServerHostname', // Defaults to
'localhost' or enter your Redis server hostname
                'port' => 'MyRedisServerPortNumber', // Defaults
to '7379 or enter your Redis server port
            ),
        ),
        'items_batch_size' => 100000,
        'check_integrity' => '03:00', // Defaults to '03:00' Use one
of the following two formats:
```

```
                                        // 1) The time of day that the
integrity check starts each day. Set in 24 hour format with leading
zeros and a : separator
                                        // 2) The number of seconds
between each integrity check. Set as a positive integer. Specify '0'
to disable the integrity check.
        'population_lock_timeout' => 300, // Timeout for initial
cache population. Defaults to 300 seconds.
    ),
```

- **`password`**: set to the password of your Redis server, defaults to null
- **`namespace`**: the prefix used for key values in the Redis cache. Defaults to **`Swarm`**.

  > **Note**
  > If you have multiple-Swarm instances running against a single Redis server, each Swarm server must use a different Redis **`namespace`**. This enables the cache data for the individual Swarm instances to be identified. The **`namespace`** is limited to ≤ 128 characters.
  >
  > If one or more of your Swarm instances is connected to multiple-Helix servers, the Redis **`namespace`** includes the **`server label`** and the character limit is reduced to ≤ 127 characters, see "Multiple-Helix server instances" on page 519.

- **`host`** ***`MyRedisServerHostname`***: Redis server host name
- **`port`** ***`MyRedisServerPortNumber`***: Redis server port number
- **`items_batch_size`**: Maximum number of key/value pairs allowed in an **`mset`** call to Redis. Sets exceeding this will be batched according to this maximum for efficiency. Defaults to **`100000`**.

  > **Note**
  > The default value of **`100000`** was chosen to strike a balance between efficiency and project data complexity. This value should not normally need to be changed, contact support before making a change to this value.

- **`check_integrity`**: In some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server. Swarm can run a regular integrity check to make sure that the Redis caches and Helix server are in sync. If an integrity check finds an out of sync cache file, Swarm automatically updates the data in that cache.

The **check_integrity** configurable specifies when the Redis cache integrity check is run. Set as a specific time of day (24 hour format with leading zeros) or a number of seconds (positive integer) between checks. Disable the integrity check with **'0'**. Defaults to **'03:00'**.

- **population_lock_timeout**: specifies the timeout, in seconds, for initial cache population. If you have a large Swarm system, increase this time if the initial cache population times out. Defaults to **300** seconds.

2. Edit the Redis server configuration in the **/opt/perforce/etc/redis-server.conf** file:

```
bind MyRedisServerHostname
port MyRedisServerPortNumber
supervised auto
save ""
dir /MyRedis/CacheDatabase/Directory/Location
```

   - **bind *MyRedisServerHostname*** - Redis server host name
   - **port *MyRedisServerPortNumber*** - Redis server port number
   - **supervised auto** - detects the use of **upstart** or **systemd** automatically to signal that the process is ready to use the supervisors
   - **save ""** - background saves disabled, recommended.
   - **dir */MyRedis/CacheDatabase/Directory/Location*** - the directory the Redis cache database is stored in

> **Tip**
> - The **redis-server.conf** file contains more detailed information about the Redis configuration for Swarm.
> - On Swarm systems with a large number of users, groups, and projects, start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves, see the **redis-server.conf** file comments for detailed information.

3. Swarm is now connected to your Redis server.
4. Your Swarm installation is complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on the next page.

Validate your Swarm installation

> **Tip**
> When Swarm starts it verifies the Redis cache, during this time you cannot log in to Swarm. The time taken to verify the Redis cache depends on the number of users, groups, and projects Swarm has. Start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves in the `redis-server.conf` file, see "Redis server configuration file" on page 542.

Now that Swarm is fully installed the final step is to check that your Swarm installation is working correctly by doing the following:

1. Create a new changelist that:
    a. Contains at least one modified file
    b. Contains the `#review` keyword in the changelist description
2. Right click on the new changelist in P4V and click **Shelve Files...**

    > **Important**
    > Do not select **Request New Swarm Review...** because this method uses the API and will not fully test the Swarm triggers.

3. Check that a new Swarm review is created for the changelist.
    - If a Swarm review is created, the Swarm triggers are working.
    - If a Swarm review is not created, see below.

You are now all set to start using Swarm, enjoy!

> **Tip**
> To get started with Swarm, see the "Quickstart" on page 29 chapter.

Review not created

If a new Swarm review is not created when you validate your installation, your Swarm triggers are probably not installed correctly on the Helix server. For instructions on how to install the Swarm triggers on your Helix server, see "Helix Core server configuration for Swarm" on page 158.

Review cannot be updated

This is rare but can sometimes happen if your Swarm server is configured for SSL.

Issue

The **swarm-trigger.pl** tries to use the Perl **HTTP::Tiny** module if it is installed. Sometimes it has a problem with certificates and a 599 error is generated.

To check if this is the issue, examine the operating system log of your Helix server for errors logged by **swarm-trigger.pl**.

- Windows platforms, look in the Windows Event viewer
- Linux platforms:
  - **Ubuntu/Debian:** look in **/var/log/syslog**
  - **CentOS/RedHat:** look in **/var/log/messages**

The error message looks something like: **Error: (599/Internal Exception) (probably invalid SSL certificate) .....**

Workaround

If you see this error, modify the Swarm trigger script file on the Helix server so that it uses **Curl** instead of **HTTP::Tiny**.

Modify the trigger script to use Curl:

There are two lines in the **swarm-trigger.pl** script that enable the use of **HTTP::Tiny**, edit these two lines to make the script fallback to **Curl**:

1. Find both instances of the following line:

   ```
   if ($HAVE_TINY) {
   ```

2. Add **!** to **$HAVE_TINY** for each line so that they look like this:

   ```
   if (!$HAVE_TINY) {
   ```

3. Save the trigger script file.

Upgrade Swarm

The section describes how to upgrade a Swarm package, OVA, or tarball install to a newer release.

> **Tip**
> If you are not already running Swarm, none of these instructions apply to you. Instead, see the Swarm installation instructions.

Swarm upgrade process:

- If you installed Swarm via a package, see "Upgrade a Swarm package installation" on the next page for details.

- If you installed Swarm via an OVA, see "Upgrade a Swarm package installation" on the next page for details.
- If you installed Swarm via a source distribution archive (tar file), see "Upgrade a Swarm tarball installation" on page 208 for details.

Upgrade a Swarm package installation

Important

- Swarm runtime dependencies change between releases, you must check that your system satisfies the Swarm runtime dependencies before starting the upgrade, see "Runtime dependencies" on page 101.
- Review the PHP requirements before you upgrade Swarm, see "PHP" on page 105.
- Review the Helix server requirements before you upgrade Swarm, see "Helix Core server requirements" on page 107.
- Helix server 2020.1 and later, permissions have changed for viewing and editing stream spec files in Swarm. To view and edit stream spec files in Swarm, the Swarm user must have *admin* permissions for the entire depot `//...`

Note

If you are upgrading from Swarm 2017.2 or earlier, run the Swarm index upgrade after you have validated your upgrade. This is the last step of the upgrade and ensures that the review activity history is displayed in the correct order on the Dashboard, and Reviews list pages.

Before you begin your Swarm upgrade

The Swarm Workflow feature was introduced in Swarm 2018.2 and was disabled by default, this feature is now enabled by default for Swarm 2019.2 and later.

If you are not currently using the Swarm workflow feature and you are using the `strict` and `enforce` triggers to control commits you have the following options:

- **Use the Swarm workflow feature:** you must comment out your `strict` and `enforce` triggers and use the new workflow triggers.

> **Note**
> **Known limitations**
>
> The workflow triggers do not support the following trigger functionality available in Swarm 2018.1 and earlier:
>
> - `EXEMPT_FILE_COUNT`
> - `EXEMPT_EXTENSIONS`
>
> To continue to use this trigger functionality, you must keep your existing `enforce` and `strict` triggers and disable the Workflow feature as shown below.

- **Continue to use the `strict` and `enforce` triggers:** keep your existing `enforce` and `strict` triggers and Disable the Workflow feature. Support for these triggers will be dropped in a later release.

  > **Tip**
  > If you disable the workflow feature in the Swarm `config.php` file, workflow will not be processed by Swarm but a small overhead is still incurred by the Helix server each time it runs a workflow trigger script. This overhead can be eliminated by commenting out the `swarm.enforce change-submit`, `swarm.strict change-content`, and `swarm.shelvesub shelve-submit` workflow triggers.

Decide whether you want to use the workflow feature before you start your upgrade because this will determine which triggers you need to use. The trigger requirements are described in more detail in the **Update your triggers** stage of the Swarm upgrade.

Upgrade Swarm

> **Important**
> For the Swarm 2015.2 release, the packages have been renamed. The following instructions upgrade your Swarm packages to the latest versions.

The following process attempts to minimize downtime, but a short period of downtime for Swarm users is unavoidable. There should be no downtime for your Helix server. After a successful upgrade, all Swarm users are logged out.

If you are using Swarm in a production environment, we encourage you to test this upgrade process in a non-production environment first.

1. Follow the instructions for your OS distribution:

   Ubuntu:

   ```
   sudo apt-get update
   sudo apt-get install helix-swarm helix-swarm-triggers helix-swarm-
   optional
   ```

CentOS

Follow the instructions for your CentOS distribution version:

CentOS 7.8 (run these commands as root):

Important

- As part of the PHP upgrade process your Apache 2.2 server will be stopped and disabled, if you are currently using the Apache 2.2 server for any other applications they will stop working. You will either need to upgrade these applications to work with PHP 7 and Apache 2.4 or move them to another server.
- **Swarm 2019.1 to 2020.1:** these versions of Swarm used PHP packages from the SCL repositories for CentOS/RHEL 7. This was to provide access to more recent versions of PHP than are available as standard on CentOS/RHEL. This required the use of the **`httpd24-httpd`** package for Apache. These were all installed into **`/opt/rh`**

  Swarm 2020.2: this version of Swarm uses the Remi repository for CentOS/RHEL 7 and 8. This provides PHP 7.4 installed in the standard file system structure. This means that the old **`httpd24-httpd`** version of Apache is no longer needed, and the standard system version of Apache is being used again.

  The SCL Apache site configuration file was installed at this location for Swarm 2019.1 to 2020.1:

  `/opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf`

  If this exists when Swarm is upgraded to 2020.2, this file is copied to **`/etc/httpd/conf.d/perforce-swarm-site.conf`** if there is no file at the destination. It is also re-written to change references from **`/var/log/httpd24`** to **`/var/log/httpd`**

  If a site configuration file for Swarm already exists in **`/etc/httpd`**, the copy and re-write is not performed.

  After upgrade, **`httpd24-httpd`** is disabled.
- To avoid seeing the Apache HTTP server Linux test page when you start the Apache server, comment out the content of the **`welcome.conf`** file located in the **`/etc/httpd/conf.d/`** directory.

The full PHP and Apache upgrade process described below is only required the first time you upgrade to PHP 7.x. For future Swarm upgrades just run **step e** to upgrade Swarm, **step l** if you want to install ImageMagick, and **step n** to check that SELinux is working correctly.

The upgrade process will deploy **`epel-release-latest-7.noarch.rpm`** to give Swarm access to the EPEL repository, deploy **`remi-release-7.rpm`** to give Swarm access to PHP 7.x and Apache 2.4, upgrade Swarm, stop and disable the Apache 2.2 server, copy and edit some Swarm files so they work with Apache 2.4, and finally start and enable the Apache 2.4 server.

a. Deploy the **`epel-release-latest-7.noarch.rpm`** repository configuration package to give Swarm access to EPEL packages:

   `yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm`

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

   `yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm`

   > **Tip**
   > If you don't deploy the Remi repository you will see an error similar to the one shown below when you do the next steps:

   ```
   --> Finished Dependency Resolution
   Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
              Requires: httpd24
   Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
              Requires: rh-php70-php-xml
   Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
              Requires: rh-php70-php-mbstring
   Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
              Requires: rh-php70-php-opcache
   Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
              Requires: rh-php70-php
   Error: Package: helix-swarm-2019.1-1761134.el6.x86_64 (perforce)
              Requires: rh-php70
    You could try using --skip-broken to work around the problem
    You could try running: rpm -Va --nofiles --nodigest
   ```

c. Install the **`yum-utils`** package to give access to the **`yum-config-manager`** command:

   `yum install yum-utils`

d. Install the **Default/Single version** of PHP:

   i. Disable **`remi-php*`**:

      `yum-config-manager --disable 'remi-php*'`

   ii. Enable PHP 7.4:

```
yum-config-manager --enable remi-php74
```

iii. Run an upgrade for PHP, this will also upgrade the Swarm packages:

```
yum update
```

e. If you didn't run the `yum update` in the previous step, run the Swarm package upgrade now:

```
yum install helix-swarm helix-swarm-triggers helix-swarm-optional
```

> **Important**
> If you are upgrading from Swarm 2019.3 to Swarm 2020.2, remove the Swarm PHP 7.1 files:
>
> ```
> yum remove $(yum list installed | grep php71 | awk '{print $1}')
> ```

f. If you are upgrading from Apache 2.2, disable your Apache 2.2 HTTP server so that is does not automatically start:

```
systemctl disable httpd
```

g. If you are upgrading from Apache 2.2, stop your Apache 2.2 HTTP server:

```
systemctl stop httpd
```

h. If you have any special `php.ini` configuration options set, copy the `php.ini` file to the `/etc/php.d/php.ini/` directory, for example:

```
cp /etc/opt/rh/rh-php72/php.d/php.ini /etc/php.d/php.ini
```

i. Copy the `perforce-swarm-site.conf` file to the `/etc/httpd/conf.d/` directory if it is not already in there, for example:

```
$ cp /opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf /etc/httpd/conf.d/perforce-swarm-site.conf
```

j. Replace the `/log/httpd24` filepath with `/log/httpd/` in the `perforce-swarm-site.conf` file using the `sed` command:

```
$ sed -i "s#/log/httpd24/#/log/httpd/#g" /etc/httpd/conf.d/perforce-swarm-site.conf
```

k. Enable your Apache 2.4 HTTP server so that it automatically starts:

```
systemctl enable httpd
```

l. Start the Apache 2.4 HTTP server:

```
systemctl start httpd
```

m. **Optional, ImageMagick:** when ImageMagick is enabled, Swarm can display the following image formats: BMP, EPS, PSD, TGA, TIFF.

   i. Install ImageMagick:

   ```
   yum install ImageMagick
   ```

   ii. Restart the web server to so that ImageMagick is picked up.

   ```
   systemctl restart httpd
   ```

n. If you are upgrading from Swarm 2020.1, restart your Redis server:

   ```
   systemctl restart redis-server-swarm
   ```

o. Check that SELinux is working correctly for your system. For instructions on configuring SELinux on CentOS, see "SELinux on CentOS/RHEL configuration" on page 134.

RHEL

Follow the instructions for your RHEL distribution version:

RHEL 7.8 (run these commands as root):

Important

- As part of the PHP upgrade process your Apache 2.2 server will be stopped and disabled, if you are currently using the Apache 2.2 server for any other applications they will stop working. You will either need to upgrade these applications to work with PHP 7 and Apache 2.4 or move them to another server.
- **Swarm 2019.1 to 2020.1:** these versions of Swarm used PHP packages from the SCL repositories for CentOS/RHEL 7. This was to provide access to more recent versions of PHP than are available as standard on CentOS/RHEL. This required the use of the **`httpd24-httpd`** package for Apache. These were all installed into **`/opt/rh`**

  Swarm 2020.2: this version of Swarm uses the Remi repository for CentOS/RHEL 7 and 8. This provides PHP 7.4 installed in the standard file system structure. This means that the old **`httpd24-httpd`** version of Apache is no longer needed, and the standard system version of Apache is being used again.

  The SCL Apache site configuration file was installed at this location for Swarm 2019.1 to 2020.1:

  `/opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf`

  If this exists when Swarm is upgraded to 2020.2, this file is copied to **`/etc/httpd/conf.d/perforce-swarm-site.conf`** if there is no file at the destination. It is also re-written to change references from **`/var/log/httpd24`** to **`/var/log/httpd`**

If a site configuration file for Swarm already exists in **/etc/httpd**, the copy and re-write is not performed.

After upgrade, **httpd24-httpd** is disabled.

- To avoid seeing the Apache HTTP server Linux test page when you start the Apache server, comment out the content of the **welcome.conf** file located in the **/etc/httpd/conf.d/** directory.

The full upgrade process described below is only required the first time you upgrade to PHP 7.x. For future Swarm upgrades just run **step f** to upgrade Swarm, **step n** if you want to install ImageMagick, and **step p** to check that SELinux is working correctly.

The upgrade process will deploy **epel-release-latest-7.noarch.rpm** to give Swarm access to the EPEL repository, deploy **remi-release-7.rpm** to give Swarm access to PHP 7.x and Apache 2.4, upgrade Swarm, stop and disable the Apache 2.2 server, copy and edit some Swarm files so they work with Apache 2.4, and finally start and enable the Apache 2.4 server.

a. Deploy the **epel-release-latest-7.noarch.rpm** repository configuration package to give Swarm access to EPEL packages:

   yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

   yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm

   Tip
   If you don't enable the Remi repository you will see an error similar to the one shown below when you do the next steps:

   ```
   Error: Package: helix-swarm-2020.2-1845646.el6.x86_64
   (perforce)
           Requires: rh-php74
   ```

c. Install the **yum-utils** package to give access to the **yum-config-manager** command:

   yum install yum-utils

d. Enable the optional channel for some dependencies:

   subscription-manager repos --enable=rhel-7-server-optional-rpms

e. Install the **Default/Single version** of PHP:

   i. Disable `remi-php*`:

   ```
   yum-config-manager --disable 'remi-php*'
   ```

   ii. Enable PHP 7.4:

   ```
   yum-config-manager --enable remi-php74
   ```

   iii. Run an upgrade for PHP, this will also upgrade the Swarm packages:

   ```
   yum update
   ```

f. If you didn't run the `yum update` in the previous step, run the Swarm package upgrade now:

   ```
   yum install helix-swarm helix-swarm-triggers helix-swarm-optional
   ```

   > **Important**
   > If you are upgrading from Swarm 2019.3 to Swarm 2020.2, remove the Swarm PHP 7.1 files:
   >
   > ```
   > yum remove $(yum list installed | grep php71 | awk '{print $1}')
   > ```

g. If you are upgrading from Apache 2.2, disable your Apache 2.2 HTTP server so that is does not automatically start:

   ```
   systemctl disable httpd
   ```

h. If you are upgrading from Apache 2.2, stop your Apache 2.2 HTTP server:

   ```
   systemctl stop httpd
   ```

i. If you have any special `php.ini` configuration options set, copy the `php.ini` file to the `/etc/php.d/` directory:

   ```
   cp /etc/opt/rh/rh-php72/php.d/php.ini /etc/php.d/php.ini
   ```

j. Copy the `perforce-swarm-site.conf` file to the `/etc/httpd/conf.d/` directory if it is not already in there, for example:

   ```
   $ cp /opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf /etc/httpd/conf.d/perforce-swarm-site.conf
   ```

k. Replace the `/log/httpd24` filepath with `/log/httpd/` in the `perforce-swarm-site.conf` file using the `sed` command:

   ```
   $ sed -i "s#/log/httpd24/#/log/httpd/#g" /etc/httpd/conf.d/perforce-swarm-site.conf
   ```

l. Enable your Apache 2.4 HTTP server so that it automatically starts:

```
systemctl enable httpd
```

m. Start the Apache 2.4 HTTP server:

```
systemctl start httpd
```

n. **Optional, ImageMagick:** when ImageMagick is enabled, Swarm can display the following image formats: BMP, EPS, PSD, TGA, TIFF.

i. Install ImageMagick:

```
yum install ImageMagick
```

ii. Restart the web server to so that ImageMagick is picked up.

```
systemctl restart httpd
```

o. If you are upgrading from Swarm 2020.1, restart your Redis server:

```
systemctl restart redis-server-swarm
```

p. Check that SELinux is working correctly for your system. For instructions on configuring SELinux on RHEL, see "SELinux on CentOS/RHEL configuration" on page 134.

2. Swarm generally has several major updates each year, and may occasionally have a patch update between major updates. To determine whether a Swarm update is available, follow the instructions for your OS distribution:

Ubuntu:

```
sudo apt-get update
sudo apt-get -s upgrade | grep swarm
```

CentOS/RHEL (run this command as root):

```
yum list updates | grep swarm
```

> **Tip**
> Swarm uses a Redis server to manage its caches. This is installed and configured on the Swarm machine during the upgrade. If you prefer to use your own Redis server, see "Use your own Redis server" on page 189.

Update your triggers

1. Copy the new Swarm trigger script to your Helix Core server machine. The trigger script is **`SWARM_ROOT/p4-bin/scripts/swarm-trigger.pl`**, and requires installation of Perl 5.08+ (use the latest available) on the Helix server machine. If Swarm is using SSL, then the triggers also require the **`IO::Socket::SSL`** Perl module.

> **Warning**
> Do not overwrite any existing trigger script at this time. Give the script a new name, for example: **`swarm-trigger-new.pl`**.

2. Configure the Swarm trigger script by creating, in the same directory on the Helix server machine, **`swarm-trigger.conf`**. It should contain:

> **Note**
> If you already have a **`swarm-trigger.conf`** file, no additional configuration is required.

```
# SWARM_HOST (required)
# Hostname of your Swarm instance, with leading "http://" or
"https://".
SWARM_HOST="http://my-swarm-host"

# SWARM_TOKEN (required)
# The token used when talking to Swarm to offer some security. To
obtain the
# value, log in to Swarm as a super user and select 'About Swarm' to
see the
# token value.
SWARM_TOKEN="MY-UUID-STYLE-TOKEN"

# ADMIN_USER (optional) Do not use if the Workflow feature is
enabled (default)
# For enforcing reviewed changes, optionally specify the normal
Perforce user
# with admin privileges (to read keys); if not set, will use whatever
Perforce
# user is set in environment.
ADMIN_USER=

# ADMIN_TICKET_FILE (optional) Do not use if the Workflow
feature is enabled (default)
# For enforcing reviewed changes, optionally specify the location of
the
```

```
# p4tickets file if different from the default ($HOME/.p4tickets).
# Ensure this user is a member of a group with an 'unlimited' or very
long
# timeout; then, manually login as this user from the Perforce server
machine to
# set the ticket.
ADMIN_TICKET_FILE=

# VERIFY_SSL (optional)
# If HTTPS is being used on the Swarm web server, then this controls
whether
# the SSL certificate is validated or not. By default this is set to
1, which
# means any SSL certificates must be valid. If the web server is
using a self
# signed certificate, then this must be set to 0.
# set the ticket.
VERIFY_SSL=1
```

Fill in the required **SWARM_HOST** and **SWARM_TOKEN** variables with the configuration from any previous Swarm trigger script, typically **swarm-trigger.pl**.

> **Tip**
> The **ADMIN_USER** and **ADMIN_TICKET** variables were used by the **'enforce triggers'** in Swarm 2019.1 and earlier. They can be removed unless you are explicitly disabling workflow and using the deprecated **'enforce triggers'**.

> **Note**
> **Swarm 2015.4 and earlier:** Swarm trigger script files were available as shell scripts in these earlier Swarm versions, typically **swarm-trigger.sh**.
>
> Swarm must now use a Perl trigger script file, typically **swarm-trigger.pl**.

3. **On Linux:** ensure that the script is executable:

   ```
   $ sudo chmod +x swarm-trigger-new.pl
   ```

4. Rename the new trigger script:

   On Linux:

   ```
   $ mv swarm-trigger-new.pl swarm-trigger.pl
   ```

On Windows:

```
C:\> ren swarm-trigger-new.pl swarm-trigger.pl
```

5. Update the triggers in your Helix server.

> **Warning**
>
> - The **`swarm.shelvedel shelve-delete`** trigger line was added to Swarm in version 2018.1 and updated in version 2020.1.
>   - **Upgrading from Swarm 2017.4 and earlier:** add the **`swarm.shelvedel shelve-delete`** trigger line to the Helix server trigger table if it is not already present, see " Update the Helix server triggers table to run the trigger script." on page 164
>   - **Upgrading from Swarm 2018.x and 2019.x:** replace the existing **`swarm.shelvedel shelve-delete`** trigger line in the Helix server trigger table with the one supplied in the Swarm version you are upgrading to.
> - **Workflow feature:**
>
>   The Workflow feature is enabled by default in Swarm 2019.2 and later. The trigger lines required when workflow is enabled are different to those required when workflow is disabled:
>   - **Workflow feature enabled (default):**
>   - Comment out the **`swarm.enforce.1`**, **`swarm.enforce.2`**, **`swarm.strict.1`**, and **`swarm.strict.2`** trigger lines in the Helix server trigger table if they are present, see " Update the Helix server triggers table to run the trigger script." on page 164
>   - Add the **`swarm.enforce change-submit`**, **`swarm.strict change-content`**, and **`swarm.shelvesub shelve-submit`** trigger lines to the Helix server trigger table if they are not already present, see " Update the Helix server triggers table to run the trigger script." on page 164
>   - **Workflow feature disabled:**
>   Comment out the **`swarm.enforce change-submit`**, **`swarm.strict change-content`**, and **`swarm.shelvesub shelve-submit`** trigger lines in the Helix server trigger table if they are present, see " Update the Helix server triggers table to run the trigger script." on page 164

a. Run the Swarm trigger script to capture (using **Ctrl+C** on Windows and Linux, **Command+C** on Mac OSX) the trigger lines that should be included in the Perforce trigger table:

On Linux:

```
$ ./swarm-trigger.pl -o
```

On Windows:

```
C:\> path/to/perl swarm-trigger.pl -o
```

b. As a Perforce user with *super* privileges, update the Perforce trigger table by running **`p4 triggers`** command and replacing any **`swarm.*`** lines with the previously captured trigger line output (using **Ctrl+V** on Windows and Linux, **Command+V** on Mac OSX).

> **Important**
> If you previously customized the Swarm trigger lines, perhaps to apply various "Trigger options" on page 589, be sure to repeat those customizations within the updated trigger lines.

Validate your upgrade

> **Tip**
> When Swarm starts it verifies the Redis cache, during this time you cannot log in to Swarm. The time taken to verify the Redis cache depends on the number of users, groups, and projects Swarm has. Start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves in the **`redis-server.conf`** file, see "Redis server configuration file" on page 542.

Check that your upgraded Swarm instance is working correctly by doing the following:

1. Create a new changelist that:
   a. Contains at least one modified file
   b. Contains the **`#review`** keyword in the changelist description
2. Right click on the new changelist in P4V and click **Shelve Files...**

   > **Important**
   > Do not select **Request New Swarm Review...** because this method uses the API and will not fully test the Swarm triggers.

3. Check that a new Swarm review is created for the changelist.
   - If a Swarm review is created, the Swarm triggers are working.
   - If a Swarm review is not created, see below.

> **Note**
> - If a new Swarm review is not created when you validate your upgrade, you may be missing some Swarm triggers on the Helix server. Periodically new triggers are added to Swarm and these need to be installed when you upgrade, see the **Update your triggers** section above. For more information about Swarm trigger configuration on your Helix server, see "Helix Core server configuration for Swarm" on page 158.

Swarm index upgrade

If you are upgrading from Swarm 2017.2 or earlier you should run the index upgrade, this ensures that the review activity history is displayed in the correct order on the Dashboard, and Reviews list pages.

> **Note**
> If you are upgrading from Swarm version 2017.3 or later, the index upgrade step is not required.

The index upgrade process can be configured to suit your Swarm system specifications. See Upgrade index for details.

Run the upgrade as an *Admin* user by visiting the following URL:

```
http://SWARM-HOST/upgrade
```

All done!

Upgrade a Swarm tarball installation

> **Warning**
> - Swarm runtime dependencies change between releases, you must check that your system satisfies the Swarm runtime dependencies before starting the upgrade, see "Runtime dependencies" on page 101.
> - P4PHP should be upgraded to the version included in the new Swarm release.
>   - If you have already configured PHP to use the Swarm-provided P4PHP (as recommended), this happens automatically.
>   - If you have manually installed P4PHP in some other fashion, configure Swarm to use the version of P4PHP included in the new Swarm tarball before you perform any of the upgrade steps below. See "PHP configuration" on page 151 for details.
>
>   **Swarm package, OVA and tarball installations:** 2 versions of P4PHP are supplied for each PHP 7 version supported by Swarm. They are located in the **`p4-bin/bin.linux26x86_64`** directory.
>   - **`perforce-php7x.so`** compatible with systems using SSL 1.0.2
>   - **`perforce-php7x-ssl1.1.1.so`** compatible with systems using SSL 1.1.1 (by default Ubuntu 18.04 and Ubuntu 20.04 use SSL 1.1.1)
>
>   Where **`x`** is the version of PHP 7.
>
>   If the **`perforce.ini`** file is not pointing at the correct version of P4PHP and you connect to an SSL enabled Helix server:
>   - The Swarm web-page will not load and you might see a **`Connection Reset`** error.
>   - There might be an **`undefined symbol: SSLeay`** message in the Apache error log

- Review the PHP requirements before you upgrade Swarm, see "PHP" on page 105.
- Review the Helix server requirements before you upgrade Swarm, see "Helix Core server requirements" on page 107.
- Helix server 2020.1 and later, permissions have changed for viewing and editing stream spec files in Swarm. To view and edit stream spec files in Swarm, the Swarm user must have *admin* permissions for the entire depot `//...`

Note
OVA upgrade process:

- **Recommended:** use the package upgrade process, see "Upgrade a Swarm package installation" on page 194 for details.
- **Not recommended:** use the tarball upgrade instructions in this section.

Note
If you are upgrading from Swarm 2017.2 or earlier, run the Swarm index upgrade after you have validated your upgrade. This is the last step of the upgrade and ensures that the review activity history is displayed in the correct order on the Dashboard, and Reviews list pages.

Before you begin your Swarm upgrade

The Swarm Workflow feature was introduced in Swarm 2018.2 and was disabled by default, this feature is now enabled by default for Swarm 2019.2 and later.

If you are not currently using the Swarm workflow feature and you are using the `strict` and `enforce` triggers to control commits you have the following options:

- **Use the Swarm workflow feature:** you must comment out your `strict` and `enforce` triggers and use the new workflow triggers.

  Note
  Known limitations

  The workflow triggers do not support the following trigger functionality available in Swarm 2018.1 and earlier:

  - `EXEMPT_FILE_COUNT`
  - `EXEMPT_EXTENSIONS`

  To continue to use this trigger functionality, you must keep your existing `enforce` and `strict` triggers and disable the Workflow feature as shown below.

- **Continue to use the `strict` and `enforce` triggers:** keep your existing `enforce` and `strict` triggers and Disable the Workflow feature. Support for these triggers will be dropped in a later release.

  > **Tip**
  > If you disable the workflow feature in the Swarm `config.php` file, workflow will not be processed by Swarm but a small overhead is still incurred by the Helix server each time it runs a workflow trigger script. This overhead can be eliminated by commenting out the `swarm.enforce change-submit`, `swarm.strict change-content`, and `swarm.shelvesub shelve-submit` workflow triggers.

Decide whether you want to use the workflow feature before you start your upgrade because this will determine which triggers you need to use. The trigger requirements are described in more detail in the **Update your triggers** stage of the Swarm upgrade.

Upgrade Swarm

The following process attempts to minimize downtime, but a short period of downtime for Swarm users is unavoidable. There should be no downtime for your Helix server. After a successful upgrade, all Swarm users are logged out.

If you are using Swarm in a production environment, we encourage you to test this upgrade process in a non-production environment first.

CentOS/RHEL: PHP 7.x and Apache 2.4 installation

CentOS and RHEL do not have PHP 7.x and Apache 2.4 by default so you must upgrade your system before you can upgrade Swarm. This process is only required the first time you upgrade to PHP 7.x.

Carry out the following steps:

> **Important**
> - As part of the PHP upgrade process your Apache 2.2 server will be stopped and disabled, if you are currently using the Apache 2.2 server for any other applications they will stop working. You will either need to upgrade these applications to work with PHP 7 and Apache 2.4 or move them to another server.
> - **CentOS 7:** Use the Remi repository configuration package (`remi-release-7.rpm`) to give Swarm access to PHP 7.x. The `sclo-php7x-php-pecl-redis` and `sclo-php7x-php-pecl-igbinary` extensions are not available from this RedHat repository and you must source them from elsewhere.
> - **RHEL 7:** Use the Remi repository configuration package (`remi-release-7.rpm`) to give Swarm access to PHP 7.x and most of the required PHP extensions. The `php7x-php-pecl-redis` and `php7x-php-pecl-igbinary` extensions are not available from this RedHat repository and you must source them from elsewhere.

- **Swarm 2019.1 to 2020.1:** these versions of Swarm used PHP packages from the SCL repositories for CentOS/RHEL 7. This was to provide access to more recent versions of PHP than are available as standard on CentOS/RHEL. This required the use of the **`httpd24-httpd`** package for Apache. These were all installed into **`/opt/rh`**

  Swarm 2020.2: this version of Swarm uses the Remi repository for CentOS/RHEL 7 and 8. This provides PHP 7.4 installed in the standard file system structure. This means that the old **`httpd24-httpd`** version of Apache is no longer needed, and the standard system version of Apache is being used again.

  The SCL Apache site configuration file was installed at this location for Swarm 2019.1 to 2020.1:

  `/opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf`

  If this exists when Swarm is upgraded to 2020.2, this file is copied to **`/etc/httpd/conf.d/perforce-swarm-site.conf`** if there is no file at the destination. It is also re-written to change references from **`/var/log/httpd24`** to **`/var/log/httpd`**

  If a site configuration file for Swarm already exists in **`/etc/httpd`**, the copy and re-write is not performed.

  After upgrade, **`httpd24-httpd`** is disabled.
- To avoid seeing the Apache HTTP server Linux test page when you start the Apache server, comment out the content of the **`welcome.conf`** file located in the **`/etc/httpd/conf.d/`** directory.

This section describes how to install PHP 7.x and Apache 2.4 on your system using a repository .

1. This process is only required the first time you upgrade to PHP 7.x. Install PHP 7.x and migrate the Swarm Apache configuration to use Apache 2.4, follow the instructions for your OS distribution:

   CentOS 7.8 (run these commands as root):

   a. Deploy the **`epel-release-latest-7.noarch.rpm`** repository configuration package to give Swarm access to EPEL packages:

      `yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm`

   b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

      `yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm`

   c. Install the **`yum-utils`** package to give access to the **`yum-config-manager`** command:

```
yum install yum-utils
```

d. Install the **Default/Single version** of PHP:

   i. Disable `remi-php*`:

   ```
   yum-config-manager --disable 'remi-php*'
   ```

   ii. Enable PHP 7.4:

   ```
   yum-config-manager --enable remi-php74
   ```

   iii. Run an upgrade for PHP:

   ```
   yum update
   ```

> **Important**
> If you are upgrading from Swarm 2019.3 to Swarm 2020.2, remove the Swarm PHP 7.1 files:
>
> ```
> $ yum remove $(yum list installed | grep php71 | awk '{print $1}')
> ```

e. If you are upgrading from Apache 2.2, disable your Apache 2.2 HTTP server so that is does not automatically start:

```
$ systemctl disable httpd
```

f. If you are upgrading from Apache 2.2, stop your Apache 2.2 HTTP server:

```
$ systemctl stop httpd
```

g. If you are upgrading from PHP 7.2, copy the `30-perforce.ini` file to `/etc/php.d/` directory, this step also changes the `ini` filename to `perforce.ini`:

```
$ cp /etc/opt/rh/rh-php72/php.d/30-perforce.ini /etc/php.d/perforce.ini
```

h. If you are upgrading from PHP 7.2, replace the PHP 7.2 references with PHP 7.4 in the `perforce.ini` file using the `sed` command:

```
$ sed -i "s/72/74/g" /etc/php.d/perforce.ini
```

i. If you have any special `php.ini` configuration options set, copy the `php.ini` file to the `/etc/php.d/` directory, for example:

```
$ cp /etc/opt/rh/rh-php72/php.d/php.ini /etc/php.d/php.ini
```

j. Copy the `perforce-swarm-site.conf` file to the `/etc/httpd/conf.d/` directory if it is not already in there, for example:

```
$ cp /opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf /etc/httpd/conf.d/perforce-swarm-site.conf
```

k. Replace the `/log/httpd24` filepath with `/log/httpd/` in the `perforce-swarm-site.conf` file using the `sed` command:

```
$ sed -i "s#/log/httpd24/#/log/httpd/#g"
/etc/httpd/conf.d/perforce-swarm-site.conf
```

l. Enable your Apache 2.4 HTTP server so that it automatically starts:

```
$ systemctl enable httpd
```

m. Start the Apache 2.4 HTTP server:

```
$ systemctl start httpd
```

n. Check that SELinux is working correctly for your system. For instructions on configuring SELinux on CentOS, see "SELinux on CentOS/RHEL configuration" on page 134.

RHEL 7.8 (run these commands as root):

a. Deploy the `epel-release-latest-7.noarch.rpm` repository configuration package to give Swarm access to EPEL packages:

```
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

b. Deploy the Remi repository configuration package to give Swarm access to PHP 7.x (only required the first time you upgrade to PHP 7.x):

```
yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

c. Install the `yum-utils` package to give access to the `yum-config-manager` command:

```
yum install yum-utils
```

d. Install the **Default/Single version** of PHP:

   i. Disable `remi-php*`:

   ```
   yum-config-manager --disable 'remi-php*'
   ```

   ii. Enable PHP 7.4:

   ```
   yum-config-manager --enable remi-php74
   ```

   iii. Run an upgrade for PHP:

   ```
   yum update
   ```

> **Important**
> If you are upgrading from Swarm 2019.3 to Swarm 2020.2, remove the Swarm PHP 7.1 files:
>
> ```
> $ yum remove $(yum list installed | grep php71 | awk '{print $1}')
> ```

e. If you are upgrading from Apache 2.2, disable your Apache 2.2 HTTP server so that is does not automatically start:

```
$ systemctl disable httpd
```

f. If you are upgrading from Apache 2.2, stop your Apache 2.2 HTTP server:

```
$ systemctl stop httpd
```

g. If you are upgrading from PHP 7.2, copy the `30-perforce.ini` file to `/etc/php.d/perforce.ini/` directory, this step also changes the `ini` filename to `perforce.ini`:

```
$ cp /etc/opt/rh/rh-php72/php.d/30-perforce.ini /etc/php.d/perforce.ini
```

h. If you are upgrading from PHP 7.2, replace the PHP 7.2 references with PHP 7.4 in the `perforce.ini` file using the `sed` command:

```
$ sed -i "s/72/74/g" /etc/php.d/perforce.ini
```

i. If you have any special `php.ini` configuration options set, copy the `php.ini` file to the `/etc/php.d/` directory, for example:

```
$ cp /etc/opt/rh/rh-php72/php.d/php.ini /etc/php.d/php.ini
```

j. Copy the `perforce-swarm-site.conf` file to the `/etc/httpd/conf.d/` directory if it is not already in there, for example:

```
$ cp /opt/rh/httpd24/root/etc/httpd/conf.d/perforce-swarm-site.conf /etc/httpd/conf.d/perforce-swarm-site.conf
```

k. Replace the `/log/httpd24` filepath with `/log/httpd/` in the `perforce-swarm-site.conf` file using the `sed` command:

```
$ sed -i "s#/log/httpd24/#/log/httpd/#g" /etc/httpd/conf.d/perforce-swarm-site.conf
```

l. Enable your Apache 2.4 HTTP server so that it automatically starts:

```
$ systemctl enable httpd
```

m. Start the Apache 2.4 HTTP server:

```
$ systemctl start httpd
```

n. Check that SELinux is working correctly for your system. For instructions on configuring SELinux on RHEL 7, see "SELinux on CentOS/RHEL configuration" on page 134.

2. PHP 7.x and Apache 2.4 are now installed, configure the Swarm Redis service on the Swarm machine, see "Configure Redis on the Swarm machine" below.

Configure Redis on the Swarm machine

Follow the instructions for the Swarm version you are upgrading from:

Upgrade from Swarm 2019.1 and earlier:

Swarm requires Redis to manage its caches and by default Swarm uses its own Redis server on the Swarm machine. Swarm caches data from the Helix server to improve the performance of common searches in Swarm and to reduce the load on the Helix server. Redis is included in the Swarm Tarball installation.

This section describes how to configure the Swarm Redis service on the Swarm machine. Do not change the default values of the following configuration files if you are using the Swarm Redis server.

> **Tip**
> If required, you can use your own Redis server instead of the Swarm Redis server. For instructions on how to configure Swarm to use your own Redis server, see "Use your own Redis server" on page 189.

Swarm has two Redis binaries in ***SWARM_ROOT*/p4-bin/bin.linux26x86_64/**:

- **redis-server-swarm**
- **redis-cli-swarm**

1. Configure the Redis cache database, enter the following details in the **redis-server.conf** file in **/opt/perforce/etc/**:

```
bind 127.0.0.1
port 7379
supervised auto
save ""
dir /opt/perforce/swarm/redis
```

Default values:

> **Tip**
> - The default settings are shown below, the **redis-server.conf** file contains more detailed information about the Redis configuration for Swarm.
> - On Swarm systems with a large number of users, groups, and projects, start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves, see the **redis-server.conf** file comments for detailed information.

- **bind 127.0.0.1** - Redis server IP address (loopback interface)
- **port 7379** - Redis server port number
- **supervised auto** - detects the use of **upstart** or **systemd** automatically to signal that the process is ready to use the supervisors
- **save ""** - background saves disabled, recommended.
- **dir /opt/perforce/swarm/redis** - the directory the Redis cache database is stored in.

> **Tip**
> To fine-tune your Redis server settings, make your changes in the Laminas Redis adapter. For information about the Laminas Redis adapter, see the Laminas Redis Adapter documentation.

2. The Redis cache database must be running before Swarm starts so we recommend setting it up as a service so that it starts automatically at boot time.

   The following example script will:

   - run the Redis service as the Perforce user
   - use the configuration file **in /opt/perforce/etc/redis-server.conf**

- assume the database server is installed in **/opt/perforce/swarm/p4-bin/bin.linux26x86_64/**

To configure Redis as a service, add a script with the following content in **/etc/systemd/system**

redis-server-swarm.service

```
[Unit]
Description=Redis Server for Swarm
After=network.target

[Service]
ExecStart=/opt/perforce/swarm/p4-bin/bin.linux26x86_64/redis-server-swarm /opt/perforce/etc/redis-server.conf
ExecStop=/opt/perforce/swarm/p4-bin/bin.linux26x86_64/redis-cli-swarm shutdown
Restart=always
RestartSec=10
StandardOutput=syslog
StandardError=syslog
SyslogIndentifier=redis
User=perforce
       Group=perforce

[Install]
WantedBy=multi-user.target
```

3. Create the *SWARM_ROOT*/**data/config.php** file if it does not already exist. The **redis** block of the *SWARM_ROOT*/**data/config.php** file contains the **password**, **namespace**, **host** and **port** details of the Swarm Redis server:

```
<?php
    // this block should be a peer of 'p4'
    'redis' => array(
        'options' => array(
            'password' => null, // Defaults to null
            'namespace' => 'Swarm',
            'server' => array(
```

```
                'host' => 'localhost', // Defaults to 'localhost' or
enter your Redis server hostname
                'port' => '7379', // Defaults to '7379 or enter your
Redis server port
            ),
        ),
        'items_batch_size' => 100000,
        'check_integrity' => '03:00', // Defaults to '03:00' Use one
of the following two formats:
                                      // 1) The time of day that the
integrity check starts each day. Set in 24 hour format with leading
zeros and a : separator
                                      // 2) The number of seconds
between each integrity check. Set as a positive integer. Specify '0'
to disable the integrity check.
        'population_lock_timeout' => 300, // Timeout for initial
cache population. Defaults to 300 seconds.
    ),
```

Configurables:

- **`password`**: Redis server password. Defaults to **`null`** and should be left at default if using the Swarm Redis server.
- **`namespace`**: the prefix used for key values in the Redis cache. Defaults to **`Swarm`** and should be left at default if using the Swarm Redis server.

  > **Note**
  > If you have multiple-Swarm instances running against a single Redis server, each Swarm server must use a different Redis **`namespace`**. This enables the cache data for the individual Swarm instances to be identified. The **`namespace`** is limited to ≤ 128 characters.
  >
  > If one or more of your Swarm instances is connected to multiple-Helix servers, the Redis **`namespace`** includes the **`server label`** and the character limit is reduced to ≤ 127 characters, see "Multiple-Helix server instances" on page 519.

- **`host`**: Redis server hostname. Defaults to **`localhost`** and should be left at default if using the Swarm Redis server.
- **`port`**: Redis server port number. Defaults to **`7379`**. Swarm uses port **`7379`** as its default to avoid clashing with other Redis servers that might be on your network. It should be left at

default if using the Swarm Redis server. The default port for a non-Swarm Redis server is `6379`.

- `items_batch_size`: Maximum number of key/value pairs allowed in an `mset` call to Redis. Sets exceeding this will be batched according to this maximum for efficiency. Defaults to `100000`.

  > **Note**
  > The default value of `100000` was chosen to strike a balance between efficiency and project data complexity. This value should not normally need to be changed, contact support before making a change to this value.

- `check_integrity`: In some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server. Swarm can run a regular integrity check to make sure that the Redis caches and Helix server are in sync. If an integrity check finds an out of sync cache file, Swarm automatically updates the data in that cache.

  The `check_integrity` configurable specifies when the Redis cache integrity check is run. Set as a specific time of day (24 hour format with leading zeros) or a number of seconds (positive integer) between checks. Disable the integrity check with `'0'`. Defaults to `'03:00'`.

- `population_lock_timeout`: specifies the timeout, in seconds, for initial cache population. If you have a large Swarm system, increase this time if the initial cache population times out. Defaults to `300` seconds.

Upgrade from Swarm 2019.2 and later:

Swarm requires Redis to manage its caches and by default Swarm uses its own Redis server on the Swarm machine. Swarm caches data from the Helix server to improve the performance of common searches in Swarm and to reduce the load on the Helix server. Redis is included in the Swarm Tarball installation.

> **Note**
> If you are using your own Redis server instead of the Swarm Redis server, upgrade your Redis server to the version supplied with your Swarm upgrade.

Swarm has two Redis binaries in ***`SWARM_ROOT`*`/p4-bin/bin.linux26x86_64/`**:

- **`redis-server-swarm`**
- **`redis-cli-swarm`**

If the Redis binaries supplied with your new version of Swarm have been updated, your Redis server is automatically upgraded to the new Redis server version. Your original Redis configuration files will not be changed. Do not change the default values of the Redis configuration files if you are using the Swarm Redis server.

Restart the Redis server:

If the Redis binaries have been updated and you are running Redis as a service (recommended), you must restart the Redis server to use the new Redis server version.

1. Run the following command to restart the Redis server:

   ```
   systemctl restart redis-server-swarm
   ```

2. You are now running the updated Redis server version.

> **Tip**
> On Swarm systems with a large number of users, groups, and projects, start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves, see the **`redis-server.conf`** file comments for detailed information.

Now that the Swarm Redis service is configured for the Swarm machine, start your Swarm upgrade, see "Upgrade Swarm" below.

Upgrade Swarm

The steps in this section describe how to upgrade Swarm using the provided archive file. **`SWARM_ROOT`** refers to the current Swarm installation.

> **Tip**
> For OVA installations, **`SWARM_ROOT`** is **`/opt/perforce/swarm`**.

1. Download the new TAR file from https://www.perforce.com/downloads/helix-swarm.
2. Expand the new **`swarm.tgz`**:

   ```
   $ tar -zxf swarm.tgz
   ```

   The contents of **`swarm.tgz`** are expanded into a top-level folder named **`swarm-`*`version`***, where ***`version`*** corresponds to the version downloaded. This directory is identified as **`SWARM_NEW`** below.

3. Move **`SWARM_NEW`** to be a peer of **`SWARM_ROOT`**:

   ```
   $ mv SWARM_NEW SWARM_ROOT/../
   ```

4. Copy the *`SWARM_ROOT`***`/data/config.php`** file from **`SWARM_ROOT`** to **`SWARM_NEW`**:

   ```
   $ cp -p SWARM_ROOT/data/config.php SWARM_NEW/data/
   ```

5. Create the queue token directory:

   ```
   $  mkdir SWARM_NEW/data/queue
   ```

6. Copy the existing trigger token(s):

```
$ sudo cp -pR SWARM_ROOT/data/queue/tokens SWARM_
NEW/data/queue/
```

7. Assign correct ownership to the new Swarm's data directory:

```
$ sudo chown -R www-data SWARM_NEW/data
```

> **Note**
> The ***www-data*** user above is an example of what the web server user name might be, and can vary based on distribution or customization. For example, the user is typically **apache** for Red Hat/Fedora/CentOS, **www-data** for Debian/Ubuntu, **wwwrun** for SuSE, **_www** for Mac OSX.

Update your triggers

1. Copy the new Swarm trigger script to your Helix Core server machine. The trigger script is **SWARM_ROOT/p4-bin/scripts/swarm-trigger.pl**, and requires installation of Perl 5.08+ (use the latest available) on the Helix server machine. If Swarm is using SSL, then the triggers also require the **IO::Socket::SSL** Perl module.

> **Warning**
> Do not overwrite any existing trigger script at this time. Give the script a new name, for example: **swarm-trigger-new.pl**.

2. Configure the Swarm trigger script by creating, in the same directory on the Helix server machine, **swarm-trigger.conf**. It should contain:

> **Note**
> If you already have a **swarm-trigger.conf** file, no additional configuration is required.

```
# SWARM_HOST (required)
# Hostname of your Swarm instance, with leading "http://" or
"https://".
SWARM_HOST="http://my-swarm-host"

# SWARM_TOKEN (required)
# The token used when talking to Swarm to offer some security. To
obtain the
# value, log in to Swarm as a super user and select 'About Swarm' to
see the
```

```
# token value.
SWARM_TOKEN="MY-UUID-STYLE-TOKEN"

# ADMIN_USER (optional) Do not use if the Workflow feature is
enabled (default)
# For enforcing reviewed changes, optionally specify the normal
Perforce user
# with admin privileges (to read keys); if not set, will use whatever
Perforce
# user is set in environment.
ADMIN_USER=

# ADMIN_TICKET_FILE (optional) Do not use if the Workflow
feature is enabled (default)
# For enforcing reviewed changes, optionally specify the location of
the
# p4tickets file if different from the default ($HOME/.p4tickets).
# Ensure this user is a member of a group with an 'unlimited' or very
long
# timeout; then, manually login as this user from the Perforce server
machine to
# set the ticket.
ADMIN_TICKET_FILE=

# VERIFY_SSL (optional)
# If HTTPS is being used on the Swarm web server, then this controls
whether
# the SSL certificate is validated or not. By default this is set to
1, which
# means any SSL certificates must be valid. If the web server is
using a self
# signed certificate, then this must be set to 0.
# set the ticket.
```

```
VERIFY_SSL=1
```

Fill in the required **SWARM_HOST** and **SWARM_TOKEN** variables with the configuration from any previous Swarm trigger script, typically **swarm-trigger.pl**.

> **Tip**
> The **ADMIN_USER** and **ADMIN_TICKET** variables were used by the **'enforce triggers'** in Swarm 2019.1 and earlier. They can be removed unless you are explicitly disabling workflow and using the deprecated **'enforce triggers'**.

> **Note**
> **Swarm 2015.4 and earlier:** Swarm trigger script files were available as shell scripts in these earlier Swarm versions, typically **swarm-trigger.sh**.
>
> Swarm must now use a Perl trigger script file, typically **swarm-trigger.pl**.

3. **On Linux:** ensure that the script is executable:

   ```
   $ sudo chmod +x swarm-trigger-new.pl
   ```

4. Rename the new trigger script:

   On Linux:

   ```
   $ mv swarm-trigger-new.pl swarm-trigger.pl
   ```

   On Windows:

   ```
   C:\> ren swarm-trigger-new.pl swarm-trigger.pl
   ```

5. Update the triggers in your Helix server.

> **Warning**
> - The **swarm.shelvedel shelve-delete** trigger line was added to Swarm in version 2018.1 and updated in version 2020.1.
>   - **Upgrading from Swarm 2017.4 and earlier:** add the **swarm.shelvedel shelve-delete** trigger line to the Helix server trigger table if it is not already present, see " Update the Helix server triggers table to run the trigger script." on page 164
>   - **Upgrading from Swarm 2018.x and 2019.x:** replace the existing **swarm.shelvedel shelve-delete** trigger line in the Helix server trigger table with the one supplied in the Swarm version you are upgrading to.
> - **Workflow feature:**
>
>   The Workflow feature is enabled by default in Swarm 2019.2 and later. The trigger lines required when workflow is enabled are different to those required when workflow is disabled:

- **Workflow feature enabled (default):**
  - Comment out the `swarm.enforce.1`, `swarm.enforce.2`, `swarm.strict.1`, and `swarm.strict.2` trigger lines in the Helix server trigger table if they are present, see " Update the Helix server triggers table to run the trigger script." on page 164
  - Add the `swarm.enforce change-submit`, `swarm.strict change-content`, and `swarm.shelvesub shelve-submit` trigger lines to the Helix server trigger table if they are not already present, see " Update the Helix server triggers table to run the trigger script." on page 164
- **Workflow feature disabled:**
  Comment out the `swarm.enforce change-submit`, `swarm.strict change-content`, and `swarm.shelvesub shelve-submit` trigger lines in the Helix server trigger table if they are present, see " Update the Helix server triggers table to run the trigger script." on page 164

a. Run the Swarm trigger script to capture (using **Ctrl+C** on Windows and Linux, **Command+C** on Mac OSX) the trigger lines that should be included in the Perforce trigger table:

   On Linux:

   ```
   $ ./swarm-trigger.pl -o
   ```

   On Windows:

   ```
   C:\> path/to/perl swarm-trigger.pl -o
   ```

b. As a Perforce user with *super* privileges, update the Perforce trigger table by running `p4 triggers` command and replacing any `swarm.*` lines with the previously captured trigger line output (using **Ctrl+V** on Windows and Linux, **Command+V** on Mac OSX).

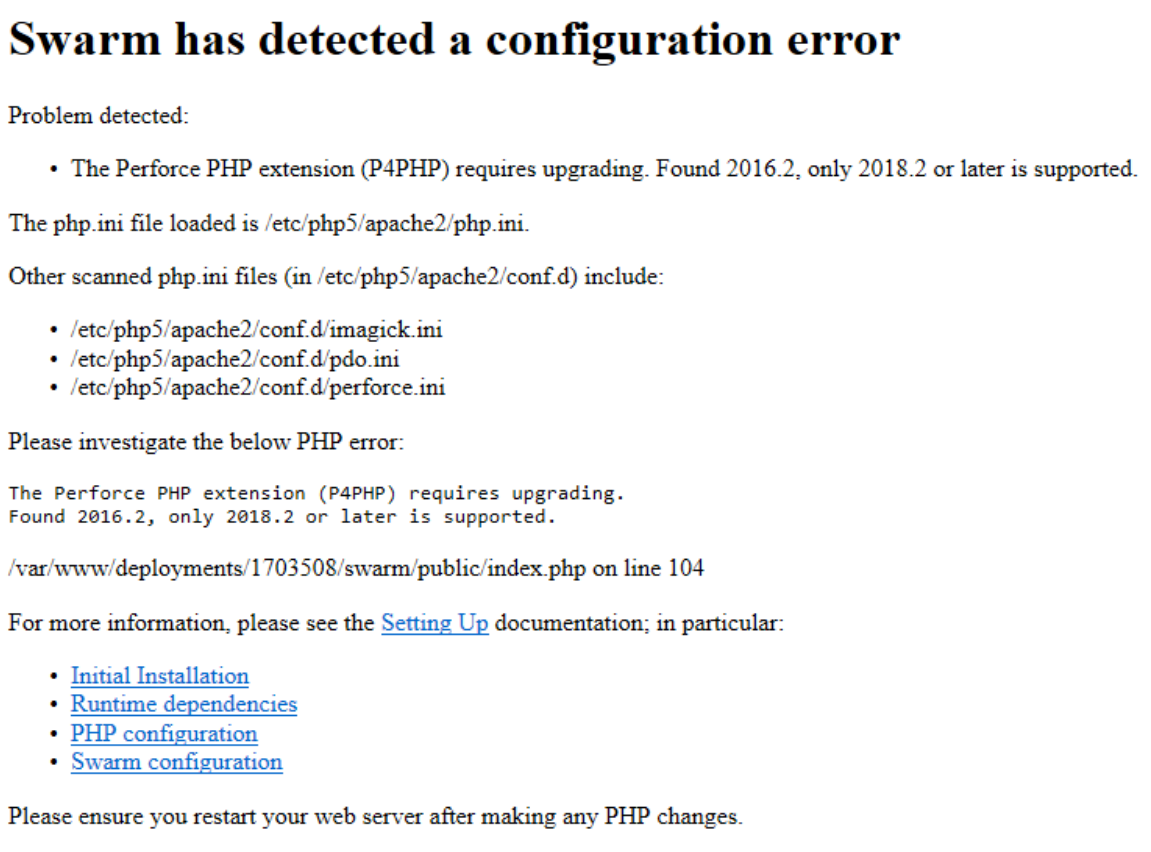
Important
If you previously customized the Swarm trigger lines, perhaps to apply various "Trigger options" on page 589, be sure to repeat those customizations within the updated trigger lines.

Replace your old Swarm instance with your new Swarm instance

Replace your old Swarm with the new Swarm. **Downtime occurs in this step.**

```
$ sudo apache2ctl stop; mv SWARM_ROOT SWARM.old; mv SWARM_NEW SWARM_ROOT; sudo apache2ctl start
```

Note
If you see the following error message when you start Swarm, Swarm is using the wrong version of P4PHP. The latest version of P4PHP is included in the Swarm tarball but you must configure Swarm to use that version of P4PHP. For instructions about how to configure Swarm to use the new version of P4PHP, see "PHP configuration" on page 151.



Swarm has detected a configuration error

Problem detected:

- The Perforce PHP extension (P4PHP) requires upgrading. Found 2016.2, only 2018.2 or later is supported.

The php.ini file loaded is /etc/php5/apache2/php.ini.

Other scanned php.ini files (in /etc/php5/apache2/conf.d) include:

- /etc/php5/apache2/conf.d/imagick.ini
- /etc/php5/apache2/conf.d/pdo.ini
- /etc/php5/apache2/conf.d/perforce.ini

Please investigate the below PHP error:

```
The Perforce PHP extension (P4PHP) requires upgrading.
Found 2016.2, only 2018.2 or later is supported.
```

/var/www/deployments/1703508/swarm/public/index.php on line 104

For more information, please see the Setting Up documentation; in particular:

- Initial Installation
- Runtime dependencies
- PHP configuration
- Swarm configuration

Please ensure you restart your web server after making any PHP changes.

Delete the Swarm config cache

Delete the Swarm config cache to force Swarm to use any new and updated modules in the upgrade. To delete the Swarm config cache, make the following curl request as an *admin* user:

```
curl -u "username:password" -X DELETE
"https://myswarm.url/api/v10/cache/config/"
```

For more information on deleting the config cache, see "Swarm config cache file delete" on page 680.

Validate your upgrade

> **Tip**
> When Swarm starts it verifies the Redis cache, during this time you cannot log in to Swarm. The time taken to verify the Redis cache depends on the number of users, groups, and projects Swarm has. Start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves in the **`redis-server.conf`** file, see "Redis server configuration file" on page 542.

Check that your new Swarm instance is working correctly by doing the following:

1. Create a new changelist that:
   a. Contains at least one modified file
   b. Contains the **`#review`** keyword in the changelist description
2. Right click on the new changelist in P4V and click **Shelve Files...**

   > **Important**
   > Do not select **Request New Swarm Review...** because this method uses the API and will not fully test the Swarm triggers.

3. Check that a new Swarm review is created for the changelist.
   - If a Swarm review is created, the Swarm triggers are working.
   - If a Swarm review is not created, see below.

> **Note**
> If a new Swarm review is not created when you validate your upgrade, you may be missing some Swarm triggers on the Helix server. Periodically new triggers are added to Swarm and these need to be installed when you upgrade, see the **Update your triggers** section above. For more information about Swarm trigger configuration on your Helix server, see "Helix Core server configuration for Swarm" on page 158.

Swarm index upgrade

If you are upgrading from Swarm 2017.2 or earlier you should run the index upgrade, this ensures that the review activity history is displayed in the correct order on the Dashboard, and Reviews list pages.

> **Note**
> If you are upgrading from Swarm version 2017.3 or later, the index upgrade step is not required.

The index upgrade process can be configured to suit your Swarm system specifications. See Upgrade index for details.

Run the upgrade as an *Admin* user by visiting the following URL:

```
http://SWARM-HOST/upgrade
```

All done!

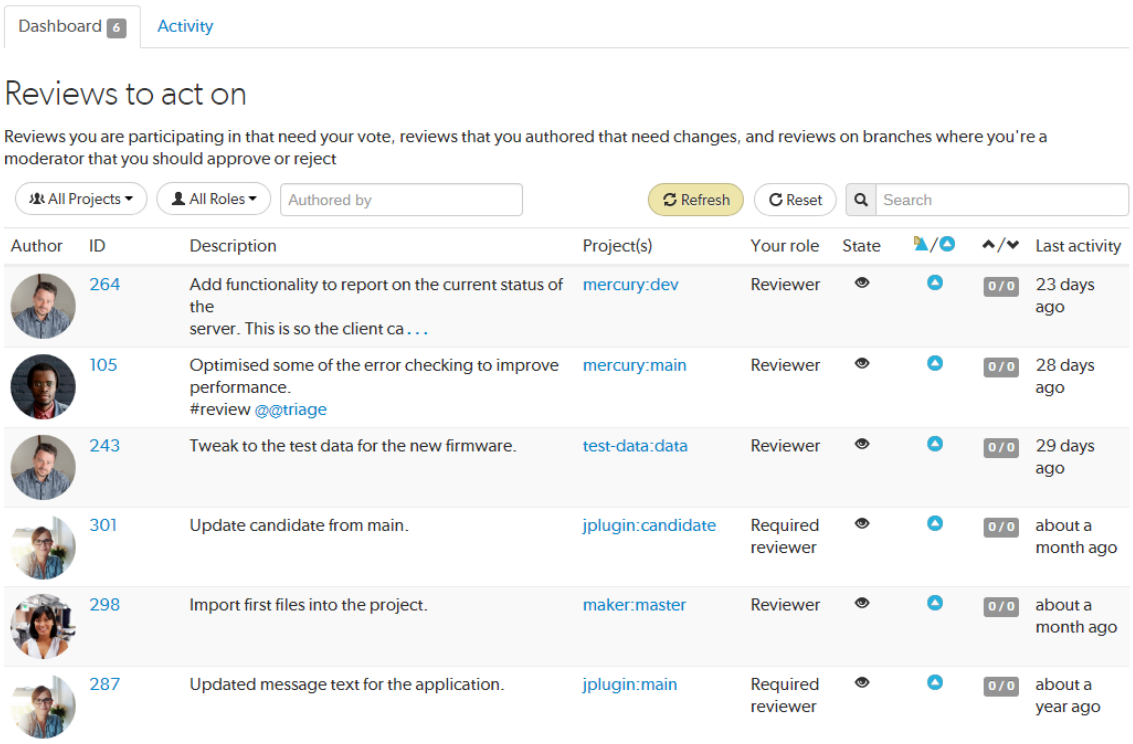
4 | Basics

This chapter covers the basic operations provided by Swarm. These include:

Dashboard

Your dashboard is available by clicking the Swarm  icon above the main menu. Your dashboard displays a list of reviews that you may need to act on.

> **Note**
> Since it is tied to the logged in user, the dashboard is only populated if you are logged in.



The purpose of the dashboard is to allow you to focus on reviews that need to be done, so that other users are not blocked. The dashboard lists reviews according to the most recently modified first, and shows your role in the review.

A review is displayed on your dashboard if any of the following criteria are met:

- You are a reviewer or required reviewer, the review status is **Needs Review** and you have not already voted on it.
- You are a member of a reviewer group or a required reviewer group, the review status is **Needs Review** and you have not already voted on it. The review will remain on your dashboard even if the group has met its criteria if you have not already voted on it.
- You are the review author and the review status is **Needs Revision**, or **Approved** (only if the review is approved but not committed).
- You are a moderator or a member of a moderator group, the review status is **Needs Review**, the **Minimum up votes** requirement for the branch is satisfied, and one of the following is true:
  - There are no required reviewers.
  - All of the required reviewers have up-voted the review.
  - You are the last remaining required reviewer for the review.

> **Tip**
> If moderator behavior is configured to require approval from one moderator per branch and the review spans multiple moderated branches:
>
> - You will see the review in your dashboard if the review has not been approved by another moderator from your branch.
> - You will be able to **Approve and Commit** the review if it has been approved by moderators from all of the other branches.

Filtering

The dashboard can be filtered to display only reviews from a particular project, authored by a particular user, or matching a role. You can click the **Reset** button to reset these filters.

Filtering options are:

- **Projects**

  You can filter by the project the review is part of, limiting results to **My Projects** (projects you are an owner of or a member of) or to an individual project. The **All Projects** drop down will only show projects for which there are reviews in your dashboard.

- **Roles**

  You can filter by your specific role in a review, limiting results to reviews for which you are the author, a reviewer, a required reviewer, or a moderator. The **All Roles** drop down will only show roles for which there are reviews in your dashboard.

- **Authored by**

  You can filter the reviews to only those that have been authored by a certain user. Type in this field to get a drop down list of users to filter by.

- **Reset**

  Clicking the **Reset** button resets all dashboard filters back to their defaults.

- **Search**

  Typing in the search field filters the reviews by their description.

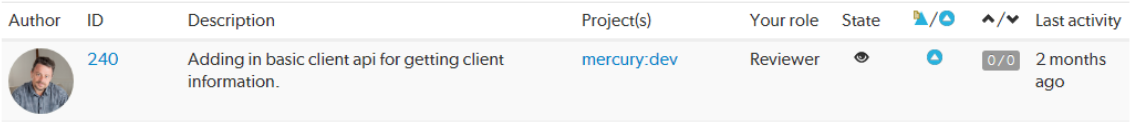
Refresh button

The **Refresh** button is displayed when new content is available for your dashboard. Click **Refresh** to update your dashboard with the new content.

> **Note**
> By default, when your dashboard is open, Swarm checks for new content every five minutes.

Review fields

The dashboard shows a summary of the information for each review.



Reviews that appear here are those which are waiting for action from you. The information presented should help you prioritize what to work on next.

- **Author**

  The author of this review.

- **ID**

  The ID of this review. Click on this to go to the review page.

- **Description**

  The review description. It may be truncated if it is too long, in which case click on the ellipsis **...** to expand it.

- **Project(s)**

  List of project branches this review covers. A review may span multiple branches and projects. Click on one of them to navigate to the project page for that branch.

- **Your role**

  The reason this review is in your dashboard. This can be Author, Reviewer, Required Reviewer, or Moderator.

- **State**

  The current status of the review. This can be Needs Review, Needs Revision, or Approved.

  > **Note**
  > The Approved state only applies to review authors, it is only shown for a review that has been approved but has not been committed.

- **Type**

  The type of review. This can be Pre-commit or Post-commit.

- **Votes**

  The double column of votes displays the number of up votes and down votes for the review.

- **Last activity**

  The last time that any changes were made to the review, including votes, comments, commits, and file changes.

Global Dashboard

> **Note**
> Global Dashboard is only available if your Swarm administrator has configured Swarm to connect to more than one Helix server instance. For instructions on how to configure Swarm to connect to multiple-Helix server instances, see "Multiple-Helix server instances" on page 519.

Your Global Dashboard displays a list of reviews that you may need to act on for each of the Helix server instances connected to Swarm.

Open your global dashboard:

1. Enter the basic Swarm URL without a Helix server instance name, for example: `https://swarm.company.com`
2. If the log in dialog is displayed:

   > **Tip**
   > If Helix Authentication Service is configured for your Helix server and Swarm you will be directed to the sign in process used by your Identity Provider (IdP).

   a. **Server:**

   - If your Helix server instances do not share log in credentials, select an individual Helix server instance from the dropdown list.
   - If all of your Helix server instances share the same log in credentials, leave set to **All available servers**.

     > **Note**
     > If Helix Authentication Service is configured for your Helix server and Swarm, you must log into the servers individually.

   b. Enter your username and password.

   Select the **Remember Me** check box if you prefer to stay logged in between browser restarts.

   > **Note**
   > Helix servers can enforce maximum login times. You may become logged out even if **Remember Me** is selected. Swarm administrators can change the maximum login time, see "Sessions" on page 563 for details.

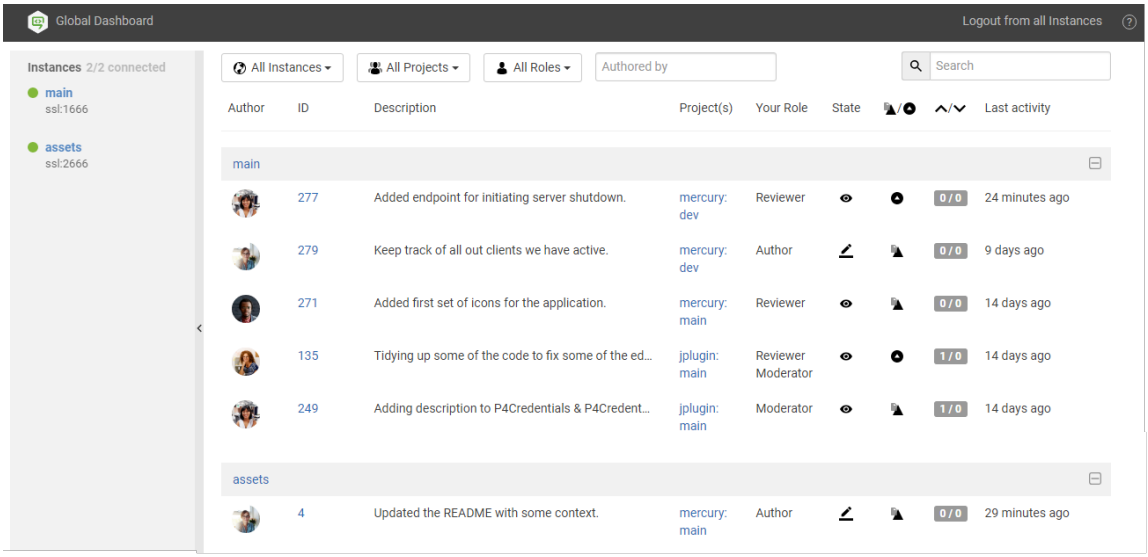
   c. Click **Connect**.

   Swarm will populate the global dashboard from the Helix server instances it connects to.

The global dashboard is displayed:

Tip

- Since it is tied to the logged in user, the global dashboard is only populated for the Helix server instances you are logged in to in Swarm.
- If you are already logged in to a Helix server instance in Swarm, the dashboard for that server will be automatically populated when you open or refresh the global dashboard.



The global dashboard is made up of three main areas:

- "Toolbar" below: log in to Helix server instances, log out of all Helix server instances, and open the Swarm help from the global dashboard toolbar.
- "Sidebar" on page 236: Log in/logout of individual Helix server instances and view you profile for any instance that you are logged in to.
- "Helix server dashboards" on page 238: view, filter and search the dashboards of the Helix server instances you are logged in to. Jump directly to a Helix server, review, or project by clicking on a link.

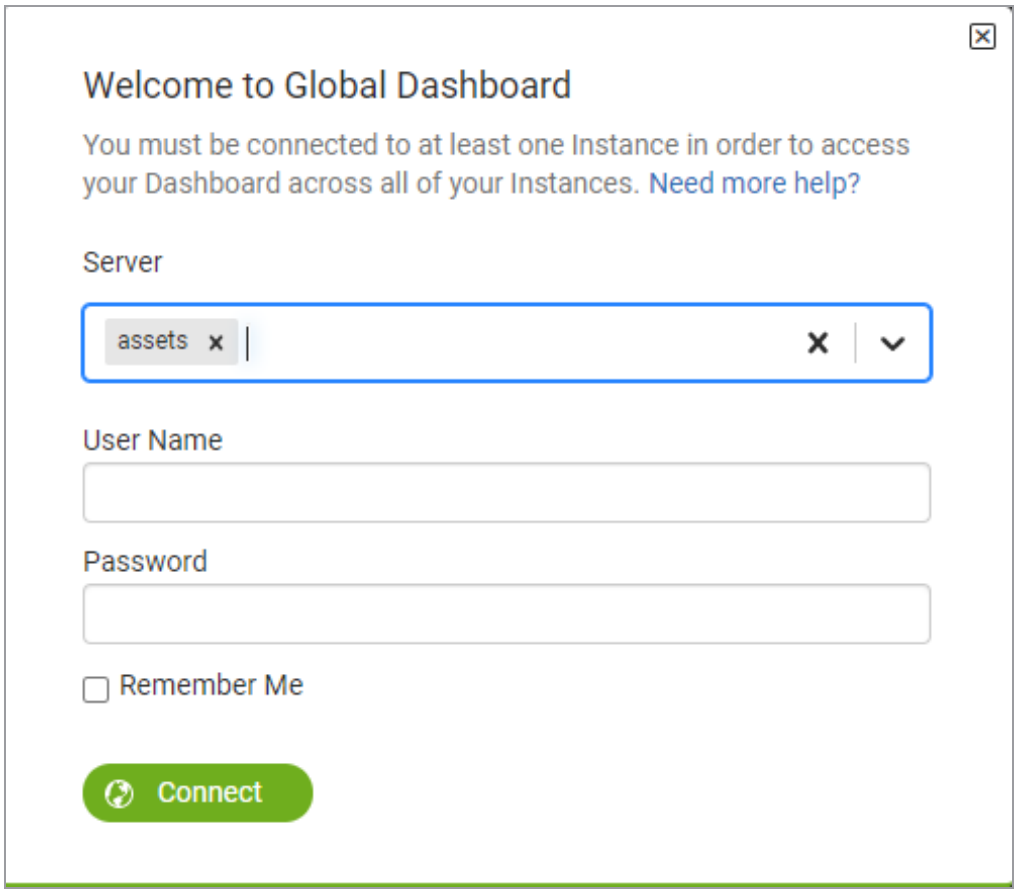
Toolbar

Log in to one or more Helix server instances

Log In is only displayed in the global dashboard toolbar if you are not logged in to any Helix server instances. If **Log In** is not available, see "Log in to a Helix server instance" on page 236.

1. In the global dashboard toolbar, click **Log In**.

> **Tip**
> If Helix Authentication Service is configured for your Helix server and Swarm you will be directed to the sign in process used by your Identity Provider (IdP).



2. **Server:**
   - If all of your Helix server instances share the same log in credentials, leave set to **All available servers**.
   - If your Helix server instances do not share log in credentials, select an individual Helix server instance from the dropdown list.

> **Note**
> If Helix Authentication Service is configured for your Helix server and Swarm, you must log into the servers individually.

> **Tip**
> If you have a number of Helix server instances that use the same login credentials, select them one at a time from the dropdown list.

3. Enter your username and password.
   Select the **Remember Me** check box if you prefer to stay logged in between browser restarts.

   > **Note**
   > Helix servers can enforce maximum login times. You may become logged out even if **Remember Me** is selected. Swarm administrators can change the maximum login time, see "Sessions" on page 563 for details.

4. Click **Connect**.
   Swarm will populate the global dashboard from the Helix server instances it connects to.

Logout of all Helix server instances

Logout from all instances is only displayed in the global dashboard toolbar if you are logged in to at least one Helix server instance.

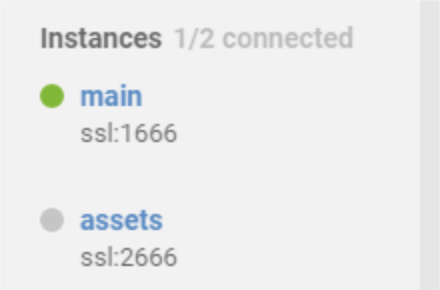
1. In the global dashboard toolbar, click **Logout from all instances**.
2. When prompted, click **Yes** to confirm that you want to logout of all of the Helix server instances.

   > **Tip**
   > - If Helix Authentication Service is configured for your Helix server, logging out of Swarm will not invalidate your Identity Provider (IdP) login status. If you try to log back in to Swarm while your IdP status is still valid, you will not be prompted to complete the log in steps.
   >
   >   If `require_login` is also enabled, Swarm will return you to the login and your IdP will automatically log you back in. In this case log out from your Identity Provider page before logging out from Swarm.
   > - If a custom redirect has been configured by your Swarm administrator, you are logged out of all of the Helix server instances by Swarm and then redirected to the URL specified by the administrator.
   >
   >   The custom redirect can be set to any internal or external URL, for example:

- Company intranet, extranet, internet, FTP, or Web-mail page
- Industry news website
- Identity Provider page to invalidate your IdP log in status

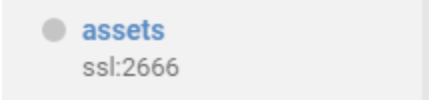
Sidebar

The Helix servers that Swarm can connect to are listed in the collapsible sidebar on the left of the page. From the server list you can log in/logout from individual Helix servers and view your user profile.

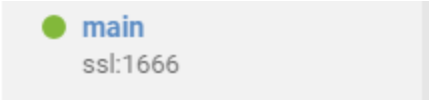


Log in status

- Not logged in to the Helix server instance:



- Logged in to the Helix server instance:

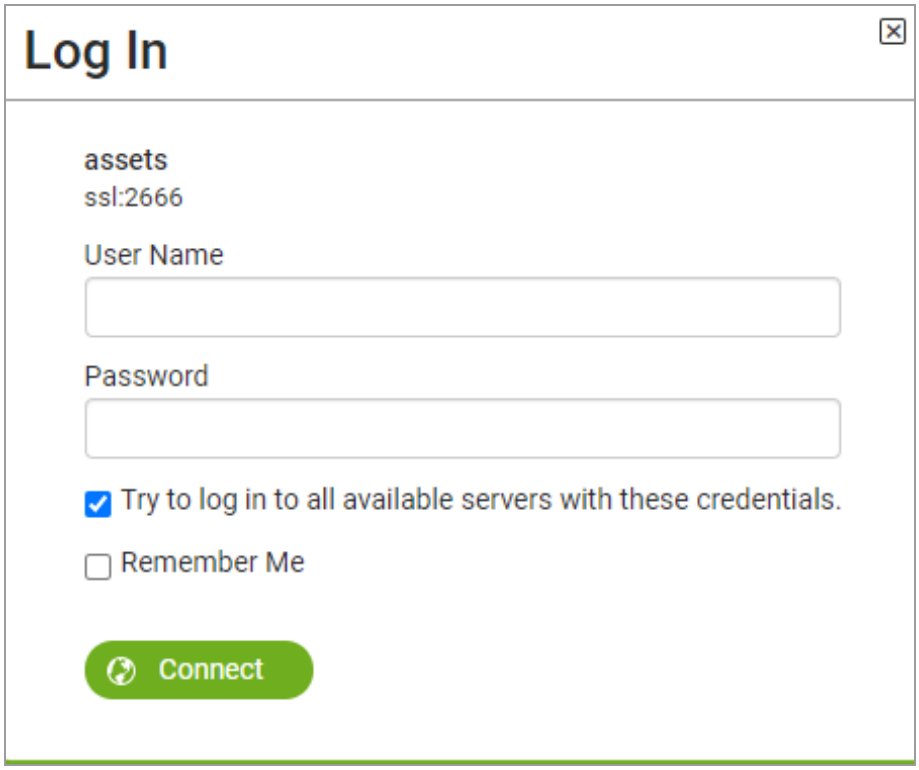


Log in to a Helix server instance

1. In the sidebar, hover over the Helix server you want to connect to.
2. Click the **Log in** button .

Tip

- If Helix Authentication Service is configured for your Helix server and Swarm you will be directed to the sign in process used by your Identity Provider (IdP).
- If you are already logged in to another Helix server that is configured for Helix Authentication Service, your IdP status is valid and you will not be prompted to complete the log in steps.



3. Type in your username and password for the Helix server.
   - Select the **Try to log in to all available servers with these credentials** checkbox if you use these credentials for more than one of the Helix server instances. Swarm will not try to log in to any Helix server instances that are configured for Helix Authentication Service, log in to them individually using the instance **Log in** button in the sidebar.
   - Select the **Remember Me** check box if you prefer to stay logged in between browser restarts.

     Note
     Helix servers can enforce maximum login times. You may become logged out even if **Remember Me** is selected. Swarm administrators can change the maximum login time, see "Sessions" on page 563 for details.

4. Click **Connect**.

   Swarm will populate the global dashboard from the Helix server.

   If you selected the **Try to log in to all available servers with these credentials** checkbox, Swarm will populate the global dashboard for the other servers in the list that it successfully connects to.

Logout of a single Helix server instance

1. In the sidebar, hover over the Helix server you want to log out from.
2. Click the **Gear** button to open the dropdown menu.
3. Click **Logout** in the dropdown menu.

> **Tip**
>
> - If Helix Authentication Service is configured for your Helix server, logging out of Swarm will not invalidate your Identity Provider (IdP) login status. If you try to log back in to Swarm while your IdP status is still valid, you will not be prompted to complete the log in steps.
>
>   If `require_login` is also enabled, Swarm will return you to the login and your IdP will automatically log you back in. In this case log out from your Identity Provider page before logging out from Swarm.
> - If a custom redirect has been configured by your Swarm administrator, you are logged out of the Helix server instance by Swarm. You are only redirected to the URL specified by the administrator if you are not logged in to any other Helix server instances on the global dashboard.
>
>   The custom redirect can be set to any internal or external URL, for example:
>
>   - Company intranet, extranet, internet, FTP, or Web-mail page
>   - Industry news website
>   - Identity Provider page to invalidate your IdP log in status

Helix server dashboards

The purpose of the global dashboard is to allow you to focus on reviews that need to be done, so that other users are not blocked. The global dashboard lists reviews by Helix server according to the most recently modified first, and shows your role in the review.

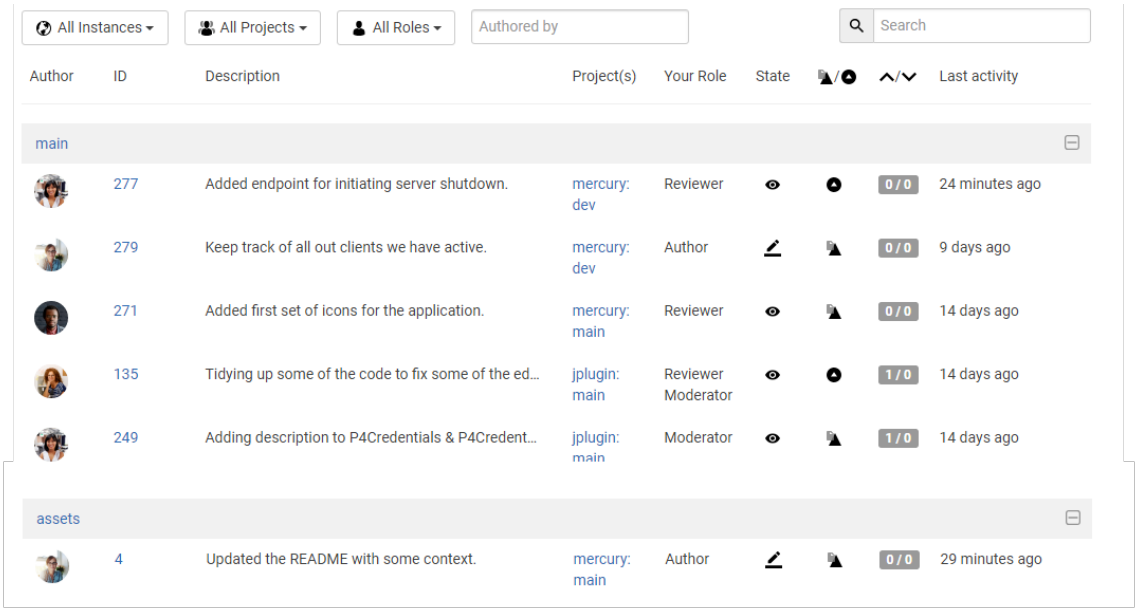
A review is displayed on your global dashboard if any of the following criteria are met:

- You are a reviewer or required reviewer, the review status is **Needs Review** and you have not already voted on it.

- You are a member of a reviewer group or a required reviewer group, the review status is **Needs Review** and you have not already voted on it. The review will remain on your dashboard even if the group has met its criteria if you have not already voted on it.
- You are the review author and the review status is **Needs Revision**, or **Approved** (only if the review is approved but not committed).
- You are a moderator or a member of a moderator group, the review status is **Needs Review**, the **Minimum up votes** requirement for the branch is satisfied, and one of the following is true:
  - There are no required reviewers.
  - All of the required reviewers have up-voted the review.
  - You are the last remaining required reviewer for the review.

> **Tip**
> If moderator behavior is configured to require approval from one moderator per branch and the review spans multiple moderated branches:
>
> - You will see the review in your dashboard if the review has not been approved by another moderator from your branch.
> - You will be able to **Approve and Commit** the review if it has been approved by moderators from all of the other branches.

Example global dashboard showing a number of Helix server dashboards:



Helix server header bars

The dashboard for each Helix server instance is displayed under the header bar for that instance. The header bars are collapsible allowing you to temporarily hide Helix server instances you are not currently interested in.

Tip

- Click on the Helix server name in the header to open Swarm for that instance in a new tab.
- Click on the header to the right of the Helix server to collapse the dashboard for that instance. Click again to expand the dashboard for the instance .

Filtering

The global dashboard can be filtered to display only reviews from a particular Helix server instance, project, authored by a particular user, or matching a role. The filter buttons filter all of the instances on the global dashboard. The filter buttons are always in view as you scroll up and down the page so that you can quickly modify the filters and see just the reviews you want to see. Click the **Reset** button to reset these filters.

Filtering options are:

- **Instances**

  You can filter by the Helix server instance, limiting results to **All Instances** or to an individual instance.

- **Projects**

  You can filter by the project the review is part of, limiting results to **All Projects** or to an individual project. The **All Projects** drop down will only show projects for which there are reviews in your dashboard.

- **Roles**

  You can filter by your specific role in a review, limiting results to reviews for which you are the author, a reviewer, a required reviewer, or a moderator. The **All Roles** drop down will only show roles for which there are reviews in your dashboard.

- **Authored by**

  You can filter the reviews to only those that have been authored by a certain user. Type in this field to get a drop down list of users to filter by.

- **Reset** (only displayed if one or more filters are set):

  Clicking the **Reset** button resets all dashboard filters back to their defaults.

- **Search**

  Typing in the search field filters the reviews by description and review ID.

Review fields

The dashboard for each Helix server shows a summary of the information for each review.

Reviews that appear here are those which are waiting for action from you. The information presented should help you prioritize what to work on next.

- **Author**

  The author of this review.

- **ID**

  The ID of this review. Click on this to go to the review page.

- **Description**

  The review description. It may be truncated if it is too long, in which case click on the description to expand it.

- **Project(s)**

  List of project branches this review covers. A review may span multiple branches and projects. Click on one of them to navigate to the project page for that branch.

- **Your role**

  The reason this review is in your dashboard. This can be Author, Reviewer, Required Reviewer, or Moderator.

- **State**

  The current status of the review. This can be Needs Review, Needs Revision, or Approved.

  > **Note**
  > The Approved state only applies to review authors, it is only shown for a review that has been approved but has not been committed.

- **Type**

  The type of review. This can be Pre-commit or Post-commit.

- **Votes**

  The double column of votes displays the number of up votes and down votes for the review.

- **Last activity**

  The last time that any changes were made to the review, including votes, comments, commits, and file changes.

Navigating directly to a specific Helix server instance in Swarm

If you want to quickly visit a specific Helix server instance in Swarm without going via the global dashboard, include the server name in the URL, for example: **`https://swarm.company.com/serverA`**.

Once you are viewing the Helix server in Swarm, Swarm works as a standard single Helix server Swarm system.

Tip

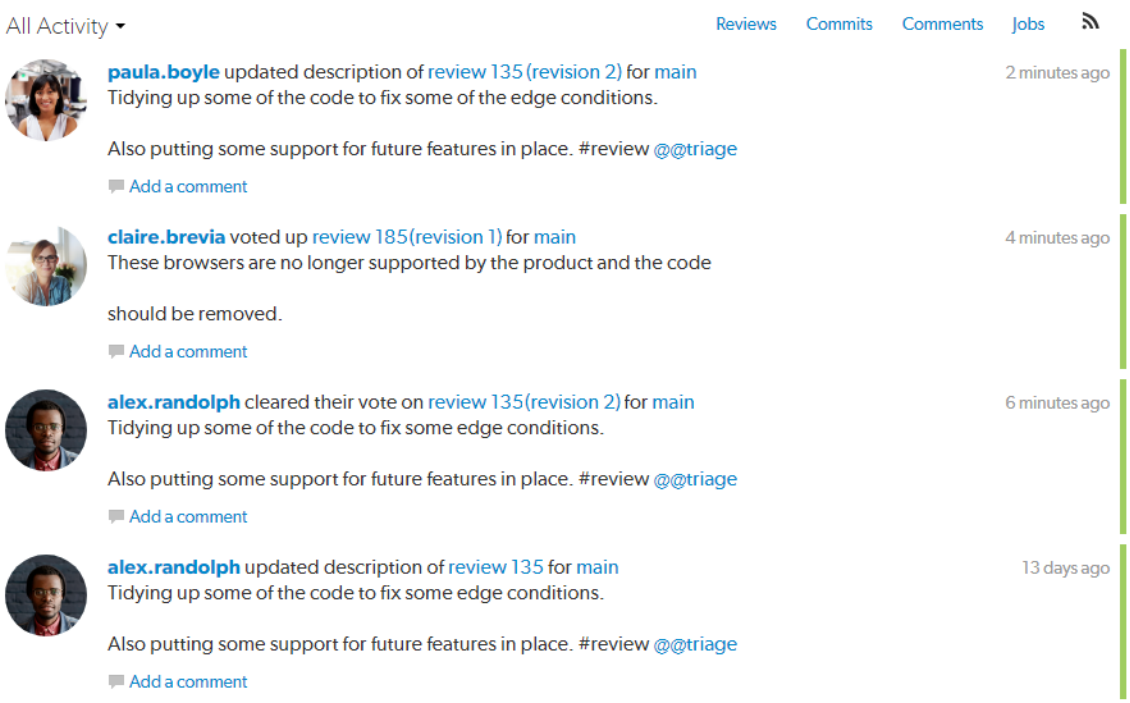
- To browse jobs on **serverA**, navigate to: **`https://swarm.company.com/serverA/jobs`**
- To browse reviews on **serverB**, navigate to: **`https://swarm.company.com/serverB/reviews`**
- To view the dashboard for **serverB**, navigate to **`https://swarm.company.com/serverB/#actionable-reviews`**

Activity streams

An activity stream displays a list of events that have occurred recently in the associated Helix server.

For example, an event is added to the activity stream whenever:

- Changelists are checked in
- Jobs are created
- Code reviews are updated
- Comments posted
- Projects are created or edited
- Workflows are created, edited, or deleted
- Tests are created, edited, deleted, pass, or fail



Activity streams are presented for global server activity, as well as for events occurring for projects and users. Logged-in users can click the **Activity** drop-down menu to choose between viewing all activity or just the activity of the projects and users that they are following.

Activity streams can be filtered to only display events related to reviews, changes, comments, or jobs. Click a filter label to enable that filter. Click the label again to disable the filter, or click a different filter to change filters.

When active, each filter label displays a distinct background color that matches the color stripe on the right side of each event in the activity stream, to help quickly identify each event's type.

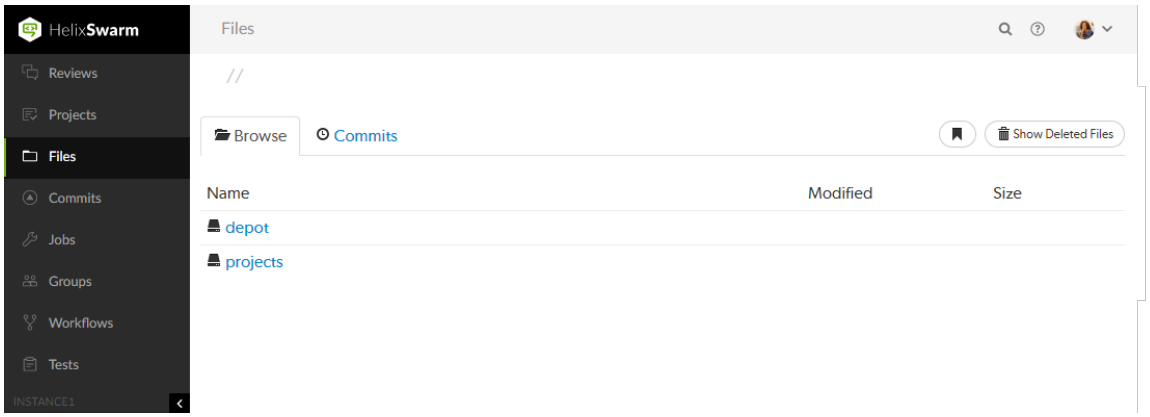
RSS icon Anyone can subscribe to an RSS (Really Simply Syndication) feed for a particular activity stream so that events can be monitored in your favorite feed reader.

Events within an activity stream contain links to their respective resources. Click the links to visit users, changelists, projects, comments, and more.

Each activity stream starts with as many as 50 events. As you scroll down the page, Swarm fetches additional events in batches of 50 until the stream is exhausted. For a long-running server, or a server with significant activity, it could take quite a while to receive all of the events.

Files

Helix server's primary task is to version files, so Swarm makes it easy to browse the depot. Start browsing by clicking **Files** in the menu.



- Swarm displays a list of breadcrumb links to help you quickly navigate to higher level directories quickly.

  // depot / Jam / MAIN / src / Jam.html #2

- Links with folder icons represent directories of files within the depot. Click a directory link to display the contents of that directory.
- Click the .. link with the up-arrow icon .., when it appears, to navigate to the current directory's parent.
- Links with dog-eared page icons represent individual files within the depot. See "File display" on the next page for more information.
- Click the **Commits** tab to display the list of changes made to files in the current directory, or any directories it contains. See "Commits" on page 252 for more information:

  Browse Commits

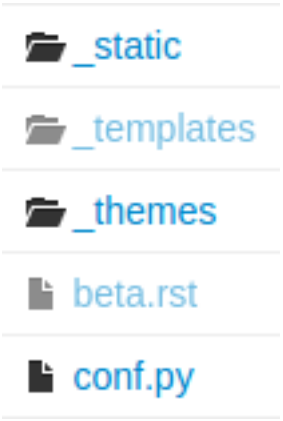
Swarm creates links for files and directories wherever they appear in the Swarm UI.

> **Note**
> Directories that start with a period, for example .git-fusion, are sorted to appear at the end of the list of directories. The `..` link, when it appears, always appears first.
>
> Similarly, files that start with a period, for example `.htaccess`, are sorted to appear at the end of the list of files.

Browsing deleted files and folders

When the **Show Deleted Files** button is clicked, Swarm toggles the inclusion of deleted folders and files in the file display.

Deleted folders and files are presented slightly muted compared to non-deleted entries:



File display

When Swarm is asked to display a file, if the file is a type that Swarm can display, Swarm presents the file's contents.

Buttons: the actual buttons displayed depends on the file type you are viewing.

- Click the **Shorten URL** button to display a short-link to the file.
- Click the **Blame** button to add a column to the display that identifies the userid responsible for each line of the file.
- Click the **Open** button to display the file content with no surrounding page markup.
- Click the **Edit** button to edit the file, see "File edit" on page 250.
- Click the **Download .zip** button to download a compressed version of the file, see "Download files as a ZIP archive" on page 251.
- Click the **Download** button to download the file.

When a file is opened it will by default be truncated to 1 MB in size. This limit can be increased (or removed) via the Swarm configurables. See Files configuration for details. This limit does not apply to downloaded files.

Along with the file name, Swarm displays the version number for the currently displayed file in the breadcrumb. For example, the following breadcrumb indicates that version 2 of **`Jam.html`** is being displayed:

If the version of a file being previewed has been deleted, the version number is displayed in red.

Every version of a file is available on the **Commits** tab.

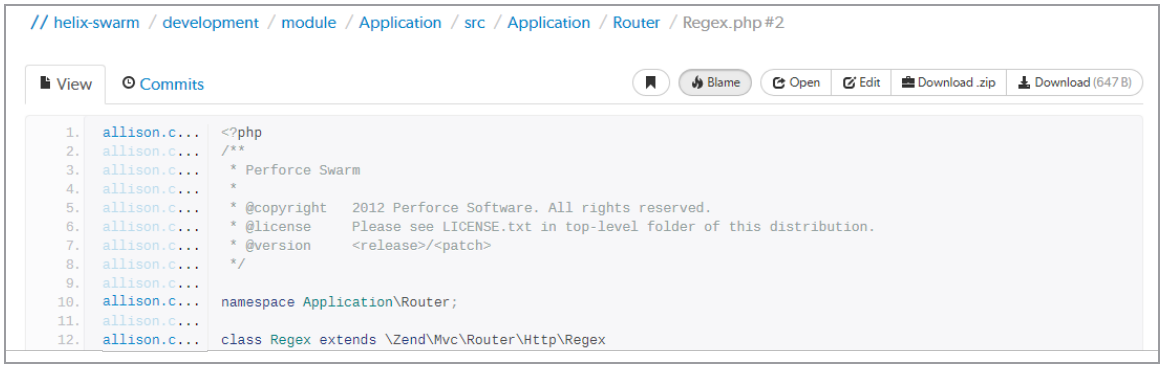
Text files

Swarm displays the contents of text files (including the Helix server filetypes unicode and UTF16) with line numbers. When possible, syntax highlighting is applied to make identification of various elements within the file easier.

For more information on Helix server filetypes, see File Types in *Helix Core P4 Command Reference*.

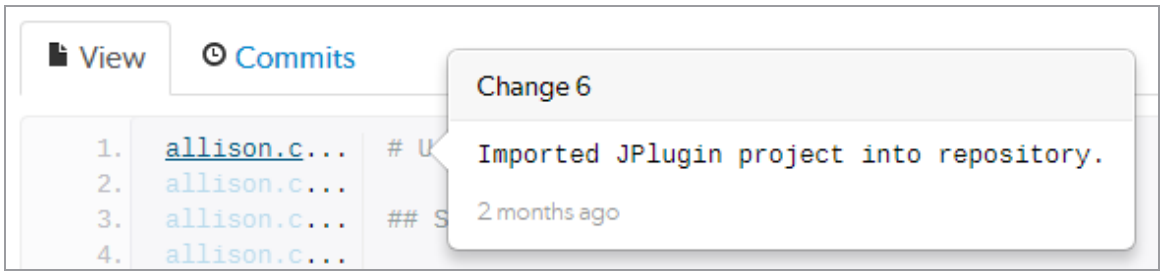
> **Tip**
> Swarm can be configured to display some non-UTF8 characters, see "Non-UTF8 character display" on page 504 for details.

Click **Blame** to add a column to the display that identifies the userid responsible for each line of the file:



Each userid presented is a link that, when clicked, displays the changelist that provided the associated text. Muted userids indicate that the associated text is from the same changelist as the line above. For example, the userid *allison.c* is responsible for lines 1, 10, and 12 in the screenshot above.

When you hover your pointer over a userid in the blame column, a tooltip appears displaying the associated changelist description:



When there are no lines displayed, for example when you are viewing empty or shelved files, the **Blame** button is disabled.

Markdown files

Swarm displays Markdown files in two tabs:

- **Markdown**: by default, Markdown content is displayed but it is limited to prevent the execution of raw HTML and JavaScript content. This can be configured by the Swarm administrator, see "Markdown" on page 512. This tab is only displayed for Markdown files.
- **View**: displays the contents of Markdown files as plain text in the same way as "Text files" on the previous page are displayed.

> **Tip**
> Valid Markdown file extensions are: **`md`**, **`markdown`**, **`mdown`**, **`mkdn`**, **`mkd`**, **`mdwn`**, **`mdtxt`**, **`mdtext`**.

Markdown tab

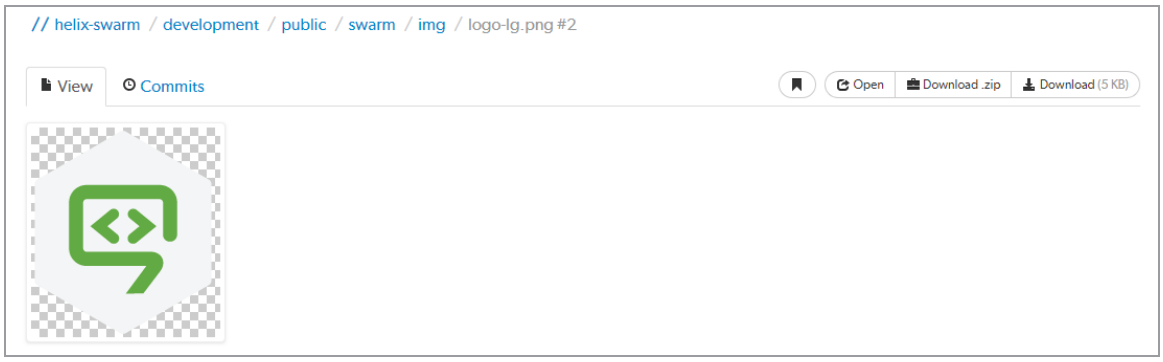
Click the **Markdown** tab to view the file content in Markdown:

View tab

Click the **View** tab to view the Markdown file in plain text:

Images

Swarm displays web-safe images:



The checkerboard background in this example is not part of the logo; it helps identify where transparency exists.

Many browsers can display SVG images with no additional plugins, so Swarm attempts to display SVG images rather than displaying the image's definition. When you use a browser that cannot natively display SVG images, you see the broken image icon.

When imagick (an optional module that integrates ImageMagick into PHP) is installed, Swarm can also display the following image formats: BMP, EPS, PSD, TGA, TIFF.

3D models

Swarm 2014.1 includes support for displaying select 3D model file types in the browser:

Supported file types include:

- **DAE** - including any referenced web-safe texture images.
- **STL** - both binary and ASCII versions of the format.
- **OBJ** - including any referenced MTL files, and web-safe texture images.

When Swarm can display a 3D model, it renders a generic grid *stage* and places the model in the center, scaled to make viewing straightforward. A toggle control appears in the top right: when enabled, you can control the view with the mouse, and when not enabled, permits auto-rotation to occur (when possible).

1. Click and hold the **left mouse button** to begin rotating the view. Drag while holding the left mouse button to rotate the view.
2. Click and hold the **right mouse button** to begin panning the view. Drag while holding the right mouse button to pan the view.
3. Roll the **mouse wheel** up or down to adjust the magnification of the view.

When possible, a second control appears allowing you to toggle between showing the model with surfaces, or just showing the model's wireframe.

> **Note**
> For systems with hardware acceleration, if your browser supports WebGL and hardware acceleration is enabled, Swarm renders the model and enables auto-rotation.
>
> For systems without hardware acceleration or WebGL, but your browser supports HTML5 canvas elements and JavaScript TypedArrays, Swarm renders the model but auto-rotation is disabled. Rendering is likely to be slow and rendering quality is likely to be low.
>
> For browsers without HTML5 canvas elements and JavaScript TypedArrays, no rendering is attempted; instead, users see a message indicating that the browser is not supported.

Other file types

It is possible to view other file types in Swarm, through the addition of additional modules, or by installing "LibreOffice" on page 451 on the Swarm host.

When the file is a type that Swarm cannot display, Swarm presents the file's history, along with the **Download** button:

File edit

Important
File edit is a Technology preview feature.

Technology preview features:

Technology Preview features are currently unsupported, might not be functionally complete, and are not suitable for deployment in production. These features are provided to the customer to solicit interest and feedback, with the goal of full support in future releases. Customers are encouraged to provide feedback and functionality suggestions for Technology Preview features before they become fully supported.

Note
Requirements:

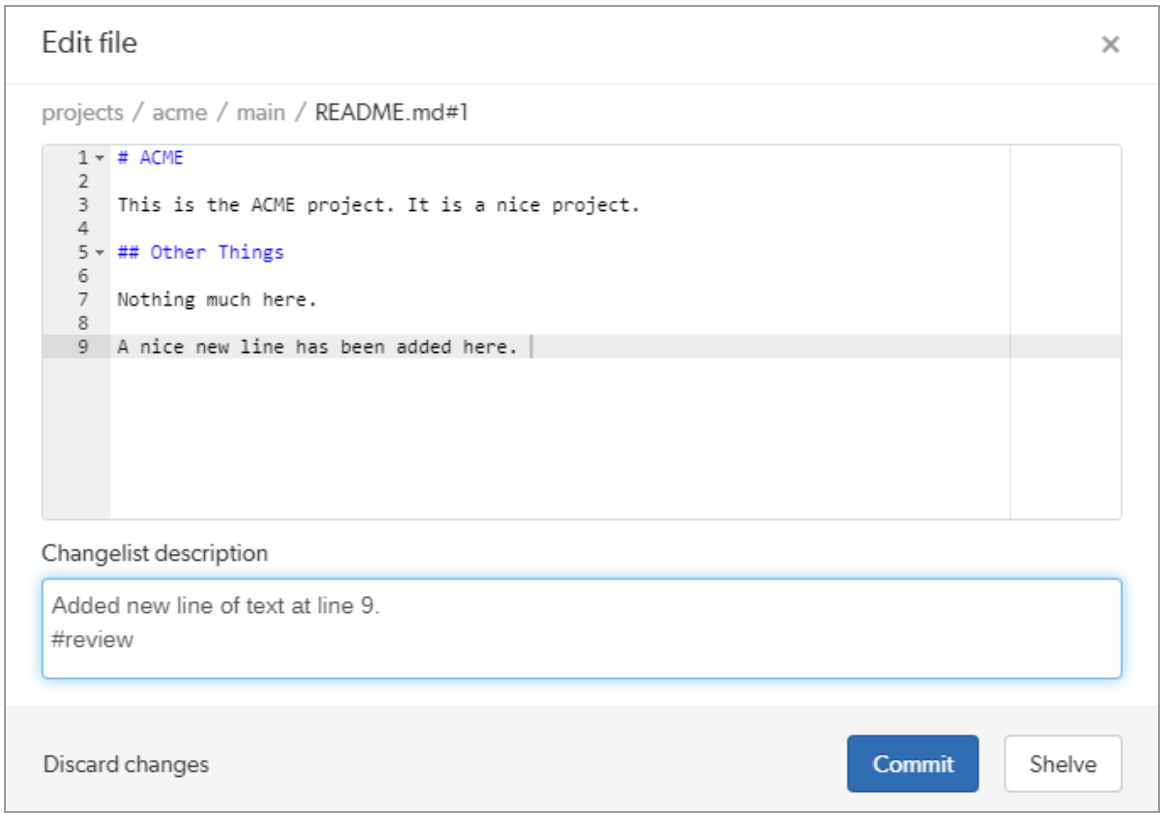
- You must have permissions to edit the file.
- If the file version changes while you are editing it or if you are not editing the Head revision of the file, committing your file changes will overwrite the current head revision of the file.

  If you are unsure, shelving your change is a safer option because it will not overwrite the head revision of the file and you can do a manual resolve when you commit the shelf.
- File edit from the **Files** browser page is enabled by default but can be disabled by your Swarm administrator, see "allow_edits" on page 500.

You can edit a file from the Swarm **Files** browser page and shelve or commit your file changes.

To edit a file from the Swarm Files page:

1. Navigate to the file on the **Files** page.
2. Click the **Edit** button above the file to open the file editor.



3. Edit the file and add a **Changelist description**.

> **Tip**
> **Optional:** you can create a review for your change or add it to an existing review by including a review keyword in the **Change description**. For instruction on creating a review and adding a changelist to a review using review keywords, see "Create a review" on page 548 and "Add a changelist to a review" on page 548.

4. Click **Commit** to commit the file or **Shelve** to shelve the file.

   To discard your changes, click **Discard changes** and confirm that you want to discard your changes when requested.

Download files as a ZIP archive

The **Download .zip** button is used to download the selected files or folders in the Helix server as a ZIP archive. This makes it easy to get a copy of files without having to setup a client.

Note
The **Download .zip** button is not displayed if the zip command-line tool is not installed on the Swarm server. For information about installing, and configuring the zip command-line tool, see Zip archive.

When you click the **Download .zip** button, Swarm performs the following steps:

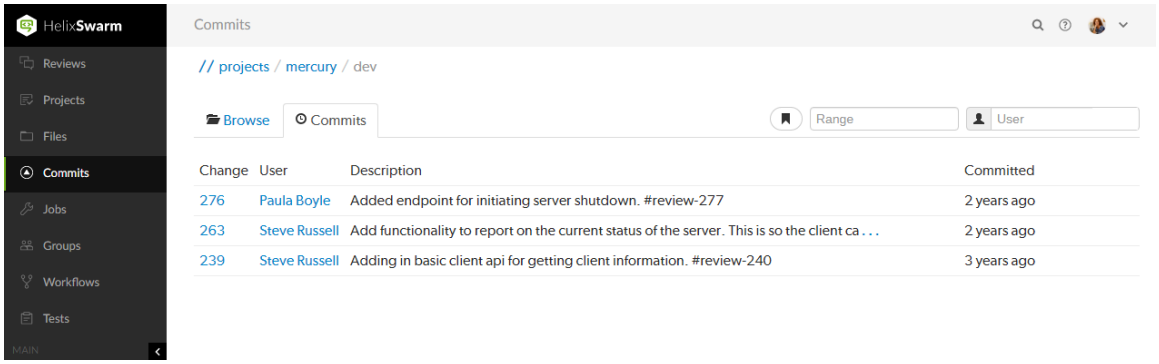
1. Scans the files/folders:
   - Checks that you have permission to access their contents, according to the Helix Core server protections.
   - Checks that the total file size is small enough to be processed by Swarm.
2. Syncs the file contents to the Swarm server from the Helix Core server.
3. Creates the ZIP archive by compressing the file content.
4. Starts a download of the generated ZIP archive.

Note
- You might not see all of the above steps; Swarm caches the resulting ZIP archives so that repeated requests to the same files/folders can skip the sync and compress steps whenever possible.
- If an error occurs while scanning, syncing, or compressing, Swarm indicates the error.

Commits

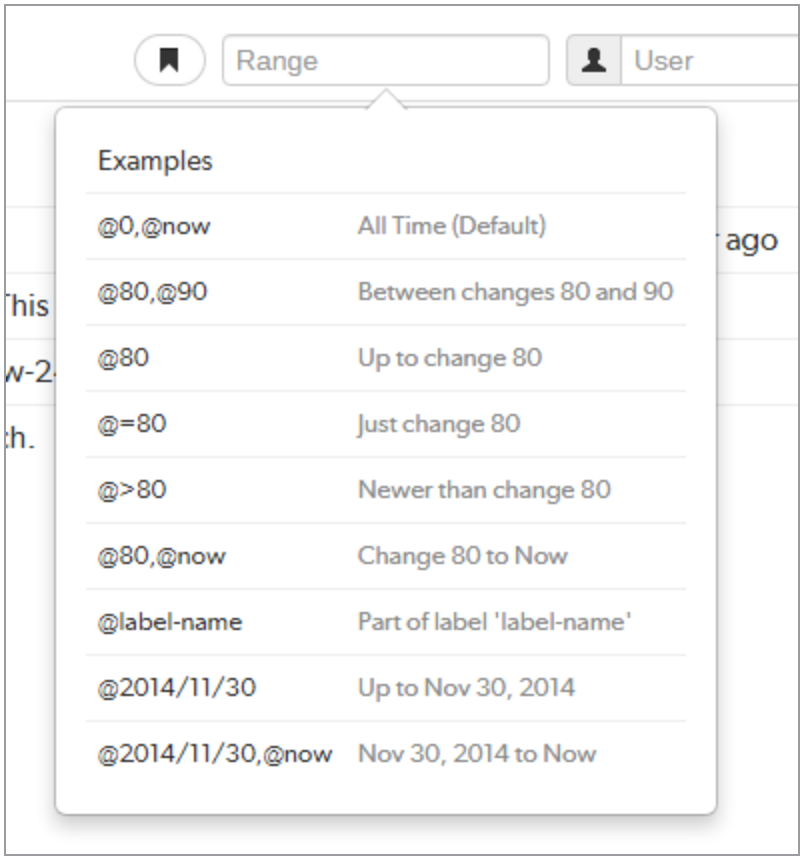
Whenever a new version of a file is checked into the Helix server, a commit record is created. Begin browsing the history of commits by clicking **Commits** in the menu.

When you are viewing a particular file or directory, clicking the **Commits** tab displays the commits for that location in the depot.



Range filter

The **Range** field lets you filter the list of changes for the depot path being viewed. When you click the **Range** field, a dropdown syntax guide appears providing sample commit filtering expressions.

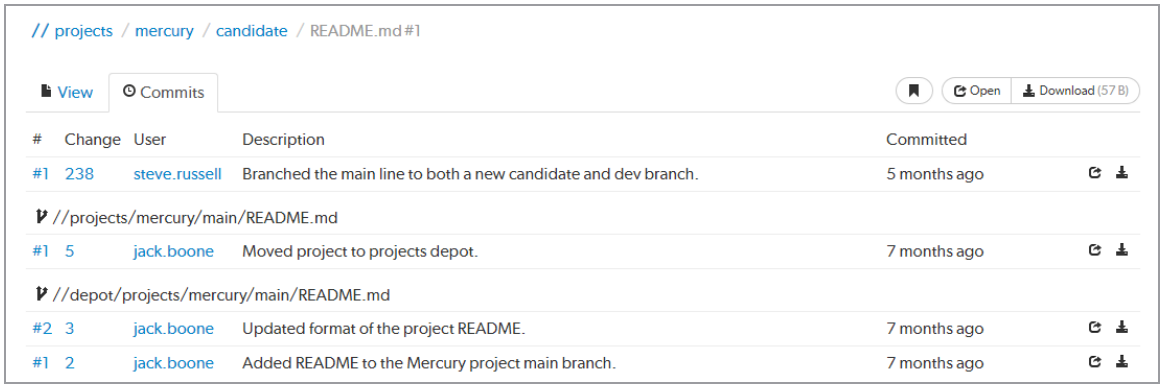


The expressions that can be used within the **Range** field include:

- @0,@now (the default): displays all commits for the current depot path.
- @80,@90: displays all changes between 80 and 90.
- @80: displays all changes up to change 80.
- @=80: displays only change 80. Change 80 might not involve the current depot path, so there may be no commits to display.
- @>80: displays all changes newer than change 80.
- @80,@now: displays all changes from change 80 to now.
- @label-name: any changes represented by label *label-name*.
- @2014/11/30: displays all changes up to November 30, 2014.
- @2014/11/30,@now: displays all changes from November 30, 2014 to now.

File Commits

A file's commit history presents each version of a file that the Helix server knows about, including the change number, userid, change description, time ago, along with **Open** and **Download** buttons.



Swarm also displays *contributing commits* when available, such as when a file has been renamed, or integrated from another location in the Helix server.

If a commit represents a deleted revision, the **Open** and **Download** links are replaced with a trashcan icon to indicate that this version is no longer available.

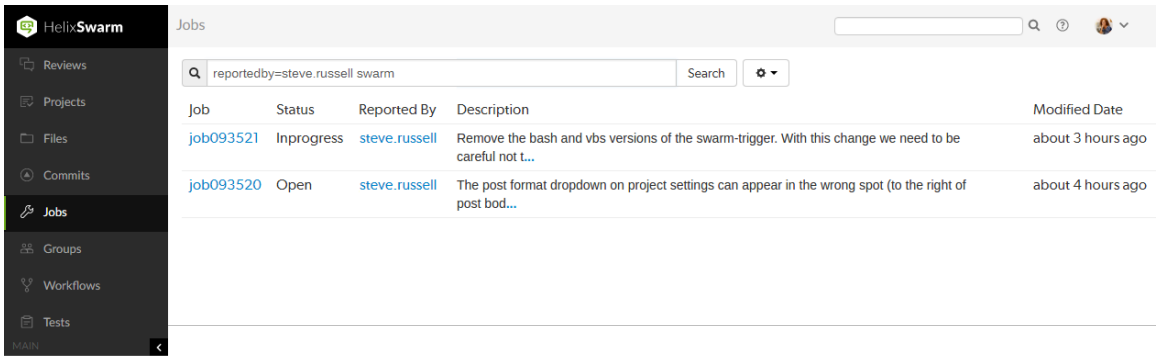
Remote depot commits

When your Helix server has a remote depot configured, you can browse the contents of the remote depot, but remote depots do not share their commit history. If you attempt to view the commits of a file provided by a remote depot, Swarm displays:

Jobs

Jobs are a component of Helix server's defect tracking system and a record of bugs found or improvement requests. Jobs can be associated with changelists to create *fix* records, indicating the work that solved the problem or provided the requested feature. Begin browsing jobs by clicking **Jobs** in the menu.

You can search available jobs by entering a job filter in the search box. Words, phrases, and `field=value` pairs can be entered. For example, entering `reportedby=steve.russell swarm` displays jobs that user *Steve Russell* has reported that also contain the word *swarm* in the description.



The fields you can search for depend on the jobspec defined in your Helix server.

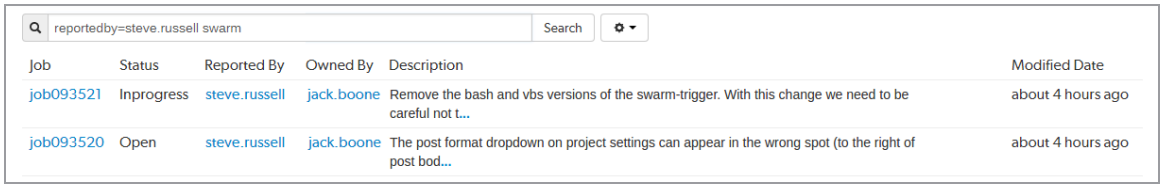
Edit the Job columns

Configure the columns that are displayed:

1. Click the **Select Columns** button  beside the search field to display the dropdown menu. The dropdown menu shows all of the available jobspec fields.
2. Select the columns to display using the checkboxes in the dropdown menu.
3. Drag the column label up or down the list to change the order the columns are displayed in.
4. Click the **Select Columns** button again, or a blank portion of the page, to hide the menu.

Change the column order from the **Jobs** page:

1. Click a column heading and drag it to the left or right to move the column to a new position.
2. The column display updates as columns are dragged.



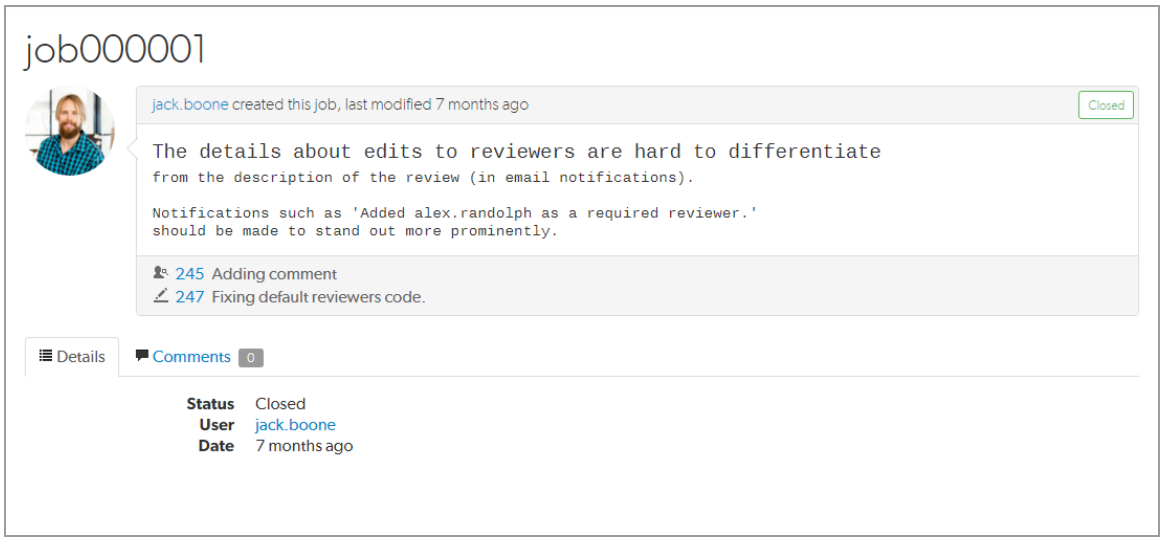
For more information on customizing job specifications, see Customizing Helix server: Job Specifications in *Helix Core Server Administrator Guide*.

Job display

Jobs are typically identified with the word job followed by six digits, e.g. `job000123`.

View a specific job by clicking on a linked job identifier, or by visiting the URL: **`https://myswarm.url/jobs/jobid`**.

When Swarm displays a job, the presentation is similar to:

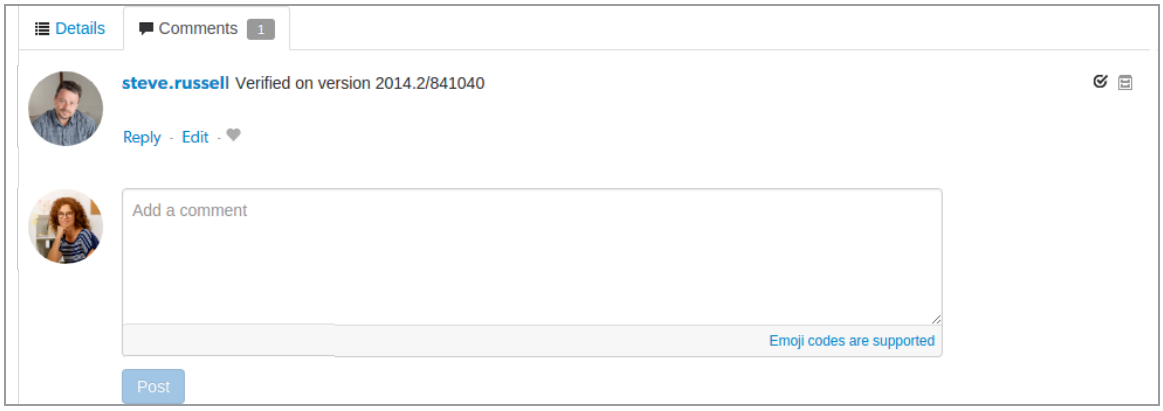


The upper portion of the job presentation includes:

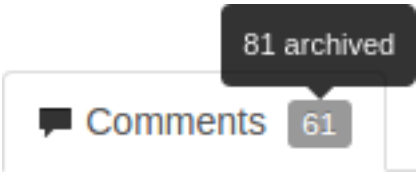
- The avatar and userid of the user that created the job
- The job's creation time
- If changes have been made to the job, the modifying userid and time
- A status indicator
- The job's description
- If any changelists have been submitted that *fix* the job, a list of those changelists and their descriptions. Each associated changelist includes an icon to represent their type: (review), (pending), (committed)

The lower portion of the job presentation lists all of the keys configured in your Helix server's jobspec. Swarm inspects the jobspec and enhances the presentation of fields it recognizes. For example, date fields display as *time ago*, and links are created for *userids*.

Click the "Comments" on page 267 tab to view any comments added to the job, or to add a comment.



Adding a comment sends a notification. The **Comments** tab displays the number of open comments associated with the job. If you hover your mouse over the comment count, a tooltip is displayed showing how many comments are archived:



For more information on customizing job specifications, see Customizing Helix server: Job Specifications in *Helix Core Server Administrator Guide*..

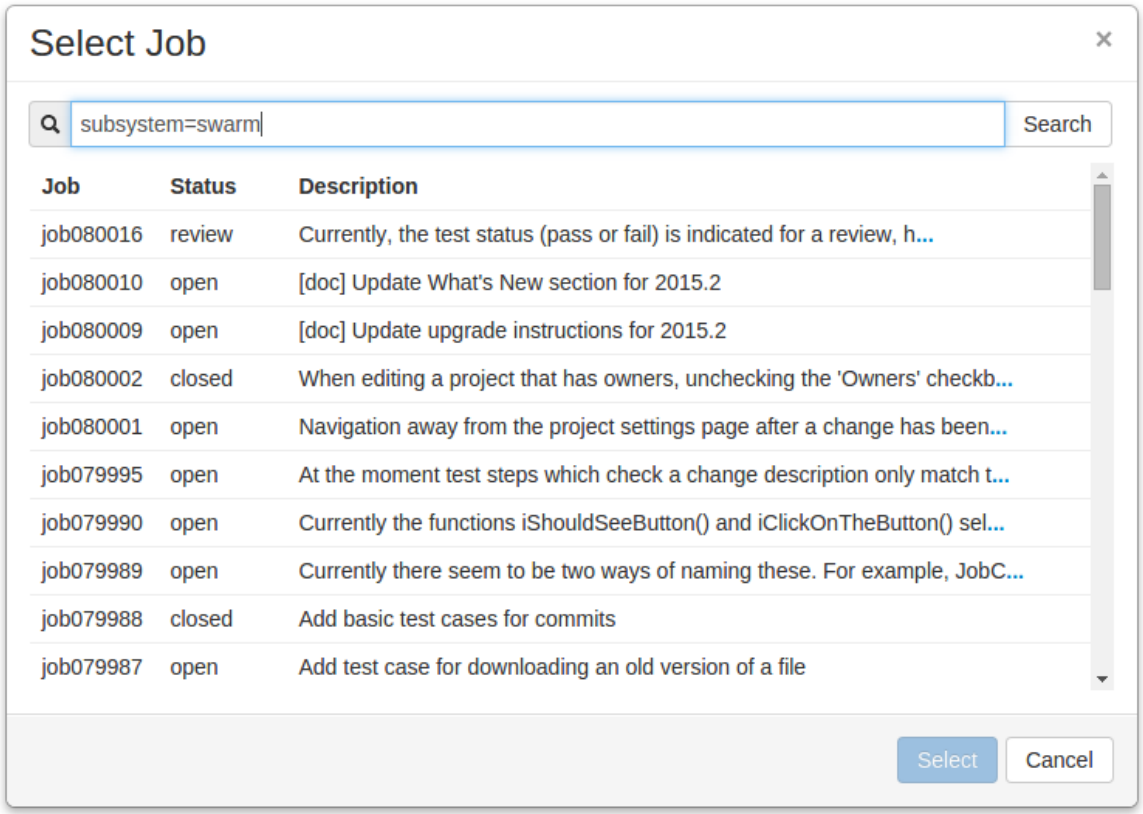
> **Note**
> The default Helix server job specification contains very few fields. Adding fields to record additional information, such as the modification time and userid, reporting time and userid, can assist Swarm use appropriate terminology when describing the current disposition of a job.

Add jobs

Swarm does not provide the ability to create new jobs in the Helix server, but jobs can be added to changelists or reviews:

1. Navigate to a changelist or review.
2. Click the **Add Job** link 

.
3. Scroll through the available jobs, or enter job search criteria to search available jobs.



For more information on job search criteria, see Jobs in *Helix Core Server User Guide*.

4. If you find the job you want to add, click its row to highlight it and then click **Select**. Or, double-click the desired job to add it.
5. If you do not find the appropriate job, click **Cancel**.

> **Note**
> If you attempt to add a job to a review that affects a single project, Swarm applies the project's job view filter to display only jobs that affect the project. It is not currently possible to expand the filter to include jobs outside of the project.

Unlinking jobs

Swarm does not provide the ability to delete jobs from the Helix server, but jobs can be unlinked from changelists or reviews:

1. Navigate to a changelist or review that has an associated job.
2. Click the **X** button beside the job.
3. When prompted for confirmation, click **Unlink** to unlink the job.

Changelists

Changelists are the basic unit of versioning work in Helix Core server. A changelist is a list of files, their revision numbers, and the changes made to those files. Commits is a shorter synonym used throughout Swarm.

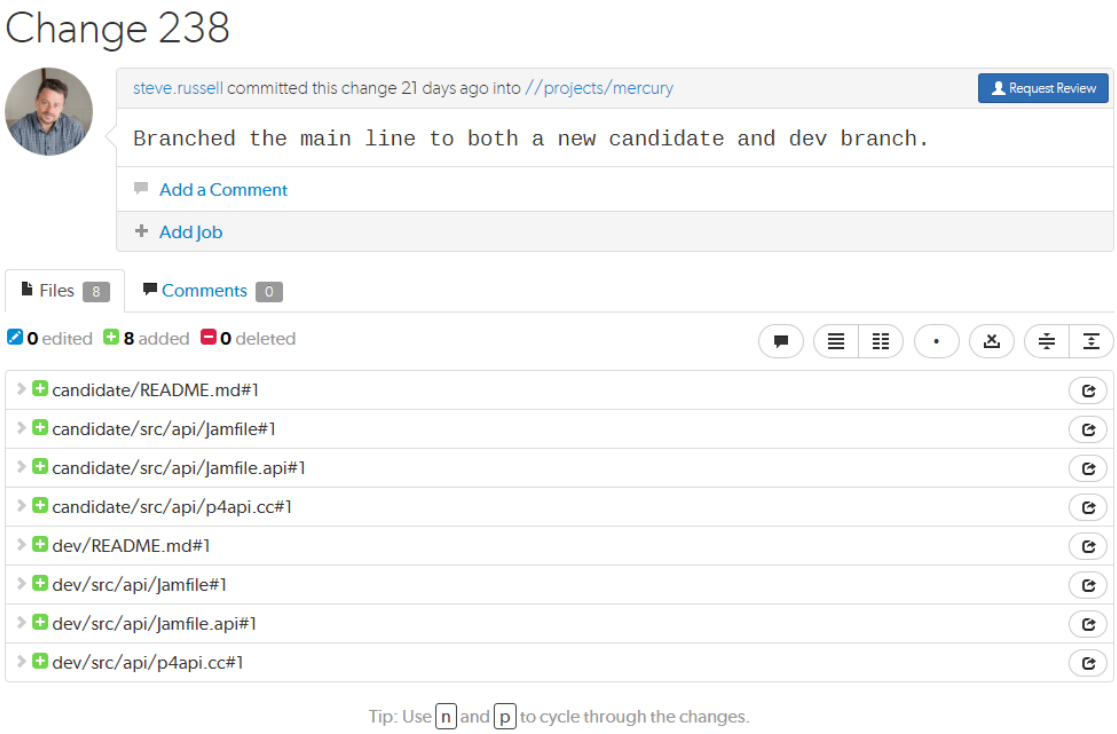
More information is available here:

- Browsing changelists
- Browsing a user's shelved changelists
- Swarm's internal use of changelists to facilitate code reviews

Changelist display

View a specific *changelist* by clicking on a linked changelist number, or by visiting the URL: **`https://myswarm.url/changes/changelist number`**.

When Swarm displays a change, the presentation is similar to:



Swarm supports stream specs in your workspace using the Private editing of streams feature. If a changelist or review contains a stream spec, it will be displayed first in **Files** with the prefix **stream: //**, for example: **`stream://MyStreamDepotName/MyStreamSpecLocationName`**. A changelist/review can only contain one stream spec.

The changelist display includes:

- The avatar and userid of the user who made the change
- The time the change was made
- The common depot location containing all the files included in the change
- The **Download .zip** button, used to download a ZIP archive that contains all of the files in the changelist:

  > **Note**
  > The **Download .zip** button is not displayed if the zip command-line tool is not installed on the Swarm server. For information about installing, and configuring the zip command-line tool, see Zip archive.

  When you click the **Download .zip** button, Swarm performs the following steps:

  1. Scans the files/folders:
     - Checks that you have permission to access their contents, according to the Helix Core server protections.
     - Checks that the total file size is small enough to be processed by Swarm.
  2. Syncs the file contents to the Swarm server from the Helix Core server.
  3. Creates the ZIP archive by compressing the file content.
  4. Starts a download of the generated ZIP archive.

  > **Note**
  > - You might not see all of the above steps; Swarm caches the resulting ZIP archives so that repeated requests to the same files/folders can skip the sync and compress steps whenever possible.
  > - If an error occurs while scanning, syncing, or compressing, Swarm indicates the error.

- If the change was involved in a code review, a link to the review along with a **View Review** button.
- The description of the change
- A list of jobs that this change *fixes*, if any. You can add jobs and unlink jobs here.
- The list of files included in the change, including any folders between the common depot location and the file, and the file's version number.
- A tab to review any comments made regarding the change, or any of its files.

Each file is presented in a diff display, showing you whether the file was added, modified, or deleted. For text-based and image files, Swarm can display any changes made within the file. For changes with only a single file, the diff display is the default; otherwise each file is listed. Click the filename to see the diff display. See "Diffs" on the next page for more information.

The **Request Review** button indicates the current state of this change; no Review record has been created. Clicking **Request Review** starts a code review for this change. For more information, see "Start a review" on page 363.

Important
If your Helix server is configured as a commit-edge deployment, and your normal connection is to an edge server, Swarm refuses to start reviews for shelved changes that have not been promoted to the commit server.

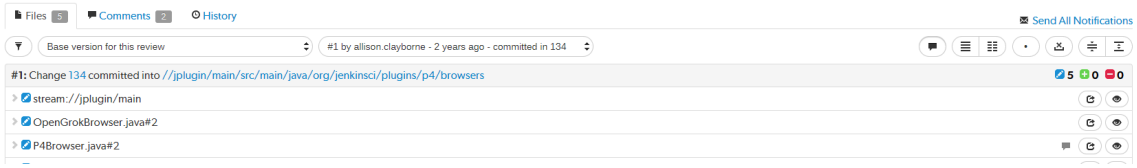
Within Swarm, this means that the **Request Review** button does not appear for unpromoted shelved changes. Outside of Swarm, attempts to start reviews for unpromoted shelved changelists appear to do nothing. Ask your Helix server administrator for assistance if you cannot start a review.

An administrator of the Helix server can automatically promote shelved changes to the commit server by setting the configurable **`dm.shelve.promote`** to 1.

When a file in a changelist has one or more associated comments, an icon appears near the far right of the file's entry.

Diffs

When you view a changelist or code review, the associated files are presented as *diffs*, short for differences, showing you how they have changed.



Swarm supports stream specs in your workspace using the Private editing of streams feature. If a changelist or review contains a stream spec, it will be displayed first in **Files** with the prefix **stream:** //, for example: **`stream://MyStreamDepotName/MyStreamSpecLocationName`**. A changelist/review can only contain one stream spec.

Note
If you append a changelist with a stream spec to a review that already contains a stream spec, the spec in the changelist replaces the original one in the review. If it is a different spec from the original spec in the review, Swarm cannot display the diff between them and displays **File content unchanged**.

The review revision selectors are used to specify which revisions of a review you want to *diff*, they are located in the **Files** tab above the list of files.

- **Left dropdown selector:** the base version for the review is selected by default, base is the version of the file that was checked out of the depot before it was changed for this review. The current version in the depot, sometimes called Head can be selected, if there are multiple revisions of the review a specific revision can be selected.
- **Right dropdown selector:** the latest revision of the review is selected by default. If there are multiple revisions of the review, a specific revisions of the review can be selected.

> **Tip**
> If a review consists of one or more Swarm-managed changelists. When comparing versions of a review, Swarm is showing any differences between the selected versions, not the review author's personal changelist. See "Internal representation" on page 339 for details.

The first row of buttons above the files allow you to (left to right):

- **Show Comments** button: toggles the display of comments to inline in files or only in the **Comments** tab.
- **Show Diffs In-Line** button: displays all diffs as inline.
- **Show Diffs Side-by-Side** button: displays all diffs as side-by-side.
- **Toggle Show Whitespace** button: toggles the display of whitespace characters (such as space, tab, and newline) for all files.
- **Toggle Ignore Whitespace** button: toggles the highlighting of whitespace changes in all of the file diffs.
  - **Highlight whitespace changes:** makes it easier to identify changes in file types where whitespace is important. This is the default value.
  - **Ignore whitespace changes:** whitespace changes are not highlighted, this makes it easier to see the important changes in file types where whitespace changes are not important.
- **Collapse All** button: collapses all files
- **Expand All** button: expands all files. By default this button is disabled if there are more than 10 files in the review. For details, see Expand All Limit.

> **Tip**
> The default states for the **Show Comments**, **Show Diffs Side-by-Side**, **Toggle Show Whitespace**, and **Toggle Ignore Whitespace** buttons are set in your user settings, see User Settings.

Each file is presented with an icon indicating whether the file was:

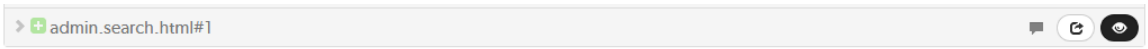
- Added/Branched/Imported
- Edited/Integrated
- Deleted

The file's presentation can be controlled with (left to right):

> **Tip**
> Only the **Show Full Context** button is available for stream specs.

- **Comments in File** icon (only displayed if the file contains comments): indicates that the file contains comments.
- **Show In-line** button: highlights line additions, modifications, and removals in a single pane.
- **Show Side-by-Side** button: highlights additions, modifications, and removals in two panes, with the older version of the file on the left, and the newer version on the right.
- **Show Whitespace** button: makes whitespace characters more visible; spaces show up as dots, tabs show up as arrows that point to a bar, and line endings show up as down-pointing arrows.
- **Ignore Whitespace** button: toggles the highlighting of whitespace changes in a file, making it easier to identify non-formatting changes. Normally, whitespace is not ignored.
- **Toggle Ignore Whitespace** button: toggles the highlighting of whitespace changes in the file diff.
  - **Highlight whitespace changes:** makes it easier to identify changes in file types where whitespace is important. This is the default value.
  - **Ignore whitespace changes:** whitespace changes are not highlighted, this makes it easier to see the important changes in file types where whitespace changes are not important.
- **Show all diffs** button (edited and integrated files only): toggles between displaying all the diffs for the file and only the first few. The limit of what are shown by default is configurable by the administrator (see Max Diffs) and defaults to 1500. If there are fewer than that then this button is hidden.
- **Show Full Context** button (edited and integrated files only): toggles between displaying only the portions of the file that have changed and the full file.
- **Open File** button (edited or integrated files only): opens a new browser tab/window display the full file (where possible), provide access to its history, and a button to download the file.
- **Mark file as read** button (displayed for code reviews only): helps you (and others) keep track of which files have been reviewed. This is particularly useful when a code review consists of many files.

  When clicked, the button color inverts and the associated file is visually muted, to make it easy to distinguish read files from unread files:

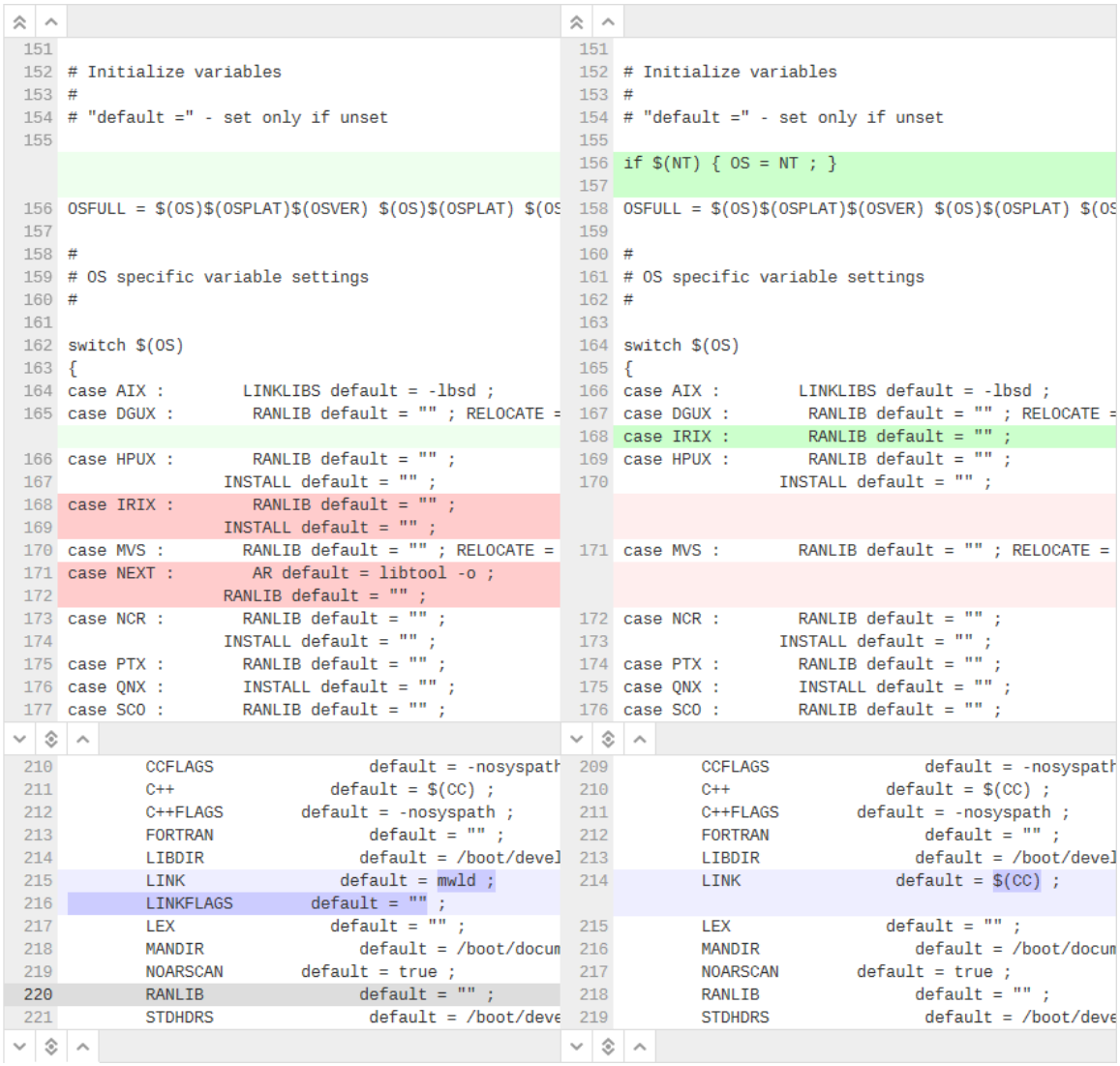
  

  If a file has been marked as read, click the button a second time to reset the status to unread.

Viewing a diff

When you view a diff, the changes are highlighted:

- Red indicates lines that have been removed.
- Blue indicates lines that have been modified.
- Green indicates lines that have been added.



Show more context buttons

The diff presentation displays a concise view of where changes were made within a file, showing the changed lines and only a few lines before and after the change. Sometimes, this concise view needs to be expanded to fully understand the context of the change, use the **Show More** and **Show All** buttons to display extra lines around the change:

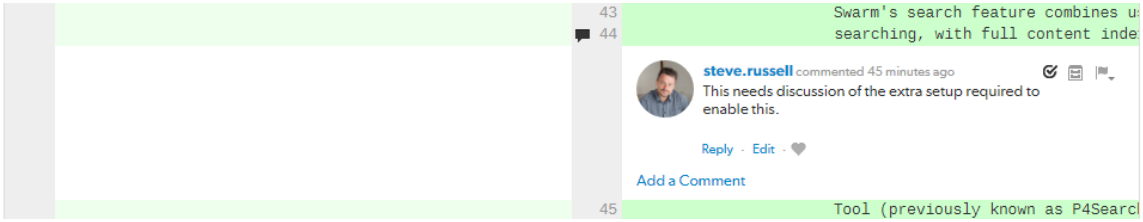
> **Note**
> By default, the number of extra lines displayed when you click the **Show More Lines Above** and **Show More Lines Below** buttons is 10. This setting can be changed by your Swarm administrator, see "More context lines" on page 559.

> **Tip**
> The following buttons are not available for stream specs.

- **Show All Lines to Start of File** button (only displayed for the first change in the file): click to show all of the lines up to the start of the file.
- **Show More Lines for the Code Below** button: click to show 10 more lines above the change, the extra lines are displayed in the pane below the button.
- **Show Entire Section** button: click to show all of the lines between the changes that are above and below the button, the two changes and the lines between them are displayed in a single pane.
- **Show More Lines for the Code Above** button: click to show 10 more lines below the change, the extra lines are displayed in the pane above the button.
- **Show All Lines to End of File** button (only displayed for the last change in the file): click to show all of the lines down to the end of the file.

Comments

Comments can be displayed within the body of a file and appear immediately below the line the commenter targeted for comment. See "Comments" on the next page for more details.



Note
The *comment here* icon appears in the line number column whenever there is a comment. Click the icon to display the comment and click the icon again to collapse the comment. This is useful when the comment display is toggled off.

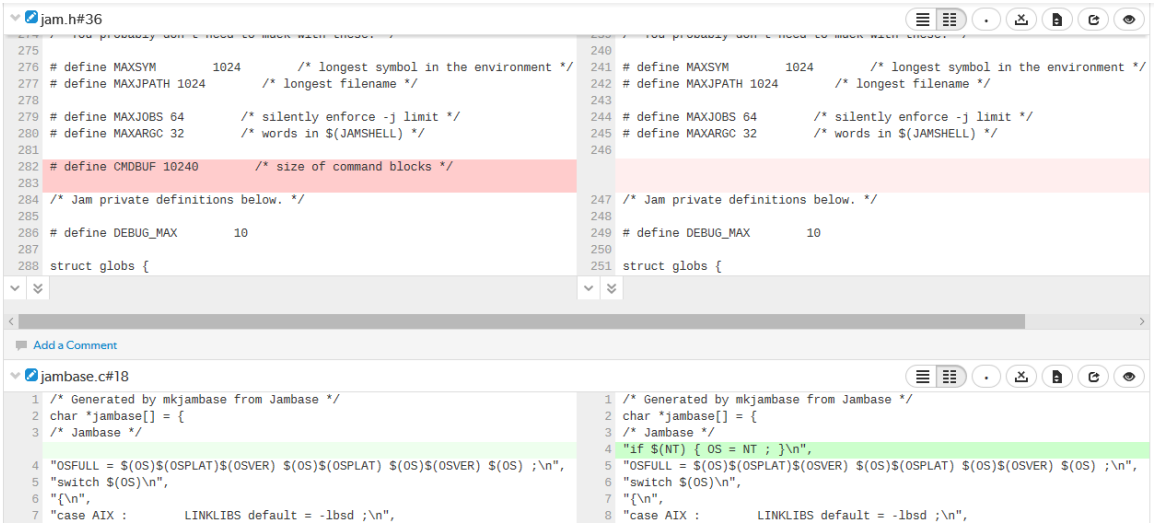
Inline diff view

When viewing a diff in-line, the line numbers for the old version are first, the line numbers for the new version are second, and they are followed by the file content. Some users find it easier to use this view to locate an area that has changed, but they then switch to the side-by-side view to understand the change better. Swarm maintains the scroll position, you do not lose your place in the file after toggling the diff view.

Note
Press **n** on your keyboard to scroll to the next changed area within a file. Press **p** to scroll to the previous change.

Per-file toolbar

When a diff contains multiple files, the changes can be taller than your browser window. Swarm keeps the *per-file toolbar* in view for each file as you scroll, this means that you can identify the file and control its presentation with the buttons on the right of the toolbar.



Comments

Comments are the primary feedback mechanism provided by Swarm. Review comments can be flagged as *tasks* for a lightweight workflow within a review that helps authors and reviewers prioritize review feedback, see "Tasks" on page 270 for details. By default, comment notifications are delayed to allow you to add or edit comments as you progress through a review without sending a notification for each individual comment on the review. Comment notifications are rolled up into a single notification that you can either leave to be sent automatically, or you can send manually, see "Comment notification delay" on page 273.

You can also like comments, add links, add attachments, and add Emoji emoticons, see "Comment features" on page 273 for details.

You can add comments to:

- A changelist, review, or job
- A changelist or review description
- A line of a text file in a changelist, or review
- A file in a changelist, or review

Access comments for a changelist, review, or job, by clicking the **Comments** tab. The number of open (non-archived comments) is displayed in the tab. Hover your mouse pointer over the comment count, the tooltip shows how many comments are archived. See "Archiving comments" on page 280 for details.

Adding comments

This section describes how to add comments.

Related comment features:

- Comment notification delay
- Reply to comment
- Emoji
- Links
- Attachments
- Markdown

> **Tip**
> You can use "@mention" on page 326 in comments. An @mention includes the specified user in the review, and they will receive a notification whenever there is an update to the review.

Commenting on a changelist, review or job

1. From the changelist, review, or job page: click **Comments** to view the **Comments** tab.
2. Add your comment in the text area.
3. Click **Post**.

Commenting on a changelist or review description

1. From the changelist or review page description area: click **Add a Comment** or **X Comments** (where **X** is the number of comments that already exist).
2. Add your comment in the text area.
3. Click **Post**.

> **Tip**
> To hide the description comments, click **X Comments** (where **X** is the number of description comments that exist). To display the comments again, click **X Comments**.

Commenting on a specific line in a file in a changelist or review

1. From the changelist or review page: click **Files** to view the **Files** tab.
2. Click on the line you want to comment on.
3. Add your comment in the text area.
4. Click **Post**.

Commenting on a file in a changelist or review

1. From the changelist or review page: click **Files** to view the **Files** tab.
2. If there are multiple files, click the file you want to comment on to expand its view.
3. Click the **Add a Comment** link in the footer of the file display.
4. Add your comment in the text area.
5. Click **Post**.

Editing comments

This section describes how to edit comments.

Related comment features:

- Comment notification delay
- Reply to comment
- Emoji
- Links
- Attachments
- Markdown

> **Note**
> - You can only edit comments that you have created.
> - If a comment is edited, all of the likes for that comment are removed.

1. Click the comment **Edit** link.
2. Edit the comment content, including adding new attachments or removing existing attachments.

   When you remove an attachment, the attachment is marked for removal, the green bar for the attachment is muted, and the **X** button is replaced with a **Restore** button .

   > **Tip**
   > Click the **Restore** button  to the right of the attachment to keep the attachment on the comment.
   >
   > Attachments cannot be restored after the comment edit has been saved. In this case you must edit the comment again and add the attachment to it.

3. Click **Save** to save the edited comment and remove any attachments marked for removal. The comment timestamp is marked as *(edited)* to show that the comment has been changed. Swarm sends a notification to everyone involved in the review, including the review author and the reviewers, but not the editor of the comment.

> **Note**
> Swarm does not provide a mechanism to see older versions of edited comments.

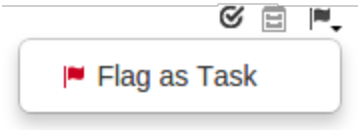
Tasks

Flagging review comments as *tasks* is a lightweight workflow within a review that helps authors and reviewers prioritize review feedback. Any comment on a code review can be flagged as a task, indicating to the code review's author that the described issue needs to be addressed, and that the review is unlikely to be approved without a fix.

> **Note**
> Changelist and Job comments cannot be flagged as tasks.

Flag a comment as a task

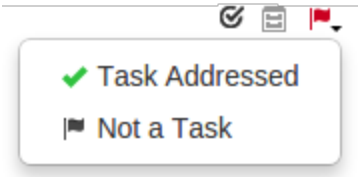
To flag a comment as a task, select the **Flag as Task** checkbox when posting a comment, or click the **Flag** button in the upper right of an existing comment and select **Flag as Task** in the drop-down menu.



> **Note**
> If you do not have permission to archive comments, you do not have permission to flag comments as tasks. Anonymous users never have permission to archive comments, and can only view current task states.

Set a task to Task Addressed or Not a Task

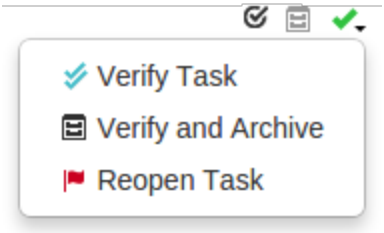
Once a comment is flagged as a task, it is considered to be an *open task*. Click the **Red Flag** button to display a drop-down menu with the following options:



- **Task Addressed**: usually used by the author of the review to indicate that the issue has been fixed.
- **Not a Task**: used to correct comments that have been flagged as tasks by mistake.

Verify a task, verify and archive a task, or reopen a task

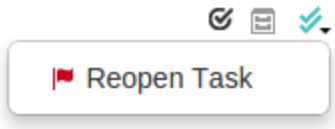
A comment with a green check indicates that the task has been addressed. Click the **Green Check** button to display a drop-down menu with the following options:



- **Verify Task**: usually used by the author of the comment, or another reviewer, after confirming that the issue is fixed.
- **Verify and Archive**: used to both indicate that the issue has been fixed, and to archive the comment so that it is hidden from view. Archived tasks, whether they are open, addressed, or verified, are not included in the task counts for the code review.
- **Reopen Task**: used if the issue needs further work after it has been marked as addressed.

Reopen a task

A comment with a blue double-check indicates that the task has been verified. Click the **Blue Double-check** to display a drop-down menu with the following option:

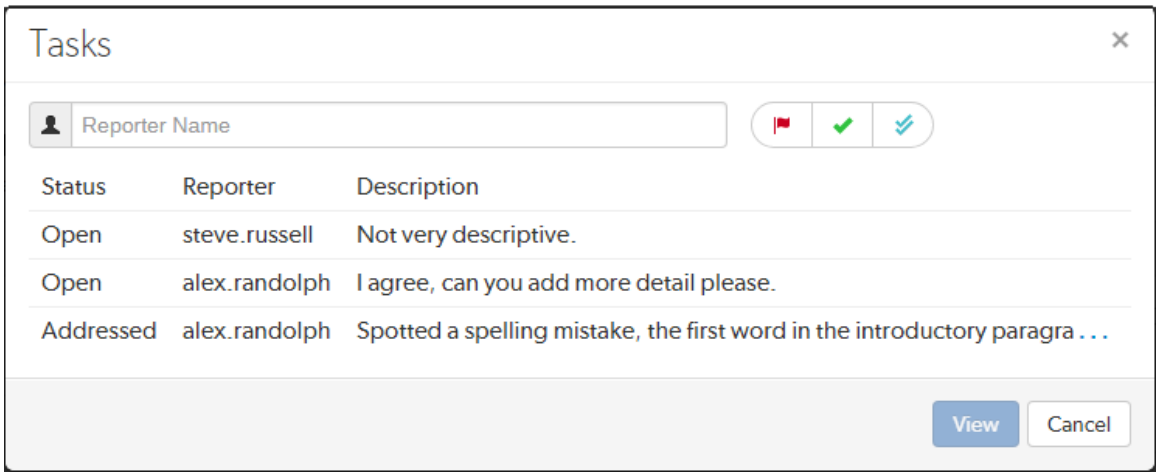


- **Reopen Task**: used if the issue needs further work post-verification, or if verification was made by mistake.

Task list

A summary of the number and status of comments flagged as tasks is displayed below the code review description. Archived comments that are flagged as tasks are not included in the summary.

Click the **Task list** button to display a dialog listing all tasks associated with the review:



Within the **Tasks** dialog, you can filter the tasks by the reporter (the userid of the user who created the task), and/or by task state:

- Click the **Red Flag** button to display only open tasks (comments that need to be addressed).
- Click the **Green Check Mark** button to display only addressed tasks (comments that have been addressed).
- Click the **Blue Double-check Mark** button to display only verified tasks (comments that have been addressed and verified).

> **Note**
> Archived tasks do not appear in the Tasks dialog.

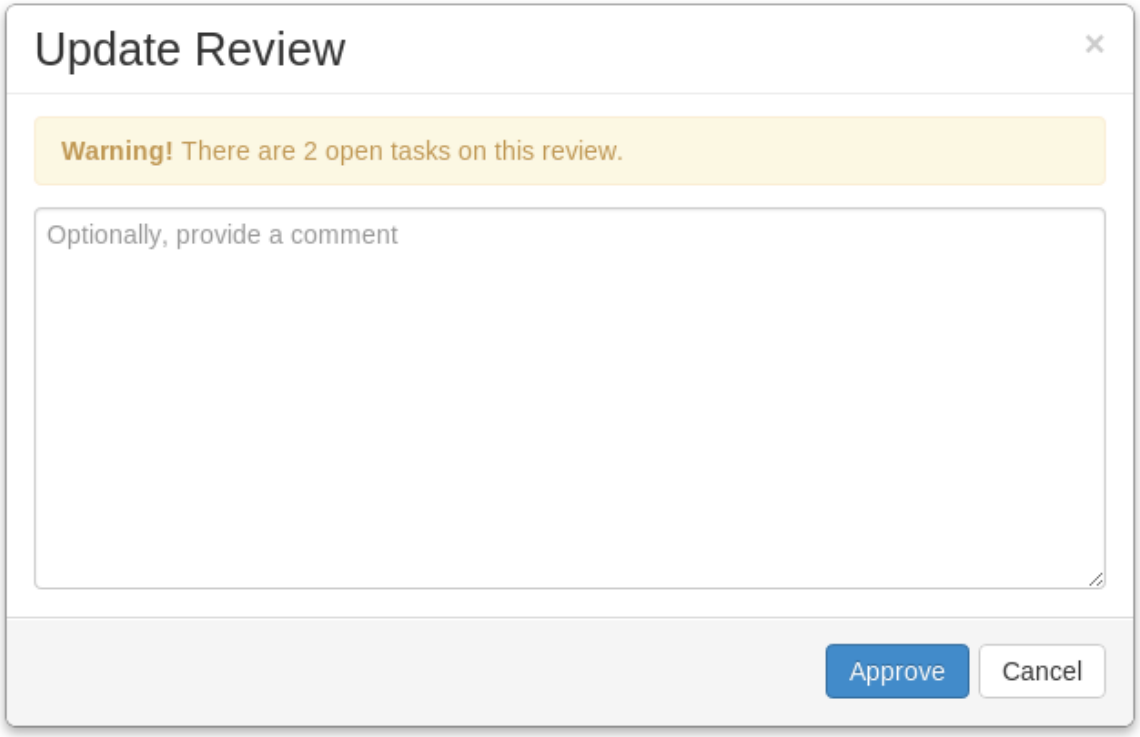
To view a particular task, select the task in the **Tasks** dialog, and click the **View** button. Alternately you can double-click on the task in the **Tasks** dialog. The **Tasks** dialog closes and Swarm adjusts the review display so that the chosen task is in view.

Approve a Review with open tasks

Flagging a comment as a task provides a visual indication that there is an identified issue that needs to be addressed before the review can be approved, see "Set a task to Task Addressed or Not a Task" on page 270.

> **Note**
> If Swarm is configured to prevent approval of reviews with open tasks and a review has open tasks, the **Approve**, and **Approve and Commit** options will not be available for the review. This option is configured by an administrator, see "Prevent Approve for reviews with open tasks" on page 553.
>
> To approve, or approve and commit a review with open tasks, you must address the tasks first and then set them to **Task Addressed**, or **Not a Task**, see "Set a task to Task Addressed or Not a Task" on page 270 for details.

If you **Approve** or, **Approve and Commit** a review that has open tasks with Swarm, a warning message is displayed in the **Update Review** dialog. The warning is only advisory, either click **Cancel** and address the open tasks or click **Approve** to approve the review. Archived open tasks will not trigger the warning message.



Comment features

Comment notification delay

By default, comment notifications are delayed to allow reviewers to add or edit comments as they progress through a review without sending a notification for each individual comment on the review. Comment notifications are rolled up into a single notification and sent either manually by the reviewer, or automatically after the notification delay time has been exceeded.

The delay countdown is reset each time the reviewer adds or edits a comment on the review, by default the notification delay time is set to 30 minutes.

- If you are commenting on more than one review, each of the reviews that you are commenting on has its own notification delay countdown that only applies to the comments that *you* make on *that* review.
- If another reviewer is making comments on the same review as you, *that* reviewer has their own notification delay timer for *that* review.

- If you manually send a delayed comment notification, the notification will only contain the comments that *you* made on *that* review.

Tip
The comment notification delay does not delay the posting of the comments, only the comment notification is delayed.

Note
Comment notifications are only delayed for comments on reviews. Comments on commits or jobs produce notifications immediately.

Note
The notification delay time is a global configuration setting configured by the Swarm administrator, see Comment notification delay.

Manually send the comment notification immediately:

1. Add or edit your comment as normal.
2. Click the **Post and Notify (X)** button to the right of the comment box.
   Where **(X)** is the number of delayed comment notifications in the queue waiting to be sent, this number does not include the current comment you are working on.

   Tip
   - If you forget to click the **Post and Notify (X)** button, click the **Send All Notifications** button below your avatar on the review page to send all of the notification for the review immediately.
   - Only the comments that you have made on this review are rolled up into the notification that is sent when you click **Post and Notify (X)** or **Send All Notifications**.

Reply to comments

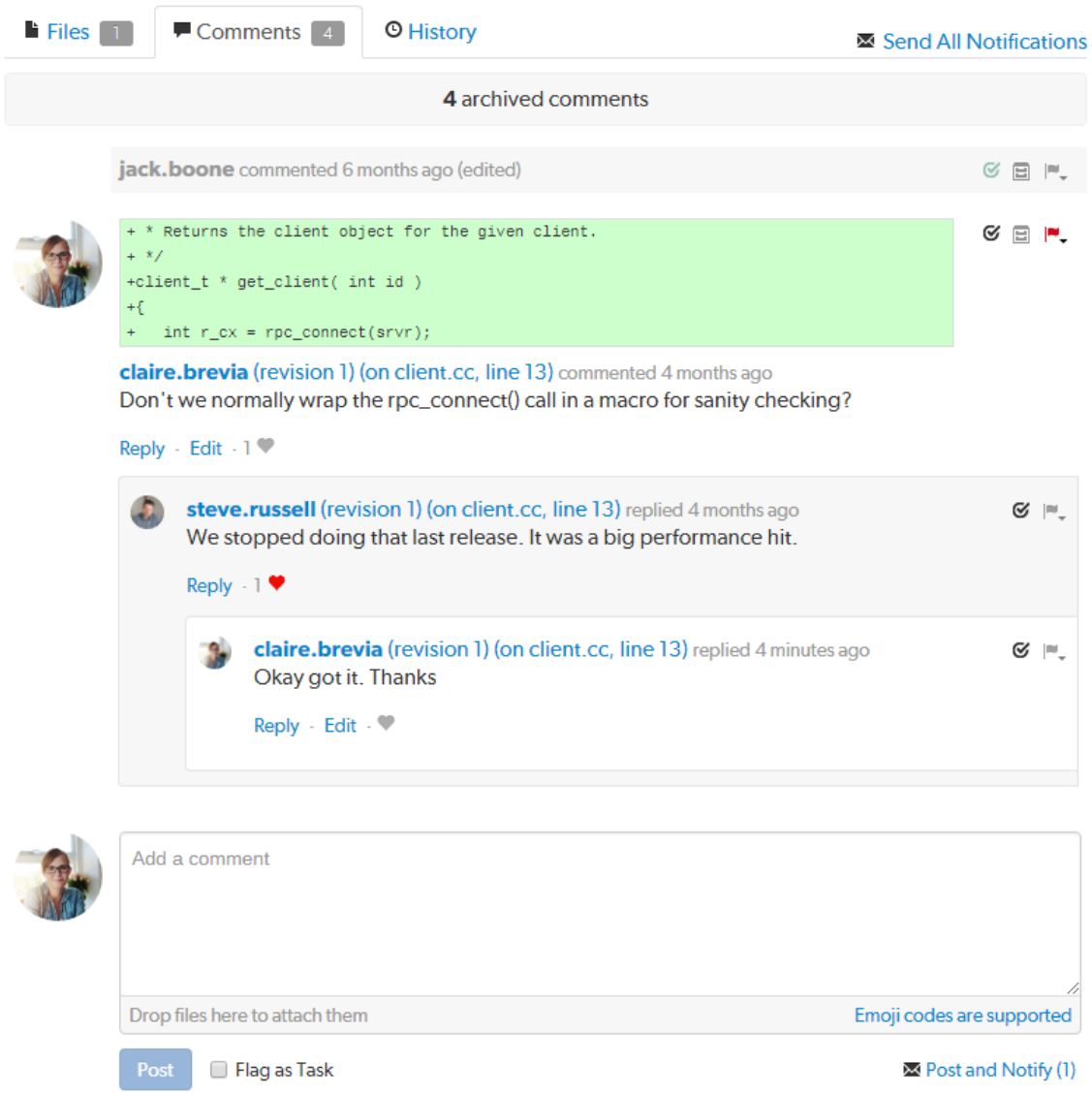
By default, you can reply to comments, replies are displayed in a thread below the parent comment. Comment thread depth is set to 4 by default, this means you can have up to 4 levels of replies for a parent comment. The **Reply** link is not displayed for replies at or above the maximum thread depth set for Swarm. If the parent comment is archived, replies are archived with the parent, see "Archiving comments" on page 280.

Note
Comment replies can be disabled, and the thread depth can be increased or reduced by a Swarm administrator, see "Comment threading" on page 475.

> **Tip**
> If the thread depth is reduced by a Swarm administrator, earlier replies at a deeper level will continue to be displayed but you cannot reply to them.

To reply to a comment:

1. Click on the **Reply** link below the comment you are replying to.
2. Add your comment in the text area.
3. Click **Post**.



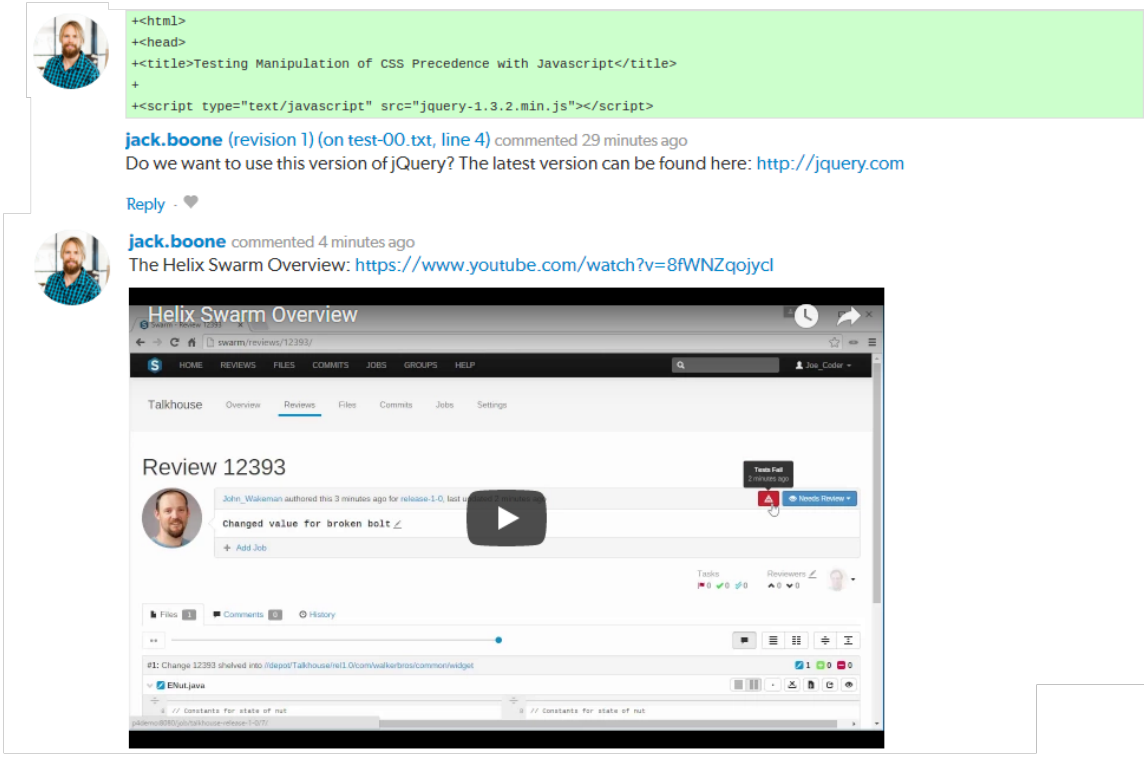
Emoji

Swarm comments support Emoji short-hand; when you save a comment, emoticon text like `:smile:` is displayed as: 😄. Hover your mouse over an Emoji emoticon to see a tooltip displaying the text short-hand for the emoticon:

For more information on Emoji, see Emoji on Wikipedia. Emoji emoticons are listed in the Emoji Cheat Sheet.

Links in comments

Whenever you include a URL in a comment, it is automatically made into a link. If the link points to an image, or a YouTube video, that resource is displayed at the end of the comment.



Comment attachments

Arbitrary files can be attached to comments. This is useful for sharing documents that are helpful in code reviews, such as screenshots of error conditions, reference code, etc.

Note
Swarm must be configured to enable comment attachments. Once the configuration is complete, the comment area will include the following text **Drop files here to attach them**.

To attach a file to an open comment:

1. Drag the file from your file browser and drop it on the comment area.

   Multiple files can be attached to a comment, either one at a time, or by dragging a group of files. Folders cannot be attached to comments.
2. The upload to Swarm starts immediately and progress is displayed below the comment.
   - **Blue Progress Bar:** the upload to Swarm is in progress.
   - **Green File Bar:** upload complete.
3. If you attach a file to the comment by mistake, you can remove it by clicking the **X** button on the **Green File Bar**.
4. Click **Post**.

The filename and file size is displayed in the comment for each attached file, click on the filename to view the file.

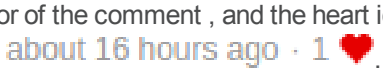
When comments are viewed in the **Comments** tab, image previews are displayed for the attached images.

Note
Image previews are only displayed in the **Comments** tab, they are not displayed when a comment is viewed in the **Files** tab.

Liking comments

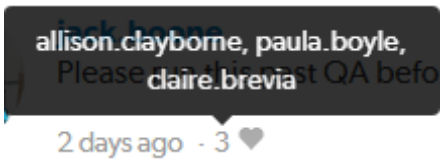
As an authenticated user, you can *like* a comment by clicking the muted heart icon beneath the comment 

.

When you like a comment, a notification is sent to the author of the comment , and the heart icon changes to red to indicate that you have liked the comment 

. The number of likes the comment has is displayed next to the heart icon. If a comment is edited, all of the likes for that comment are removed.

Hover your mouse pointer over the number of likes to display the usernames of everyone that has liked the comment:

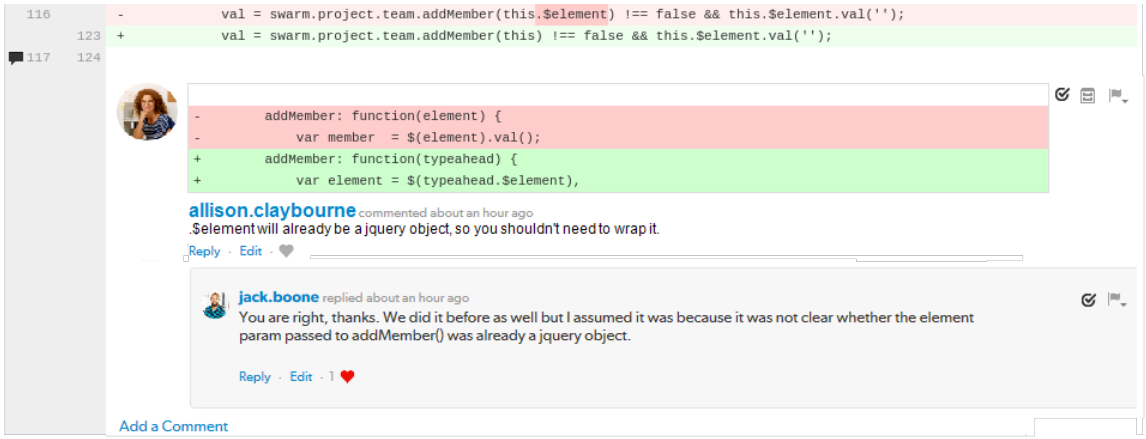


Click the heart icon again to *unlike* a comment.

> **Note**
> You cannot like/unlike a comment that has been archived.

Comment context

When comments are added to files in a review, on lines that have been changed, Swarm records several lines of context before the line receiving the comment. This helps makes sense of the comments should later changes remove those lines.



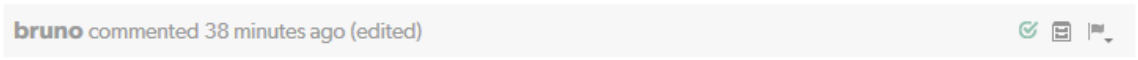
Each comment associated with that line has a record of the context, but only the first comment displays that context.

Mark comments as read

When you mark a comment as read, the comment is rolled up into a single line to save space and make it easier for you to find comments you have not read. The read flag is remembered independently for each user.

Mark a single comment as read:

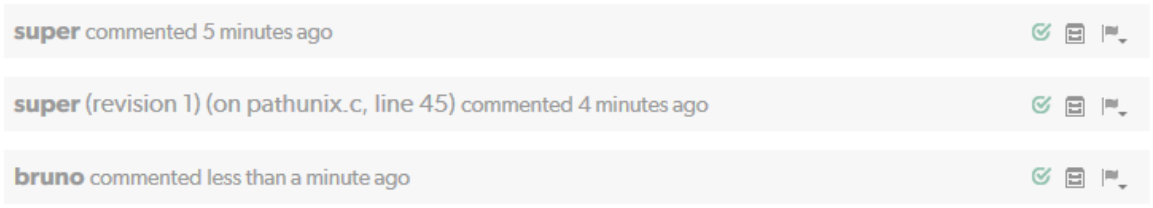
1. Click the **Mark comment as read** button at the top right of a posted comment.
2. The comment is rolled up into a single line to save space.



> **Note**
> Marking a parent comment as read will not mark the child comments as read.

Mark all of the comments on a review as read:

1. Click the **Mark all comments read** button in the review heading.
2. All of the comments in the review are rolled up into single lines to save space.



> **Tip**
> **Mark all comments read** will mark all the comments as read including the description comments.

Mark comments as unread

Marking a comment as unread expands the comment so that you can view the comment content. If a comment is marked as read and the comment changes, Swarm will automatically clear the read flag so that you can see that the comment has changed. Changes that automatically mark a comment as unread are:

- Comment text is edited
- Comment attachments are added or removed
- Comment is marked as a task

- Task is reopened
- Comment or task is unarchived

Mark a single comment as unread:

1. Click the **Mark comment as unread** button to the right of the comment.
2. The comment content is expanded.

> **Note**
> Marking a parent comment as unread will not mark the child comments as unread.

Mark all of the comments on a review as unread:

1. Click the **Mark all comments unread** button in the review heading.
2. The content of all of the comments in the review are expanded.

> **Tip**
> **Mark all comments unread** will expand the content of all the comments including the description comments.

Archiving comments

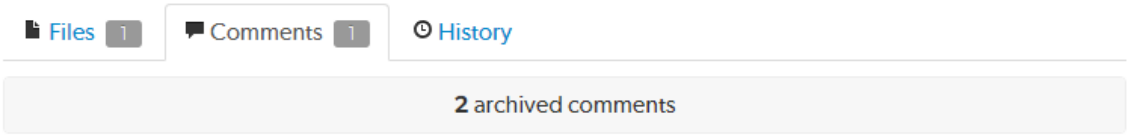
As a code review progresses, comments made on earlier versions of a file might become less useful. Archiving these comments tidies up the comment view and makes it easier to find the more important comments.

The **Archive** button is only available for top level comments, if a comment has replies they are archived with the parent comment.

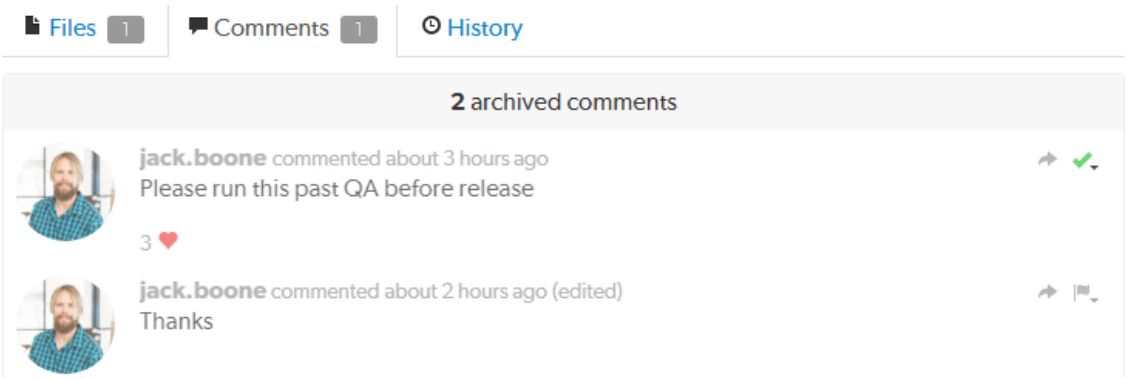
To archive a comment:

- Click the *Archive* button at the top right of a posted comment.

Archived comments are hidden from view, the number of archived comments that exist for the review is displayed in the **archived comments** button at the top of the **Comments** tab.



Click on the **archived comments** button to toggle the archive comment display on and off.

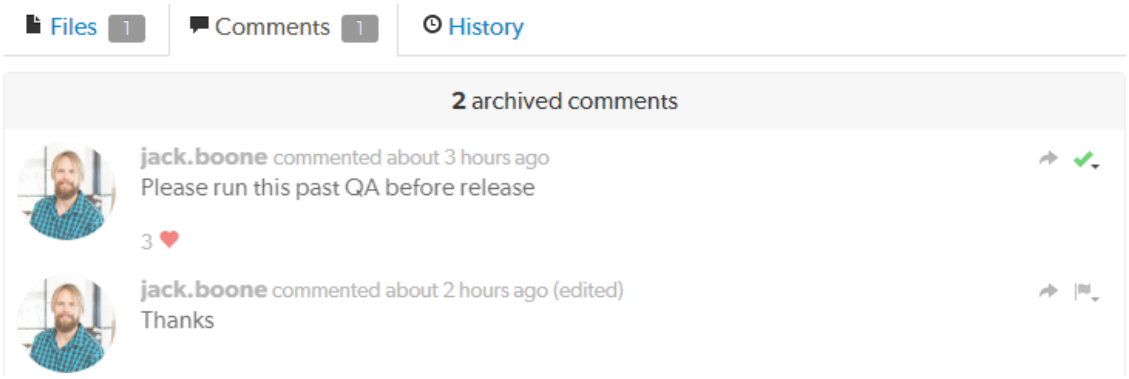


Restore comments

Archived comments can be restored. If archived comments have replies, the replies are also restored.

To restore archived comments:

1. Click the **archived comments** button at the top of the **Comments** tab to display all of the archived comments.



2. Find the comment you want to restore.
3. Click the **Restore** button in the top right of the comment to restore it.

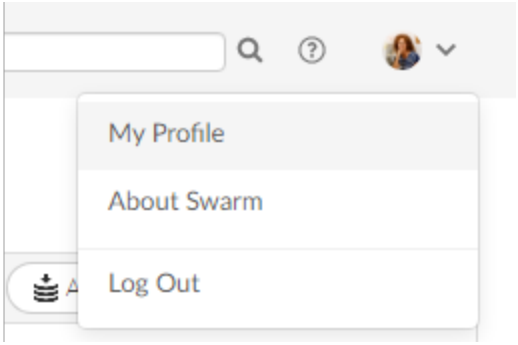
Users

Swarm's users are based on the users configured in the Helix server.

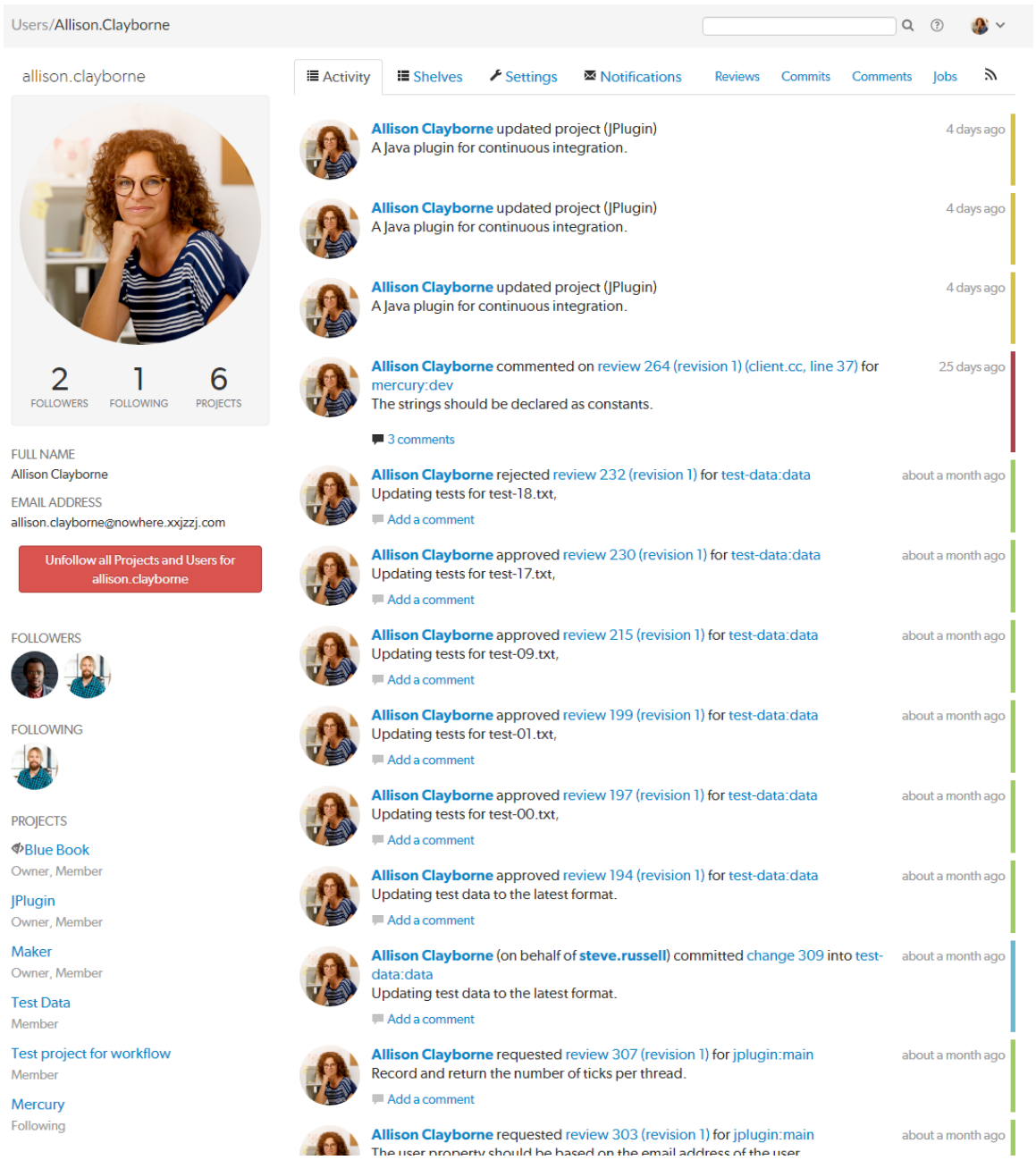
- "Viewing your user profile" on the facing page
- "Viewing another user's profile" on page 289
- "Viewing another user's profile when you have admin or super user privileges" on page 290

Viewing your user profile

Display your own user profile when you are logged in by clicking on your userid in the header and selecting **My Profile**.



The user profile page displays your avatar, your full name, your email address, an **Unfollow all projects and users for your-username** button, a list of users following your activity, the users or projects that you are following, and the list of projects of which you are an owner or member, including an *eye* icon ( ) beside any private projects you belong to.



Unfollow all projects and users for yourself

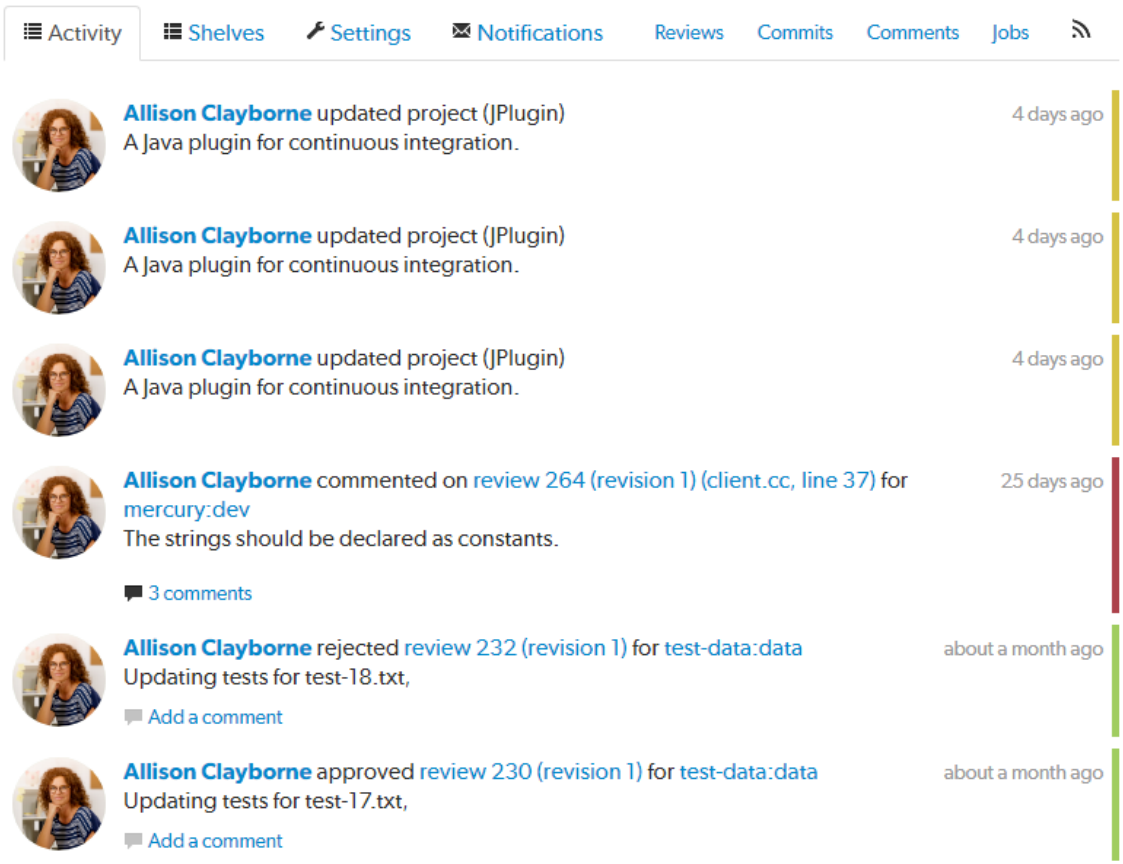
When you are viewing your own user profile, you can unfollow all projects and users that you are following.

Note
This action cannot be undone.

1. Click the **Unfollow all projects and users for *your-username*** button.
2. Click **OK** when the confirmation dialog is displayed to complete the unfollow action.
3. You will no longer be following any projects or users.

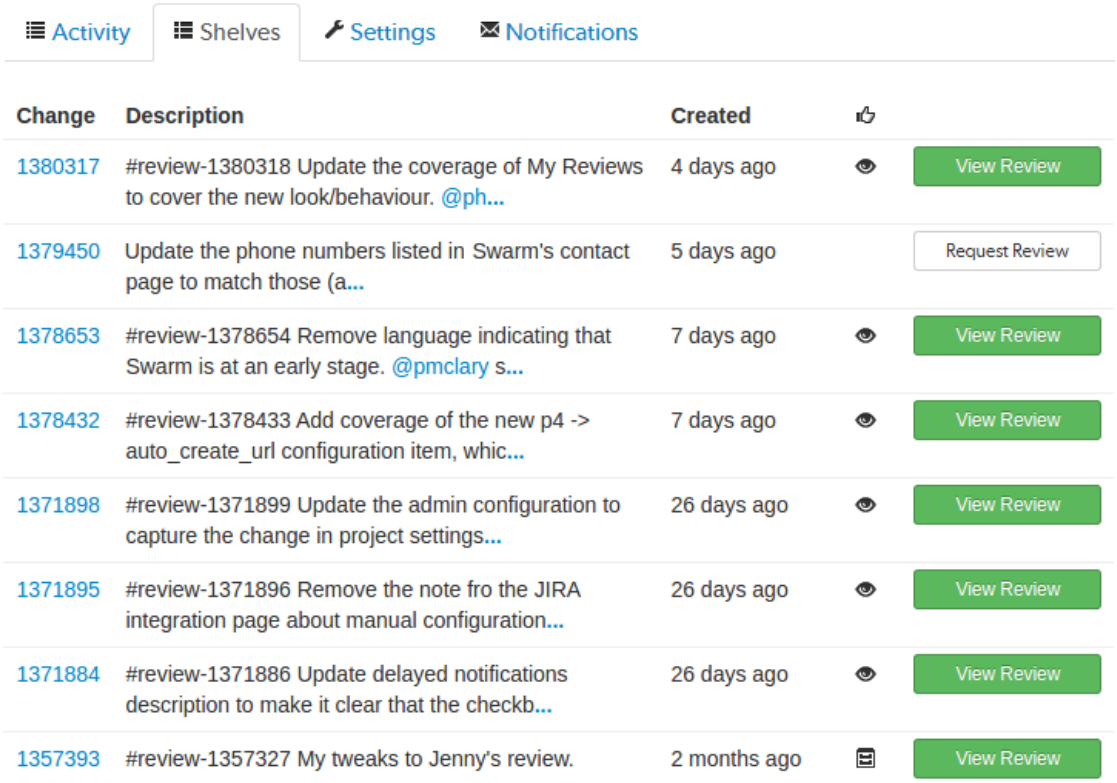
Activity tab

Click the **Activity** tab to display the activity stream for events you have created.



Shelves tab

Click the **Shelves** tab to display a list of your shelved changelists.



A *shelved* changelist is a pending changelist that has a copy of one or more files from within the changelist stored on the server. Shelved files are not versioned. If you update the shelved files, the update replaces any existing files on the changelist's shelf.

> **Note**
> Swarm can use multiple shelved changes to record the history of reviews. See "Internal representation" on page 339 for details.

Swarm uses shelved changelists as the basis of its code review feature. However, not all shelved changelists are reviews. Users may shelve files for other reasons, including ensuring that the Helix server has a copy of work in progress, or as a way to move temporary work from one workspace to another.

Click **Request Review** to start a Swarm review for any shelved changelist that is not already involved in a review.

Click **View Review** to view the Swarm review associated with shelved changelists when a review has already started.

> **Important**
> By default, when you delete files from a shelved changelist, the files are not removed from the associated review.

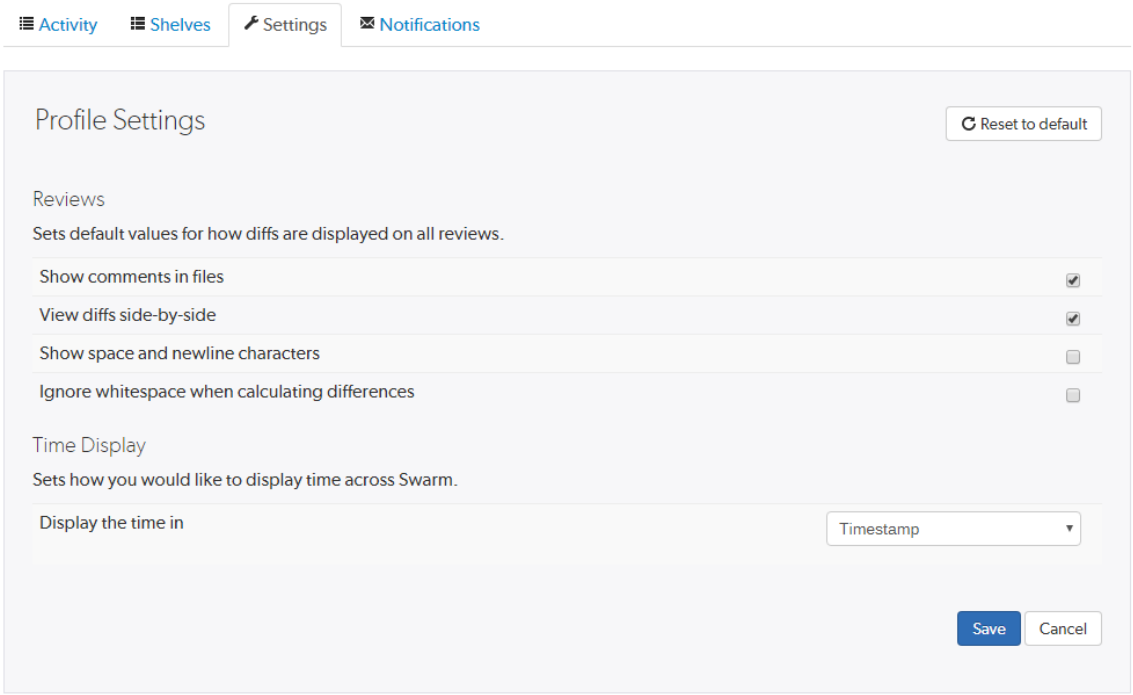
However, Swarm can be configured to remove files from a review when they are deleted from an associated shelf, see "Process shelf file delete when" on page 555. To delete a shelved changelist without removing all of the shelved files from the associated review, see "Deleting shelves" on page 369.

Do not use the Swarm user that is configured in the Swarm configuration file when deleting shelves, or deleting files from shelves. The Swarm logic processes the *shelve-delete* trigger event, if the event is invoked by the Swarm user it is rejected. The delete operations will fail.

Settings tab

The **Settings** tab allows you to configure how diffs are initially displayed when you view them, and how time is displayed by Swarm.

Note
Defaults for these options are configured by the Swarm administrator, see Users. The defaults set by the Swarm administrator are used until you change your settings.



Toggle the **Reviews** settings to control how diffs are initially displayed on changelists and reviews by default.

Choose the **Time Display** settings to configure how time is displayed by Swarm. You can configure Swarm to display timestamps based on your local machine browser time, or by how long ago events happened.

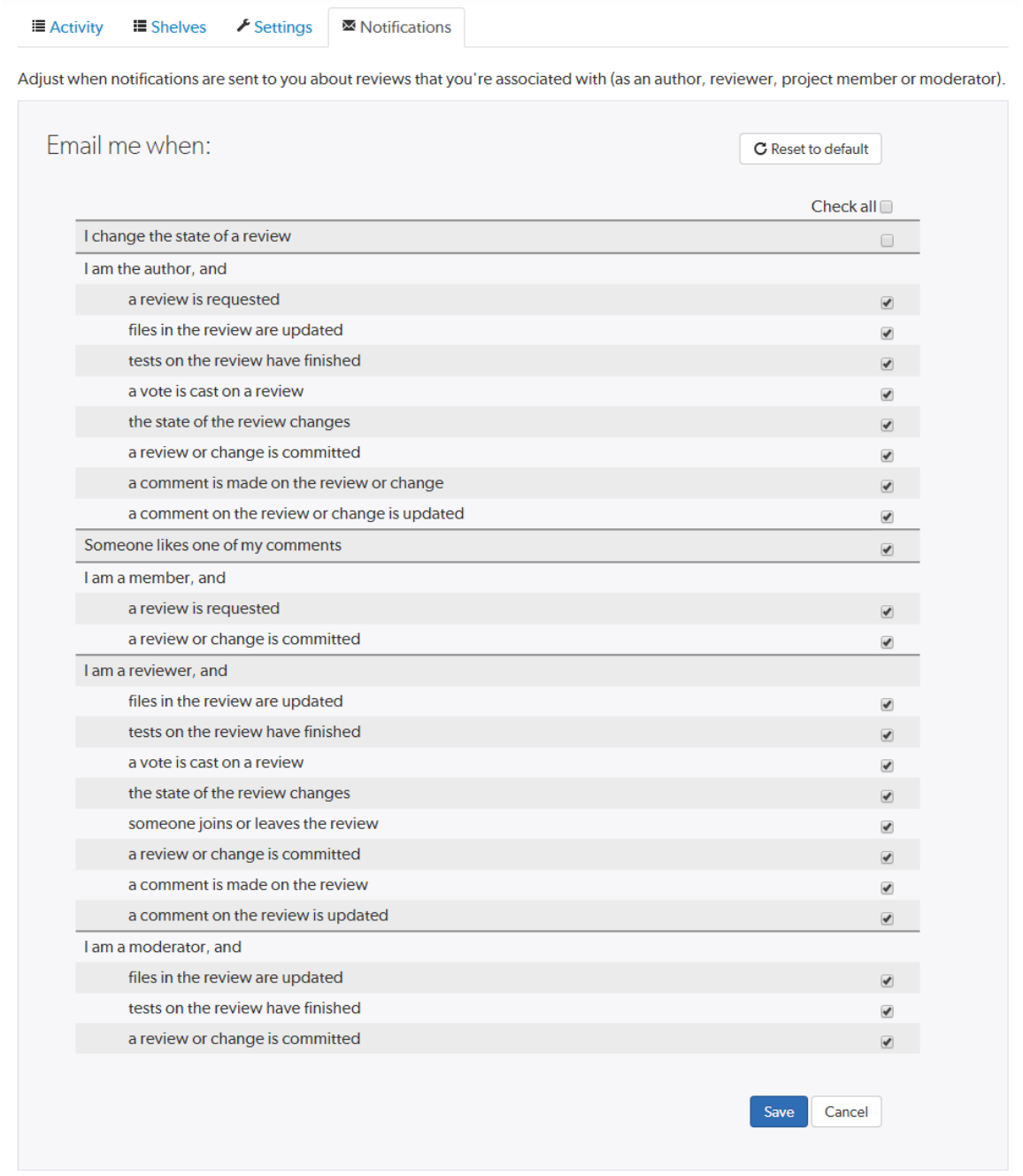
> **Tip**
> If timestamp is selected, the date format used by Swarm matches the date format configuration of the local machine browser.

Click the **Save** button to save the settings. Click **Reset to default** to reset the settings back to system defaults.

Notifications tab

The **Notifications** tab allows you to configure which notifications you receive when events occur within Swarm. This allows you to limit the number of emails you receive to just those you are interested in. The settings apply across all projects.

> **Note**
> Defaults for these options are configured by the Swarm administrator. Your Swarm administrator may force some of these options to on or off, see "Global settings" on page 529.



Toggle notifications for each event on or off to control whether you receive an email when that event occurs.

Click the **Save** button to save the settings. Click **Reset to default** to reset the settings back to system defaults.

Viewing another user's profile

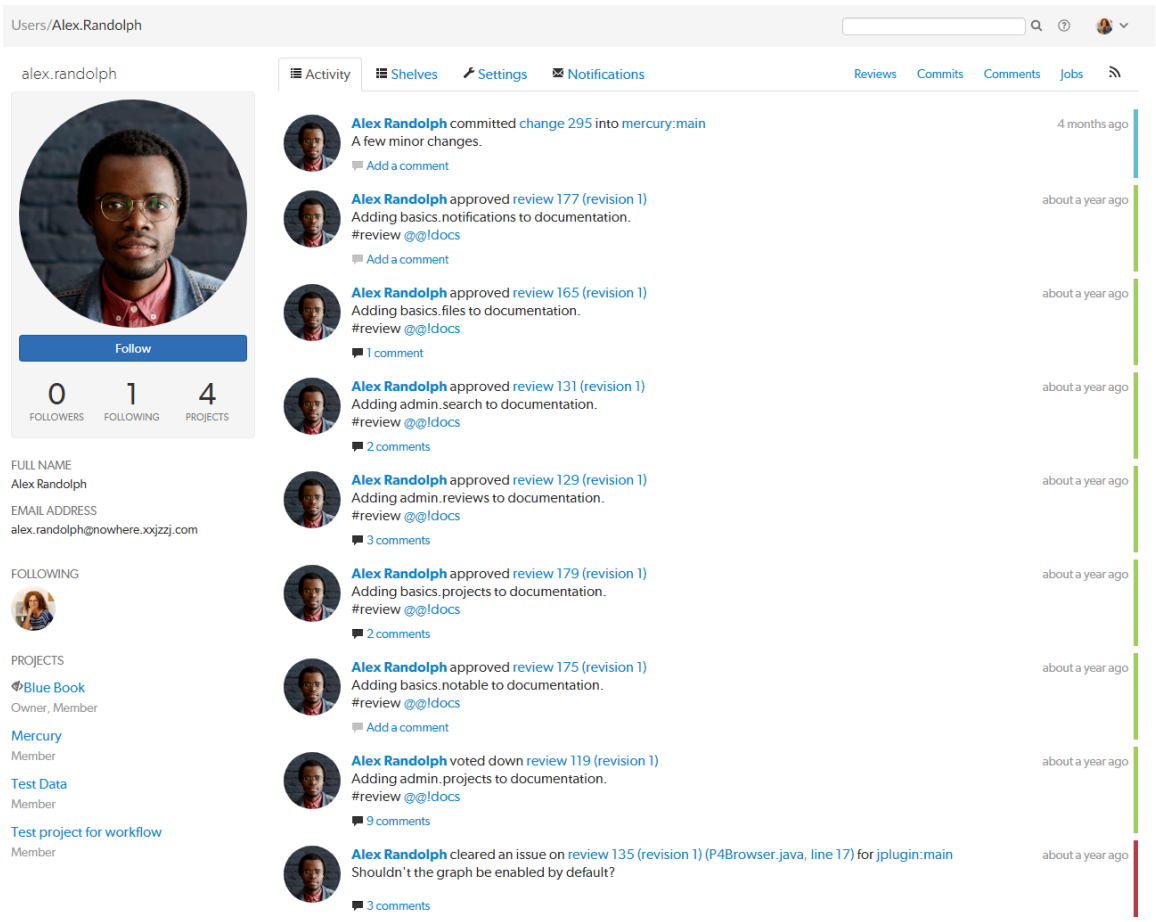
View the profile pages of other users displayed anywhere in Swarm by clicking on their avatar, *userid*, or by visiting the URL: **`https://myswarm.url/users/userid`**.

> **Note**
> When viewing another user's profile page you can see the user's **Settings** tab but you cannot make changes to it.

> **Note**
> Currently, Swarm does not provide an overall list of users.

Follow or unfollow the user

- **If you are not following the user:** click the **Follow** button below the user's avatar to start following the user.
- **If you are following the user:** click the **Unfollow** button below the user's avatar to stop following the user.



Viewing another user's profile when you have admin or super user privileges

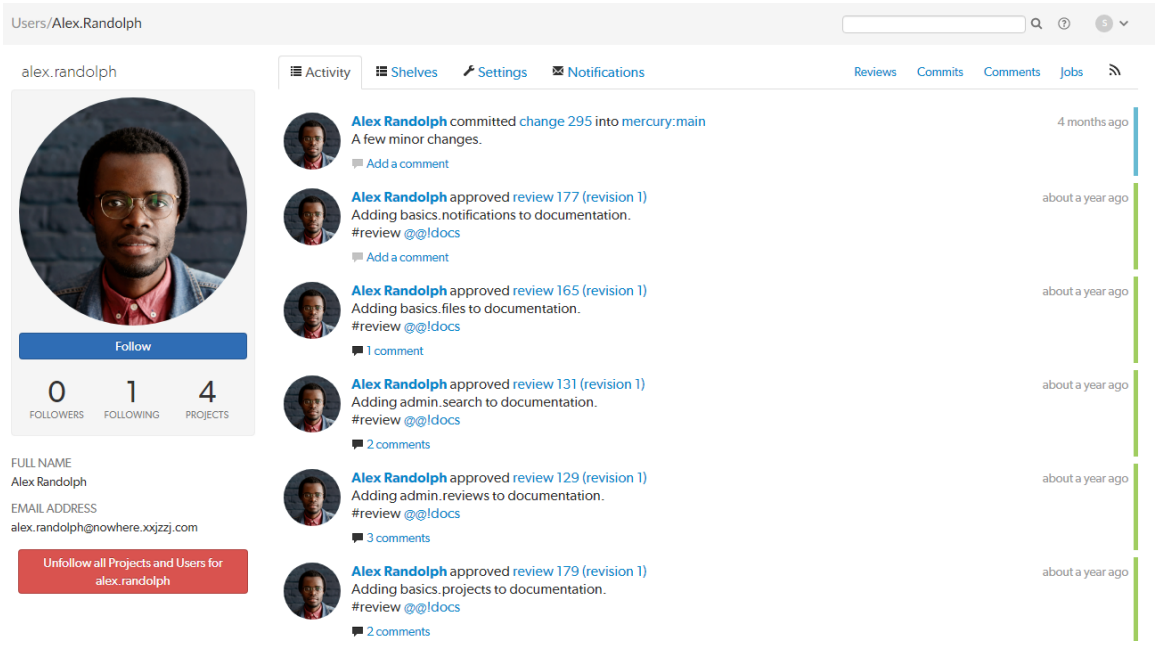
View the profile pages of other users displayed anywhere in Swarm by clicking on their avatar, *userid*, or by visiting the URL: **`https://myswarm.url/users/userid`**.

Unfollow all projects and users for another user

When a user has been removed from the Helix server but they are still following projects and users, it is useful to be able to remove all of their follows. This helps to keep the project and user follower lists up to date.

> **Warning**
> This action cannot be undone.

1. Click the **Unfollow all projects and users for *username*** button located below the user's email address.
2. Click **OK** when the confirmation dialog is displayed to complete the unfollow action.
3. The user will no longer be following any projects or users.



Groups

Groups are a feature of Helix server that makes it easier to manage permissions for users.

It is important to be aware of certain aspects of Helix server groups:

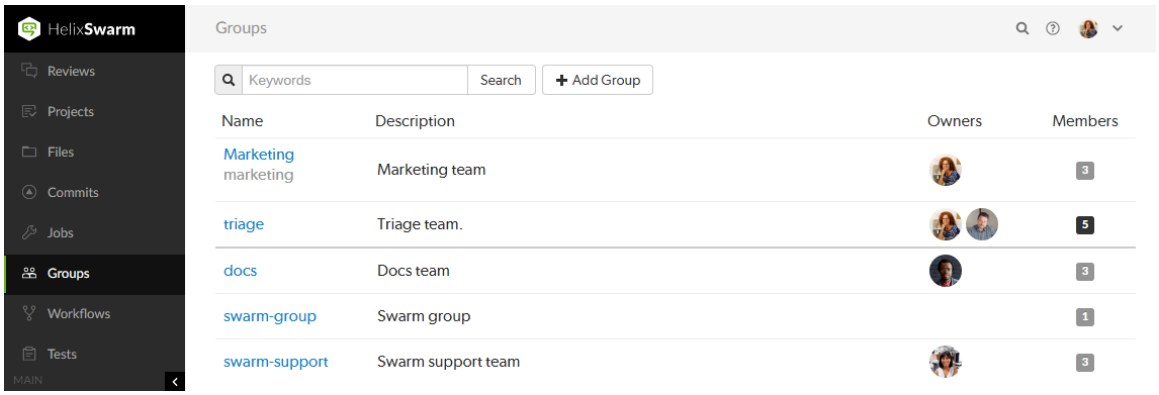
- Groups can have both users and sub-groups. A user that is a member of a sub-group is also a member of the parent group.
- Groups have owners, but owners are not members of their groups. A group owner can be added as a member of a group.
- Groups can be created by users with *super* privileges in Helix server (`p4d`). If p4d is at version 2012.1 or newer, users with *admin* privileges can also add groups.
- Groups can be edited by their owners, or by users with *super* privileges in Helix server.
- Groups have a separate namespace from users; you can have a user named `fish` that is a member of a group named `fish`.
- Groups can be project moderators
- Groups can be reviewers, see Review display and @@mention for details.

- You can `@@mention` a group in a code review comment to ensure the mentioned group receives notifications of code review events. This also creates a link that displays the group's details when clicked. See @@mention for details.
- Groups cannot be project owners.

More information on adding, editing, and removing groups is included in the "Groups" on page 393 chapter.

Listing groups

Begin browsing groups by clicking **Groups** in the menu.

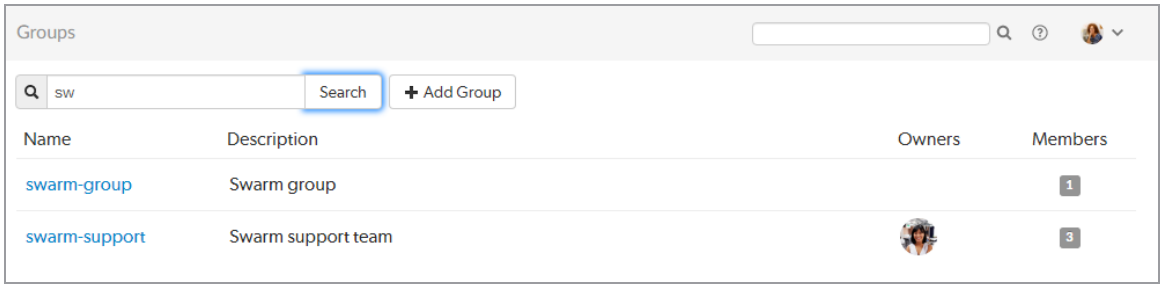


For each group, the group's name, description, list of owner avatars, and a membership count (which has a darker background when you are a member). Click on a group name to display details for that group.

The groups listing is sorted by multiple criteria:

- Groups that you own, or belong to, are listed first. A thicker row border indicates where your group ownership/membership ends; you are not an owner/member of any group below the thicker border.
- Groups are then sorted by name.

You can search available groups by entering some text in the **Search** field and clicking **Search**. Any group names or descriptions that match the entered text are displayed.



If you have *super* privileges in Helix server (P4D), or have admin privileges in P4D version 2012.1 or newer, Swarm displays the **+ Add Group** button. Click **+ Add Group** to add a new group.

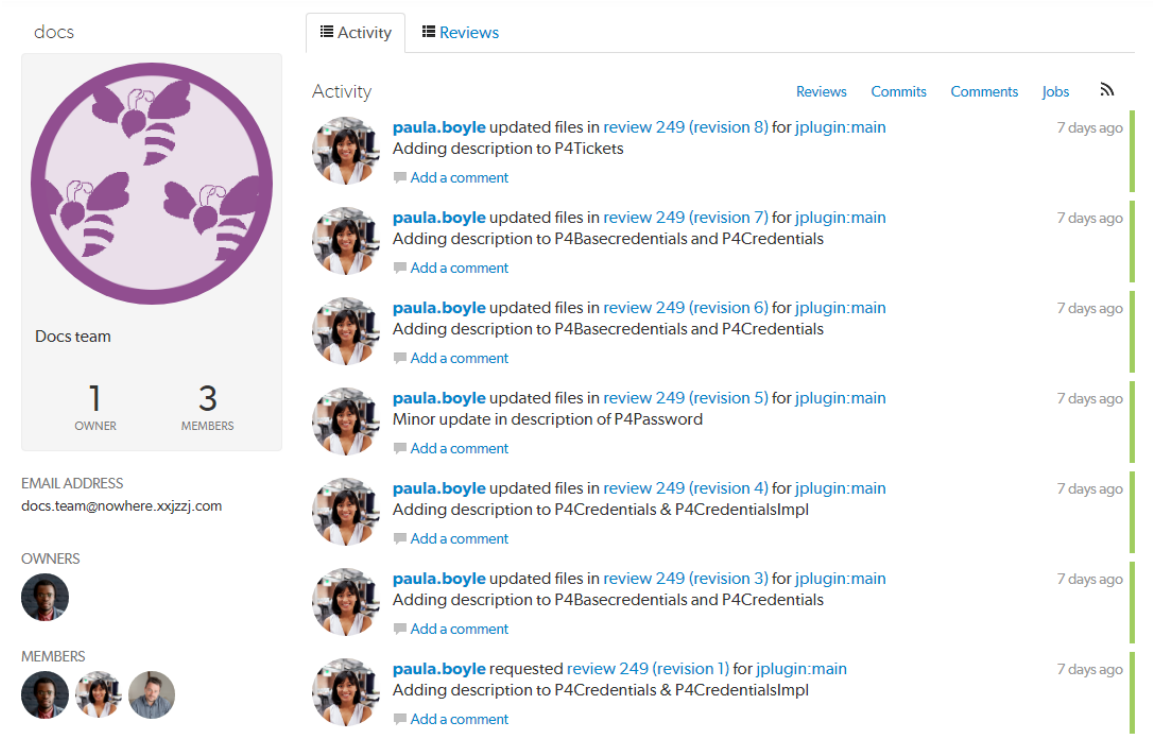
> **Note**
> If your Helix server has a large number of groups, the time required to display the list of groups might be notable.

Viewing a group

View a specific group by clicking on a linked group name, or by visiting the URL: **`https://myswarm.url/groups/group-id`**.

Activity page

The group **Activity** page shows the activity stream for events generated group members to the right of the group sidebar.



The following information is displayed in the group sidebar:

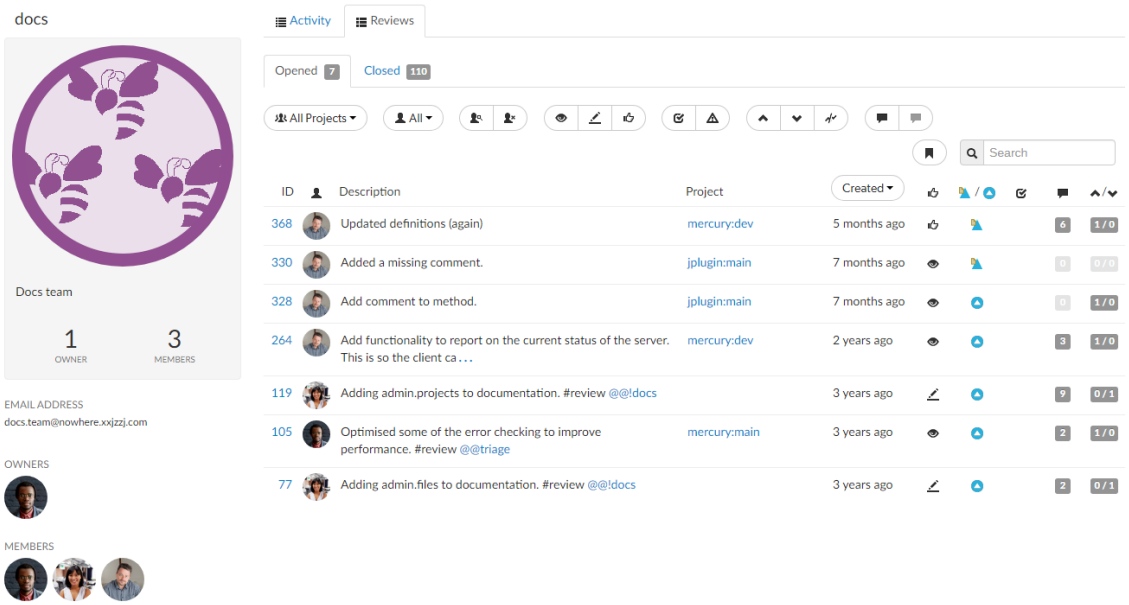
- Group name
- Group avatar, if it has one
- Group description, if it has one
- Count of the number of owners and members of the group
- Group email address, if it has one

- Avatars of the group owners
- Avatars of the group members

Reviews page

Click the **Reviews** link in the group toolbar to display a list of reviews associated with the group.

The group **Reviews** page shows a list of code reviews authored by any of the group members. It lists code reviews that are in progress, and code reviews that are complete.



The following information is displayed in the group sidebar:

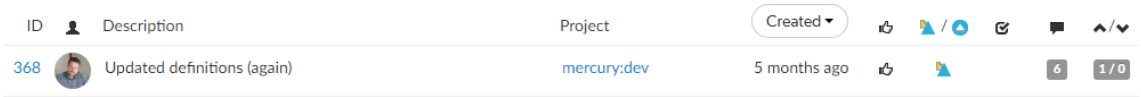
- Group name
- Group avatar, if it has one
- Group description, if it has one
- Count of the number of owners and members of the group
- Group email address, if it has one
- Avatars of the group owners
- Avatars of the group members

Reviews list

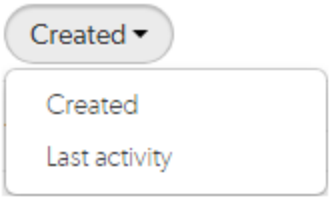
The **Opened** and **Closed** tabs display:

- **Opened tab:** displays a list of all code reviews that have started, are being reviewed, are awaiting revisions, or need to be committed.
- **Closed tab:** displays a list of all code reviews that have been approved and committed, rejected, or archived.

Each reviews list displays a summary for each review:



- The review id
- The avatar of the review author
- The review description
- The associated project name and branch
- **Created** or **Last activity** dropdown: click to change the order the reviews are displayed in, options are:
  - **Created:** Reviews sorted by when they were created
  - **Last activity:** Reviews sorted by when they were last updated



> **Note**
> - If the **Result order** button is not displayed reviews are sorted by when they were created.
> - The **Result order** button display is a global setting controlled by the Swarm administrator. See "Reviews filter" on page 551 for details.
> - When review results are older than 24 hours they are displayed in numerical order within each day.

- An icon indicating the current review state
- An icon indicating the review type. This can be Pre-commit or Post-commit.
- An icon indicating the test suite state for the review, either tests in progress, tests passed, or tests failed.
- A counter for the number of open (non-archived) comments that are associated with the review. Hover your mouse over the comment count to display a tooltip showing the number of archived

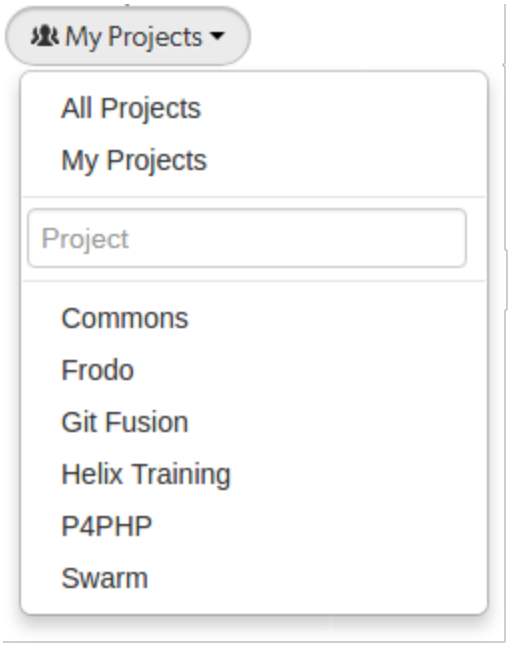
comments associated with the review.

- An indicator showing the number of up votes and down votes

Filtering reviews

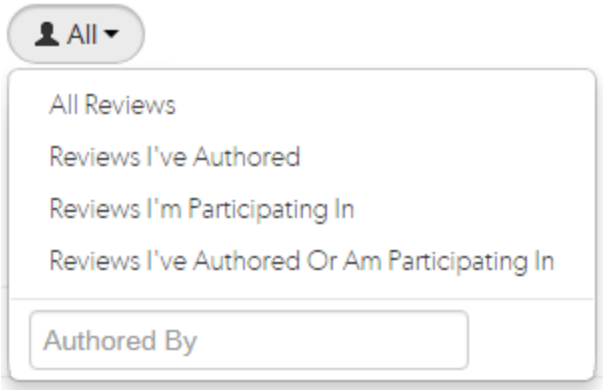
The following filtering options are available for code reviews:

- **Project:** a dropdown menu that lets you filter which reviews to display based on project:



  - **All Projects:** all reviews are displayed for all projects.
  - **My Projects:** all reviews for all of the projects you are participating in, as a member, owner, moderator, or follower.
  - **Project Search:** an auto-complete search field that allows you to choose one of the projects defined in Helix server. Once specified, only reviews for the selected project are displayed. Click the **X** button to remove a *projectid* after it has been specified.
  - **Select Project:** selecting a project name displays only reviews for that project.

    When you select one of the available options, the list of options updates to match the currently selected filter, and the **Projects** dropdown indicates the current filter: **All Projects**, **My Projects**, or ***projectid***.
- **Users:** a dropdown menu that lets you filter which reviews to display based on user involvement:



- **All Reviews:** displays all reviews.
- **Reviews I've Authored:** displays reviews that you have authored.
- **Reviews I'm Participating In:** displays reviews that you are a reviewer of, but not an author of.
- **Reviews I've Authored Or Am Participating In:** displays reviews that you have authored, or are a reviewer of.
- **Specific User:** An auto-complete search field that allows you to choose one of the user accounts defined in the Helix server. Once specified, only reviews authored by the user are displayed. Click the **X** button to remove a *userid* after it has been specified.

  When you select one of the available options, the list of options updates to match the currently selected filter, and the **Users** dropdown indicates the current filter: **All**, **Author**, **Participant**, or ***userid***.

- **Reviewer presence** (**Opened** tab only):

  - **Has Reviewers:** displays reviews that have one or more reviewers.
  - **No Reviewers:** displays reviews that have no reviewers.

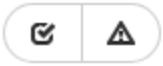
- **Review state** (**Opened** tab only):

  - **Needs review:** the review's changes need to be reviewed.
  - **Needs revision:** the review's changes have been reviewed, but further revisions are required before the review can be accepted.
  - **Approved:** the review's changes have been approved, and should be committed.

- **Review state** (**Closed** tab only):

- **Approved:** the review's changes have been approved and committed.
- **Rejected:** the review's changes have been rejected.
- **Archived:** the review's changes have been put aside.

- **Test status:**



  - **Tests pass:** when automated tests are enabled for the associated project, and the test suite execution succeeds, Swarm updates the review accordingly.
  - **Tests fail:** similar to the Tests pass state, except that the test suite execution has failed. Check with your test suite to determine why the tests failed.

- **Vote status:**

  - **Voted up:** I have voted the review up.
  - **Voted down:** I have voted the review down.
  - **Not voted:** I am a participant but have not voted on the review.

> **Note**
> Filters for voting only apply to reviews which you are a participant of. Commenting on or voting on a review will automatically add you as a participant. If you leave the review after commenting on it, then this review will not be included in the list.

- **Comment status:**

  - **Has comments:** I have commented on the review.
  - **Does not have comments:** I have not commented on the review.

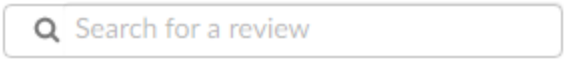
> **Note**
> Filters for commenting only apply to reviews which you are a participant of. Commenting on or voting on a review will automatically add you as a participant. If you leave the review after voting on it, then this review will not be included in the list.

- **Bookmark:** review filters can be remembered by bookmarking the page, and this icon acts as a reminder of that.

  Swarm updates the URL in your browser to reflect filtering options. This makes it easy to bookmark or share review list URLs. Swarm maintains the current filtering if you click on a review link and then use your browser **Back** button to return to the review list.

- **Search term:** where review descriptions match your search string.

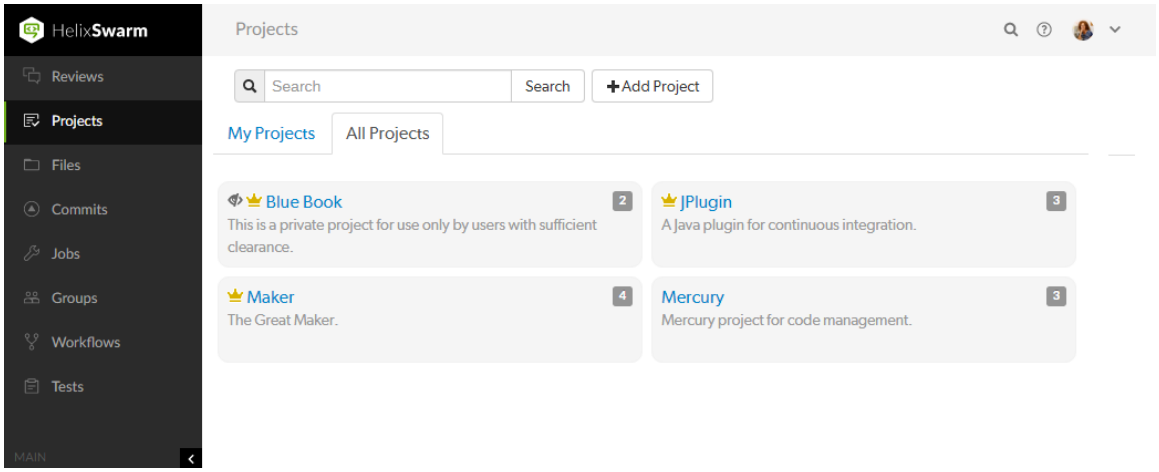
  

Projects

A Swarm project is made up of a group of Helix server users who are working together on one or more codelines within the Helix server. A project's definition includes one or more branches of code, and optionally a job filter, automated test integration, and automated deployment. This section provides an introduction to the interactions users have with projects. For details about managing projects, see "Projects" on page 399.

Listing projects

To view a list of projects, click **Projects** in the menu.



Logged-in users can choose which projects to display by clicking on the **My Projects** tab or the **All Projects** tab. Anonymous users only see the public projects, the **My Projects** tab is not available to anonymous users:

- The **My Projects:** tab lists all of the projects that you are an owner of or a member of.
- The **All Projects:** tab lists all of the available projects.

Tip

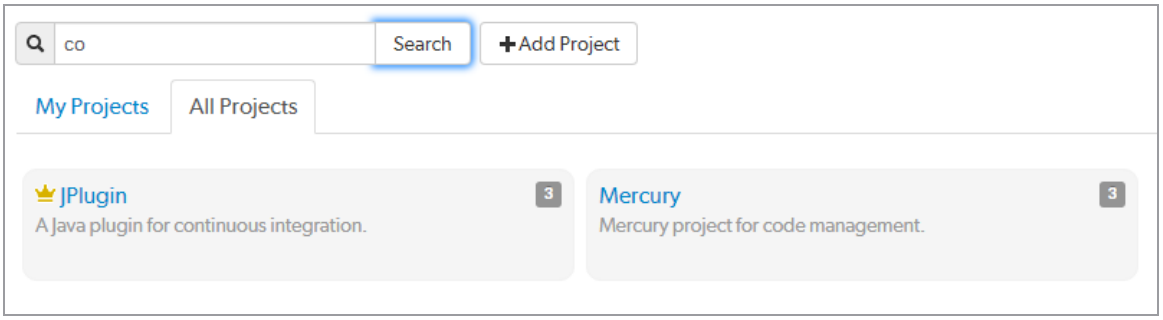
- The **Project Owner** icon is displayed for projects you own.
- The **Private Project** icon is displayed for private projects you are authorized to view. For details, see "Private projects" on page 306.
- The number in the icon to the right of the project name displays the total number of members for the project.

Search project names and project descriptions

You can search for Swarm project names and for the content of the project description from the projects page:

1. Enter text in the search box.
2. Click **Search**.

   Any project names, and descriptions that match the search text are displayed in the project tab:



Tip
Switch to the other project tab to display search results for that tab.

Viewing a project

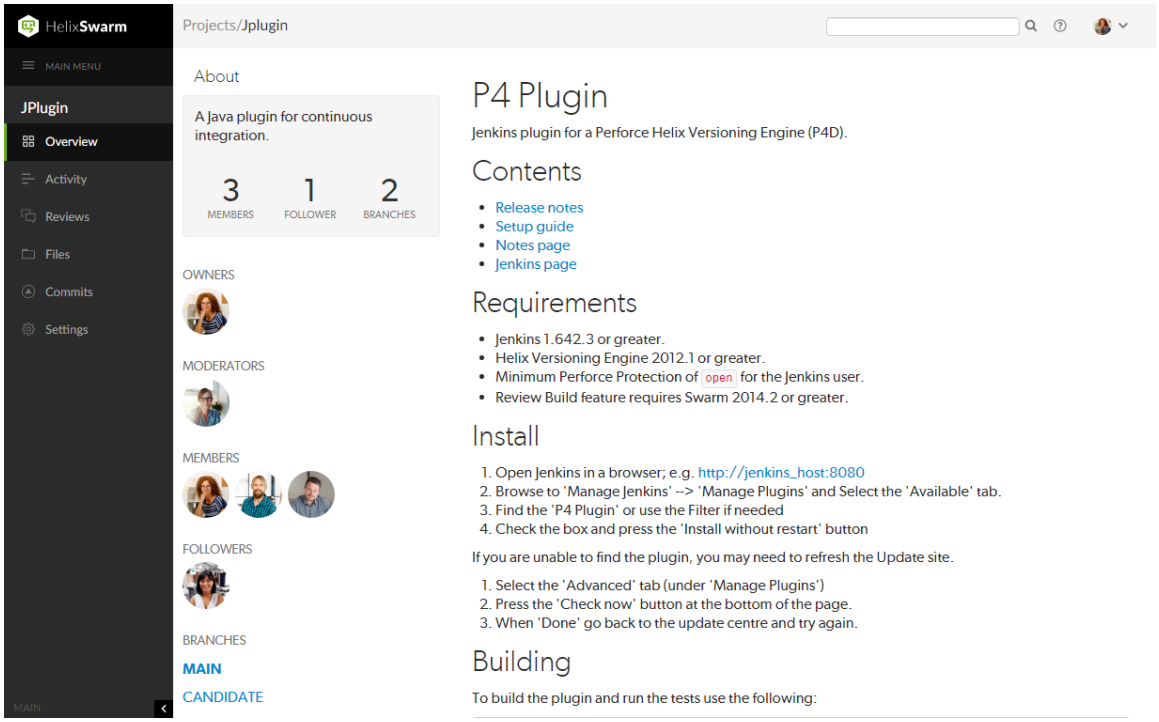
View a project by doing one of the following:

- Click on the project name in the projects list on the Swarm **Projects** page.
- Click on the project name or branch identifier in an activity stream.
- Visit the URL: **`https://myswarm.url/projects/project-name`**.

Overview page

The project **Overview** page shows a description of the project.

> **Important**
> The project's **Overview** page is only displayed if there is a **README** markdown file in the project's mainline.
>
> The **README** file can contain Markdown text, allowing a formatted description of the project to be provided. For a description of the type of formatting that is supported, see "Markdown in projects" on page 334. For details on how to configure the mainline of a project, see "Mainline branch identification" on page 509.



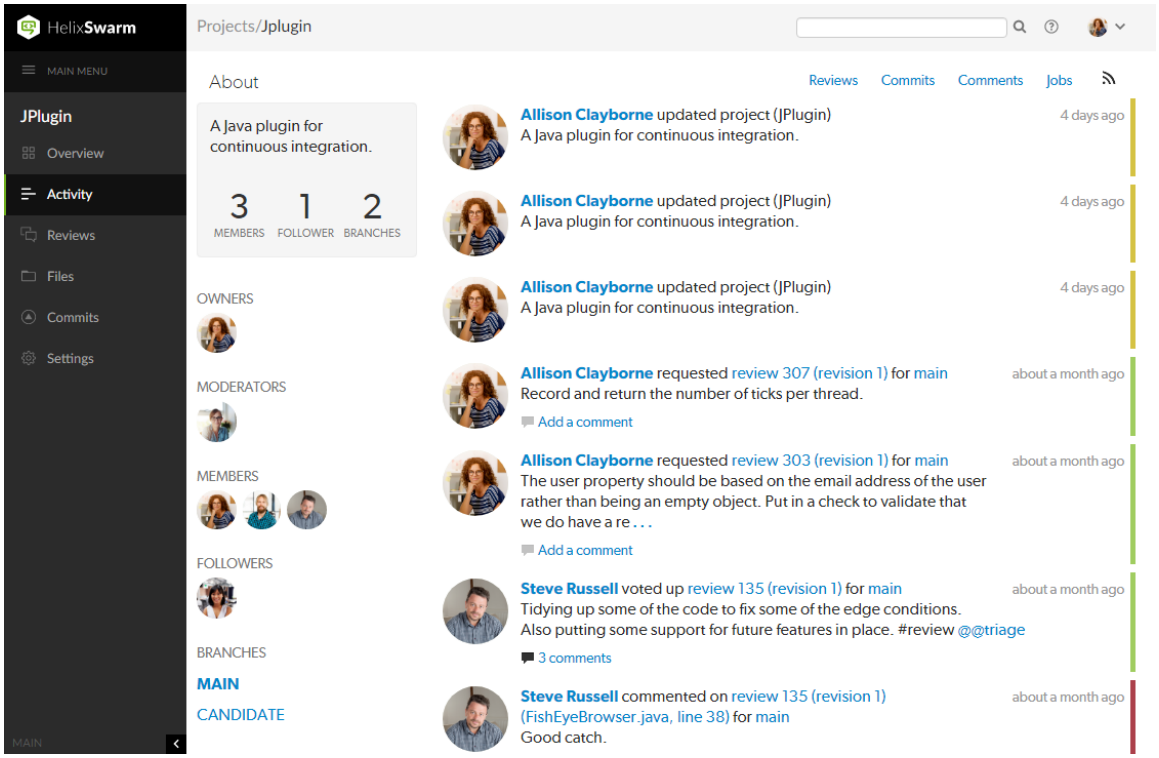
The following information is displayed in the project sidebar:

- A short description of the project
- A **Follow** button, click to follow the project. This button is not available if you are a member of the project.
- A list of project owners
- A list of project moderators
- A list of project members
- A list of project followers
- A list of the branches defined for the project

> **Tip**
> Hover over a branch name to view the moderators on that branch.

Activity page

The project **Activity** page shows the activity stream for the project



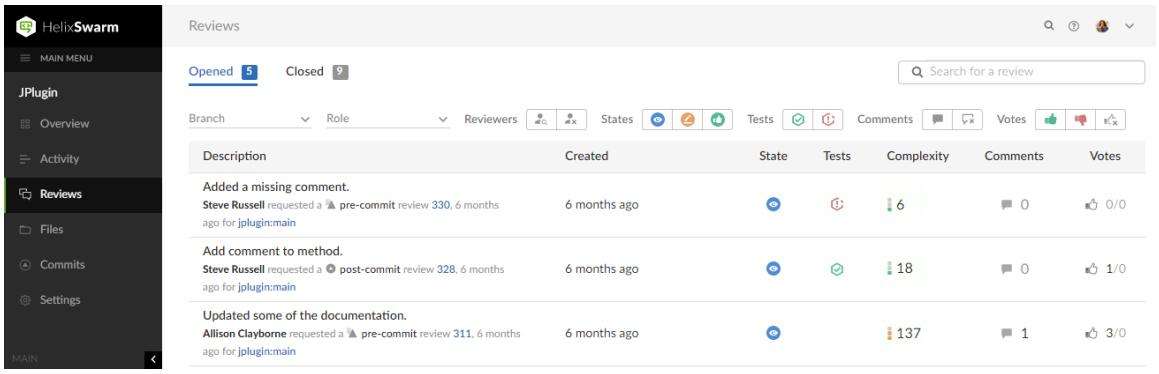
The following information is displayed in the project sidebar:

- A short description of the project
- A **Follow** button, click to follow the project. This button is not available if you are a member of the project.
- A list of project owners
- A list of project moderators
- A list of project members
- A list of project followers
- A list of the branches defined for the project

> **Tip**
> Hover over a branch name to view the moderators on that branch.

Reviews page

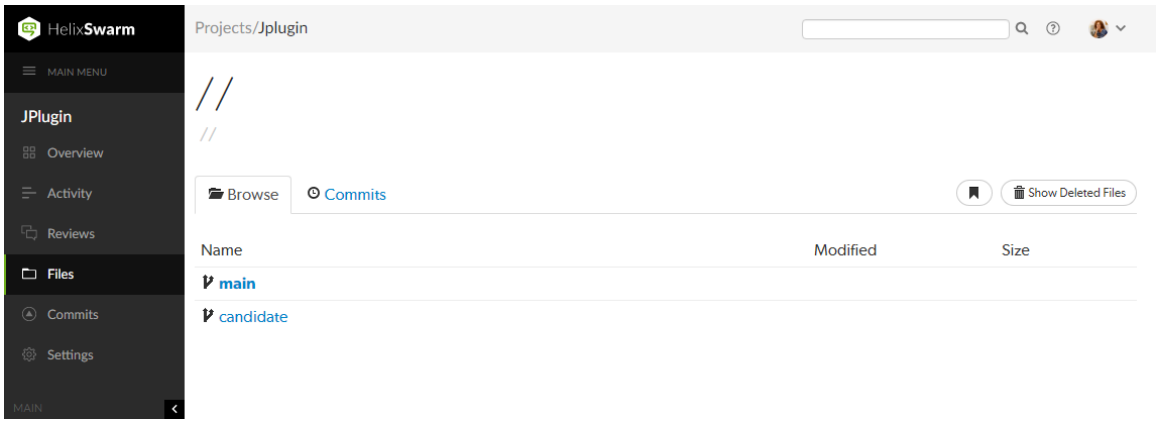
The project **Reviews** page shows a list of code reviews specific to the project.



For more details on browsing, filtering, and searching reviews, see "Reviews list" on page 342.

Files page

The project **Files** page shows a list of files for the project, starting with a folder view representing each branch. Branches are designated with the *branch* icon .

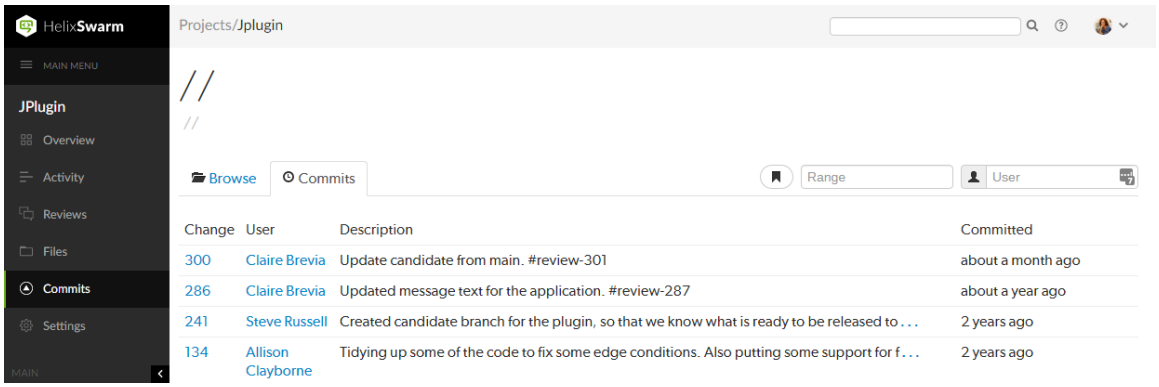


The project's *main* branch, identified by using a name such as *main*, *mainline*, *master*, *trunk*, is sorted to the top of the list of branches and appears in bold. The list of names can be configured, see "Mainline branch identification" on page 509 for details.

For more information on browsing files, see "Files" on page 243.

Commits page

The project **Commits** page shows a list of changes made to the project.

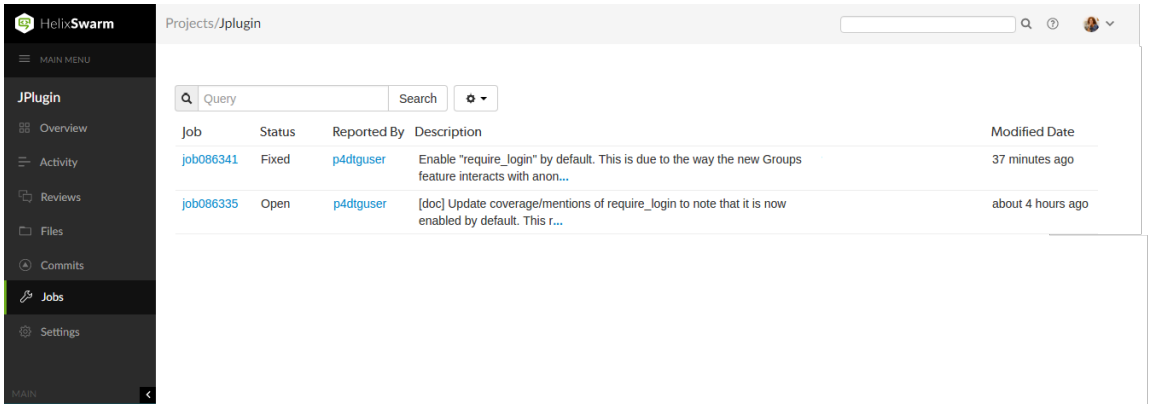


For more details on history browsing, see "Commits" on page 252.

Jobs page

The project **Jobs** page shows a list of jobs associated with the project.

> **Important**
> The **Jobs** page is only displayed when the project configuration includes a *job filter*. See "Add a project" on page 399 for details.



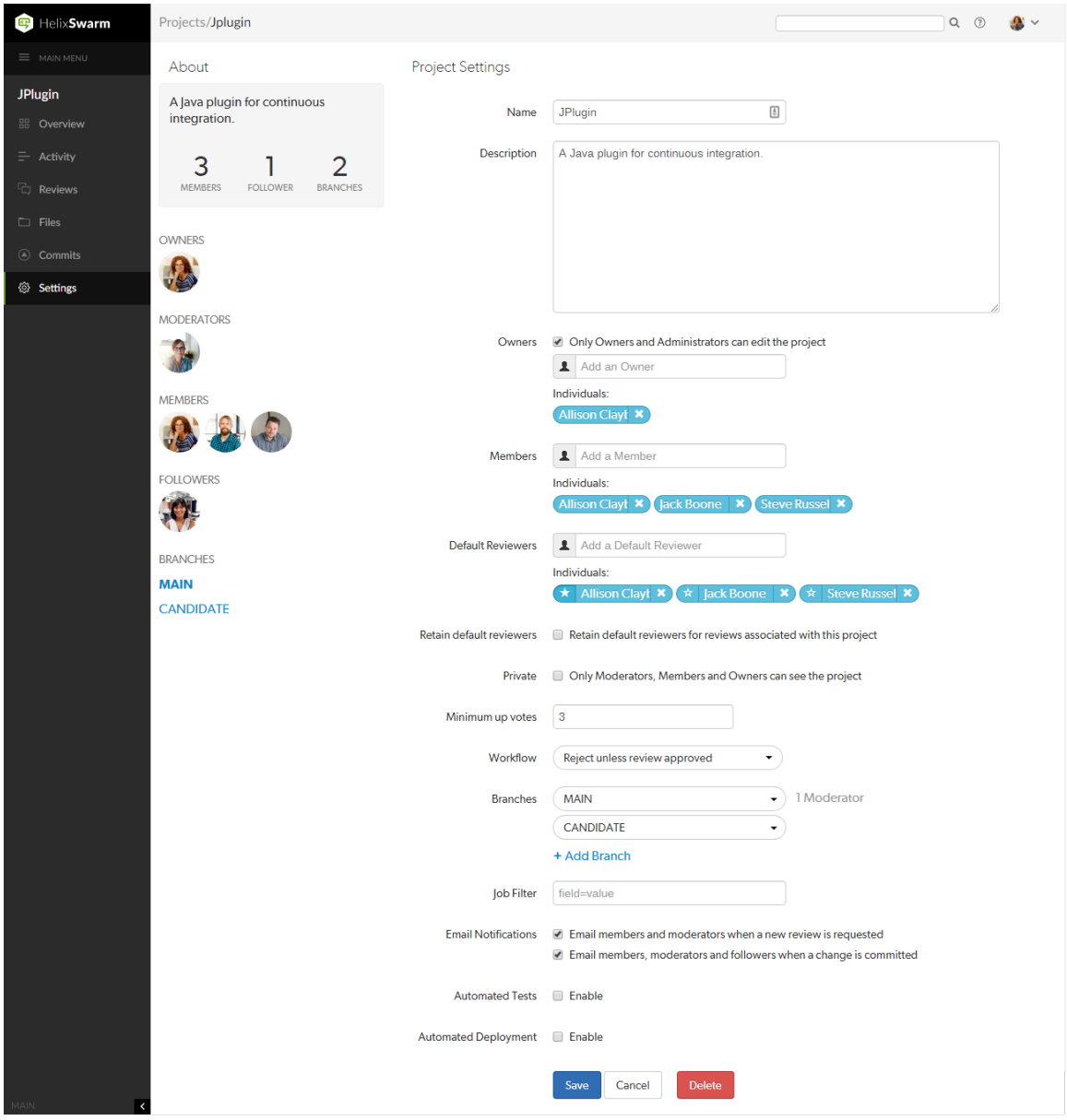
For more details on browsing and searching jobs, see "Jobs" on page 254.

Settings page

The project **Settings** page shows the settings associated with the project.

> **Tip**
> By default, if the **Only Owners and Administrators can edit the project** check box is selected for a project, only the project owners and administrators can view the project **Settings** page.

This behavior can be changed by your Swarm administrator to allow project members that are not owners or administrators to view a read-only version of the project **Settings** page, see "Allow project members to view project settings" on page 539. The project **Automated Tests** and **Automated Deployment** details are hidden from project members unless they are an owner or an administrator. This enables project members to check the project settings but not change them.



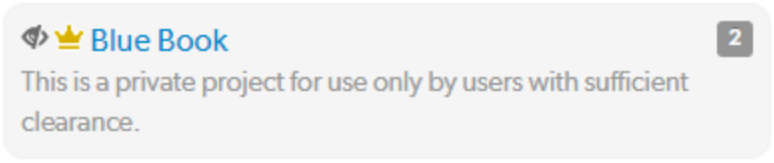
Tip
Hover over the **X Moderators** text of a branch to view the moderators on that branch.

For more information about project settings, see "Add a project" on page 399.

Private projects

Private projects, introduced in Swarm 2016.2, provide a way to make specific projects and their activity less visible to Swarm users. When a project is made private, only the projects owners, moderators, and members, plus users with *super* privileges in the Helix server, can see the project, its activity streams, and ongoing reviews.

If you are logged in as an owner, moderator, or member of a private Swarm project, that project appears on the Swarm project page with an *eye* icon to indicate that it is private and has limited visibility:



Similarly, the *eye* icon appears beside the project's title when viewing the project. When you hover your mouse over the *eye* icon, a tooltip appears indicating that this project is indeed private.

Caveats

The following are important caveats regarding private projects:

- While Swarm can mask the existence of projects, their activity, and reviews, Swarm honors each user's access to files within the Helix server. This means that users that have access to the files in a private project's branches can browse to those branch paths within the depot and see the files and any committed changes.
- If you need to prevent access to important files, your Helix server administrator is required to manage the protections table accordingly. For details, see Authorizing Access in the *Helix Core Server Administrator Guide*.

- It is possible for a user to start a review that touches files that belong to a private project. If the user is not a member, owner, or moderator of the private project, or does not have super privileges in the Helix server, they cannot participate in the review.
- If a user is not a member, owner, or moderator of a private project, or does not have super privileges in the Helix server, and they are added as a review participant (via an "@mention" on page 326 or by editing a review's participants) to a review containing files within the private project's branches, that user cannot participate in the review: the user cannot see the review, its files, or comments. Due to limitations in how Swarm sends email notifications, such users could still receive notifications for reviews they cannot participate in.
- Notifications usually include links and mentions of the associated projects. Notifications involving private projects are filtered to remove any references to those private projects.

Workflows

Important
Workflow is enabled by default and can be disabled by your Swarm administrator, see "workflow " on page 605.

A workflow can be applied to a project or project branch to ensure that changelists and code reviews in the project/branch follow the rules specified in that workflow.

- Swarm workflows can be created by any Swarm user.
- Shared Swarm workflows can be viewed by any Swarm user.
- Shared Swarm workflows can be applied to a project or project branch by any Swarm user that is authorized to edit the project.
- The global workflow can be viewed by any Swarm user but can only be edited by a Swarm user with *super* or *admin* privileges, or users that have been made an owner.

Important
The Swarm administrator can enforce a minimum workflow rule setting (by setting the global rule to **Enforce**) for the entire Helix Core server. If a project or project branch has an associated workflow, the global workflow rule is merged with the workflow rule and the most restrictive setting is used.

- For more information about how workflow rules are merged, see "Merging multiple workflows" on page 429.
- For more information about setting Global Workflow rules, see "Workflow global rules" on page 605.

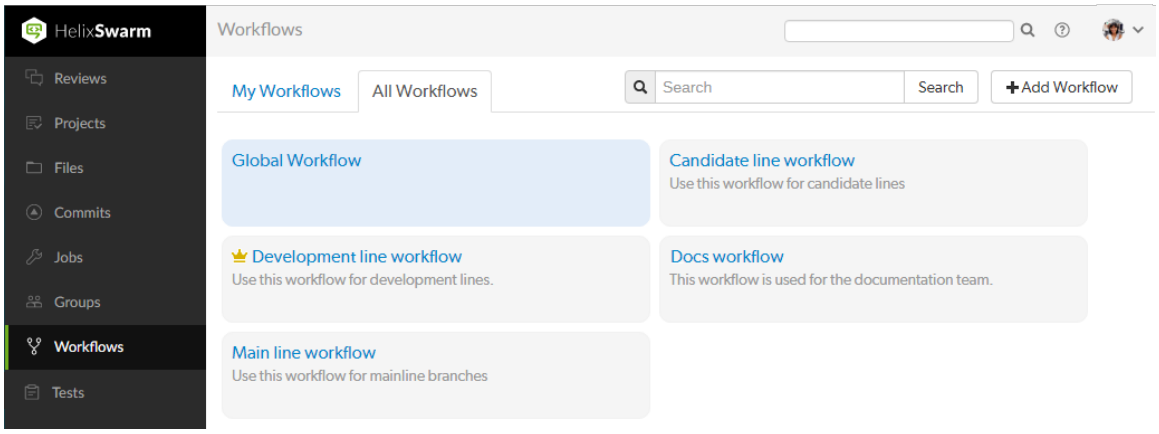
This section provides an introduction to listing, searching, and viewing workflows.

- For an overview of workflows, see "Workflow overview" on page 422.
- For instructions on how to add a workflow, see "Add a workflow" on page 433.

- For instructions on how to delete a workflow, see "Delete a workflow" on page 438.
- For instructions on how to add a workflow to a project, and a project branch, see "Add a project" on page 399.

Listing workflows

To view a list of workflows, click **Workflows** in the menu.



Logged-in users can choose which workflows to display by clicking on the **My Workflows** tab or the **All Workflows** tab. Anonymous users only see the global and shared workflows, the **My Workflows** tab is not available to anonymous users:

- The **My Workflows:** tab lists all of the workflows that you are an owner of.
- The **All Workflows:** tab lists the global workflow, all of the shared workflows, and workflows that you are an owner of.

> **Tip**
> The **Workflow Owner** icon is displayed for workflows you own.

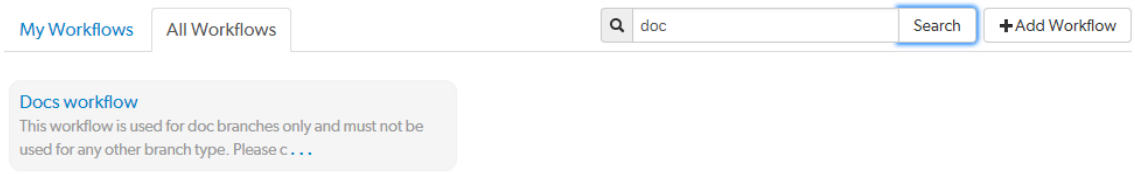
Search workflow names and workflow descriptions

You can search for Swarm workflow names and for the content of the workflow description from the workflows page:

1. Enter text in the search box.
2. Click **Search**.

   Any workflow names, and descriptions that match the search text are displayed in the workflows

tab:



Tip
Switch to the other workflow tab to display search results for that tab.

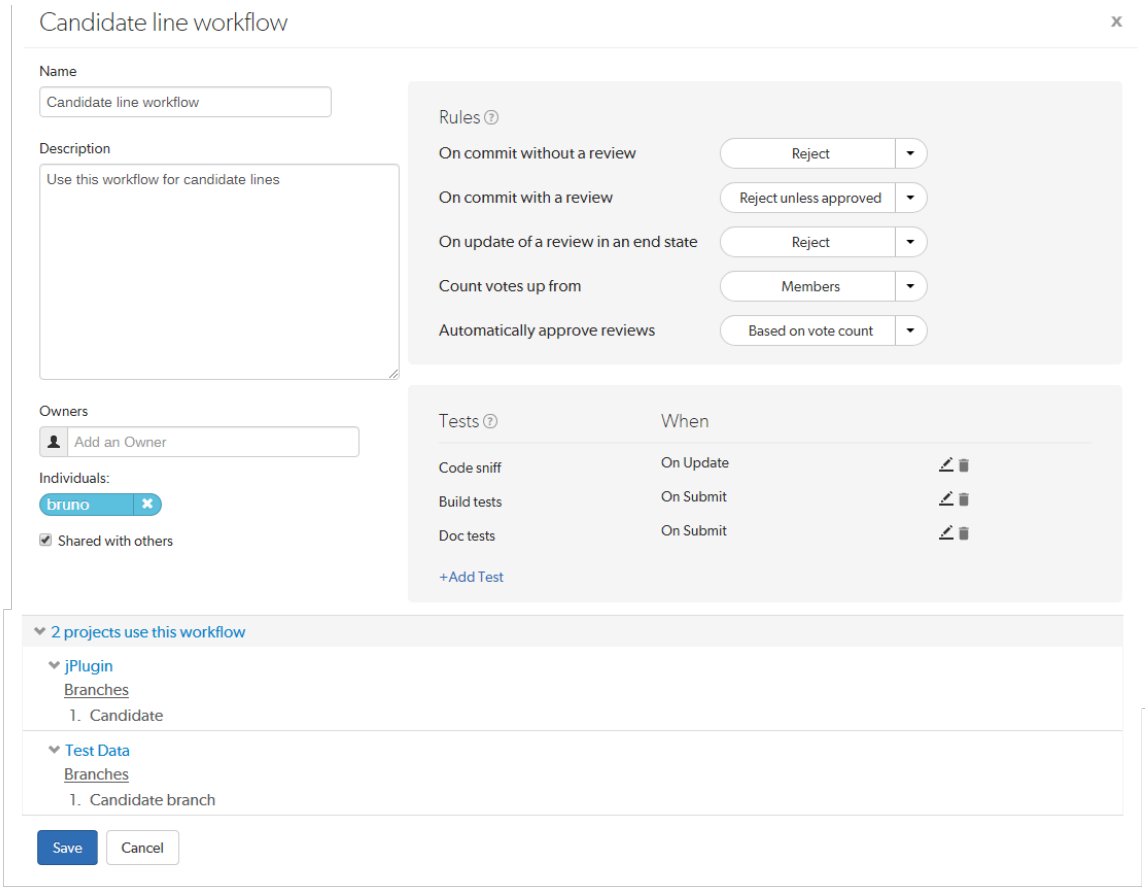
Viewing a workflow that you own

To view workflow details:

1. Click **Workflows** in the menu.
2. Click on the name of the workflow you want to view.

   For information about editing the workflow, see "Add a workflow" on page 433.

   Note
   You can edit a workflow if you are the owner of the workflow, or if you have *super* user rights.



Tip

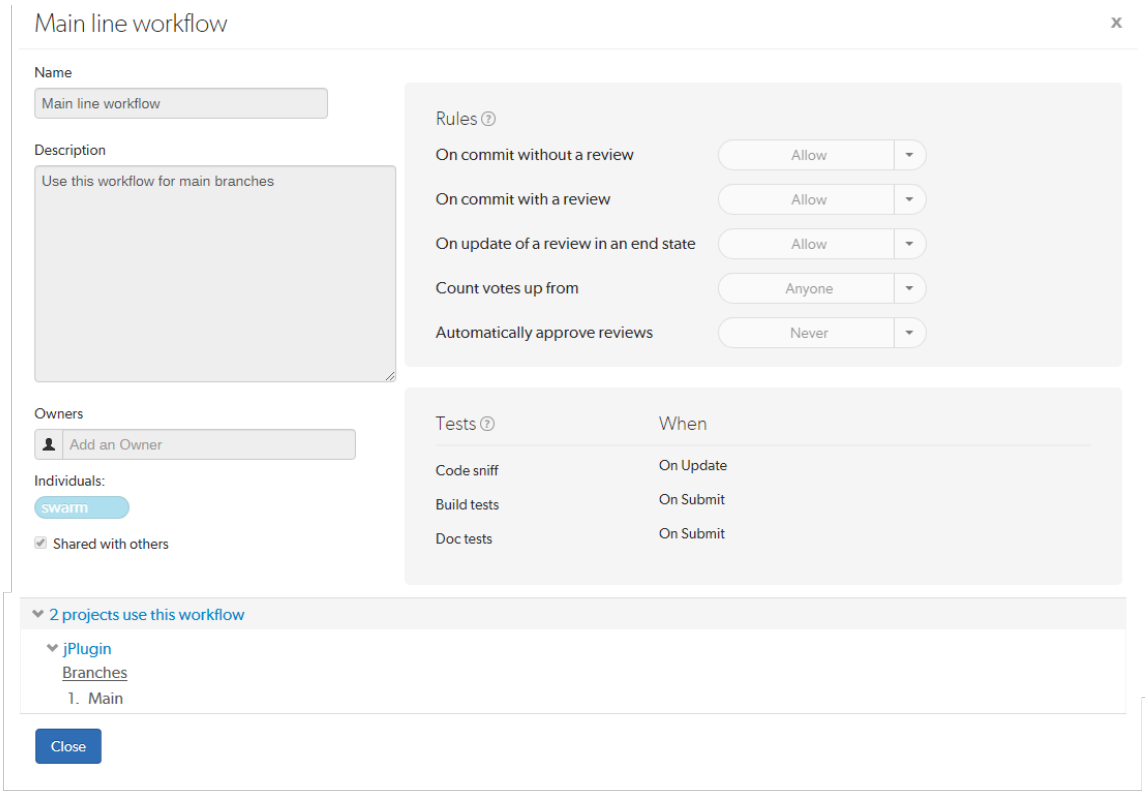
- The project count does not include project branches.
- If the workflow is associated with a Private project, the private project name is only displayed if you are authorized to view it. Private projects are included in the project count.

Viewing a shared workflow that you do not own

Shared workflows can be viewed and used by all Swarm users.

To view workflow details:

1. Click **Workflows** in the menu.
2. Click on the name of the workflow you want to view.



Tip

- The project count does not include project branches.
- If the workflow is associated with a Private project, the private project name is only displayed if you are authorized to view it. Private projects are included in the project count.

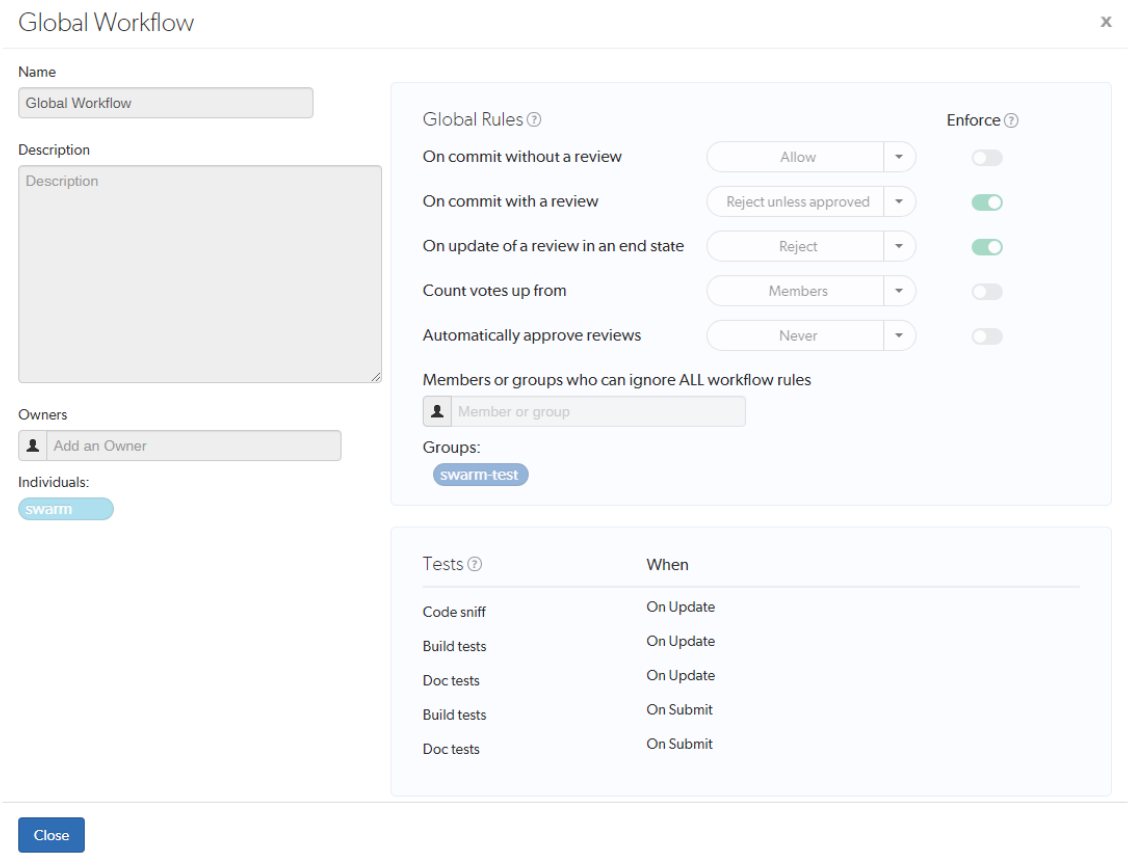
Viewing the global workflow

The global workflow can be viewed by any Swarm user but can only be edited by a Swarm user with *super* or *admin* privileges, or users that have been made an owner. For information about editing the global workflow, see "Workflow global rules" on page 605.

To view the global workflow details:

1. Click **Workflows** in the menu.
2. Select the **All Workflows** tab.
3. Click on **Global Workflow** to view it.

Tip
By default, the global workflow is named **Global Workflow**, but it can be changed by your Swarm administrator. To help you identify the global workflow it is always shown as the first workflow in the **All Workflows** tab and it has a different background color to the other workflows.



Tests

Important
The **Tests** page is only available if Workflow is enabled, workflow is enabled by default.

Associate a test with a workflow to ensure that the test is run when a review associated with that workflow is started or updated. Associate a test with the global workflow to ensure that the test is run whenever a Swarm review is started or updated. This ensures that the global tests are enforced for all changes even if they are not part of a project.

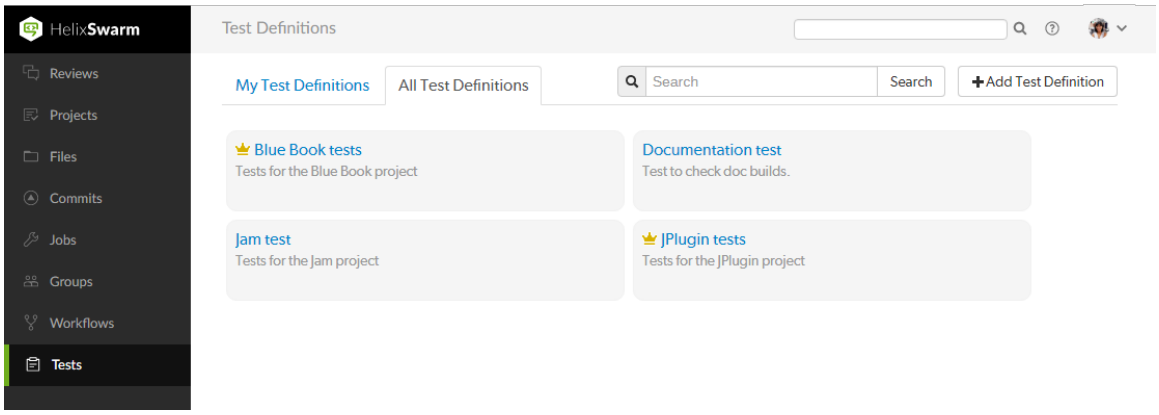
- Swarm tests can be created by any Swarm user.
- Shared Swarm tests can be viewed by any Swarm user.
- Shared Swarm tests can be added to a workflow by any Swarm user that is authorized to edit the workflow.
- Shared Swarm tests can be added to the global workflow by any Swarm user that is authorized to edit the global workflow.

This section provides an introduction to listing, searching, and viewing tests.

- For instructions on how to add a test, see "Add a test" on page 440.
- For instructions on how to edit a test, see "Edit a test" on page 445.
- For instructions on how to delete a test, see "Delete a test" on page 446.
- For instructions on how to add a test to a workflow, see "Add a workflow" on page 433.
- For instructions on how to add a test to the global workflow, see "Global workflow" on page 605.

Listing tests

To view a list of tests, click **Tests** in the menu.



Logged-in users can choose which tests to display by clicking on the **My Tests** tab or the **All Tests** tab. Anonymous users only see the shared tests, the **My Tests** tab is not available to anonymous users:

- The **My Tests:** tab lists all of the tests that you are an owner of.
- The **All Tests:** tab lists the all of the shared tests, and tests that you are an owner of.

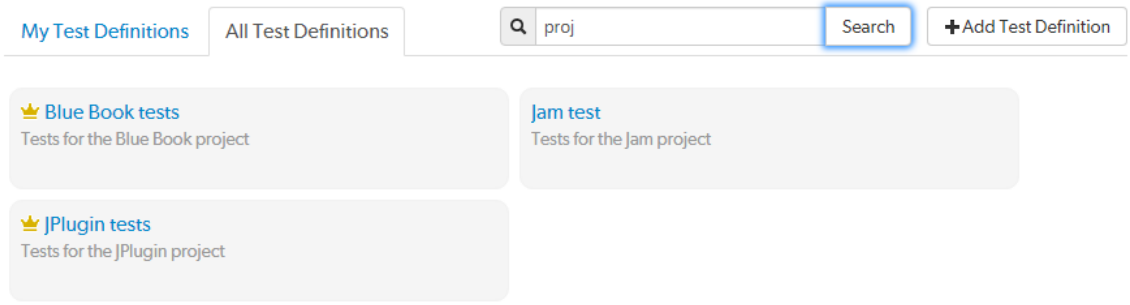
> **Tip**
> The **Test Owner** icon is displayed for tests you own.

Search test names and test descriptions

You can search for Swarm test names and for the content of the test description from the tests page:

1. Enter text in the search box.
2. Click **Search**.

   Any tests names and descriptions that match the search text are displayed in the tab:



> **Tip**
> Switch to the other test tab to display search results for that tab.

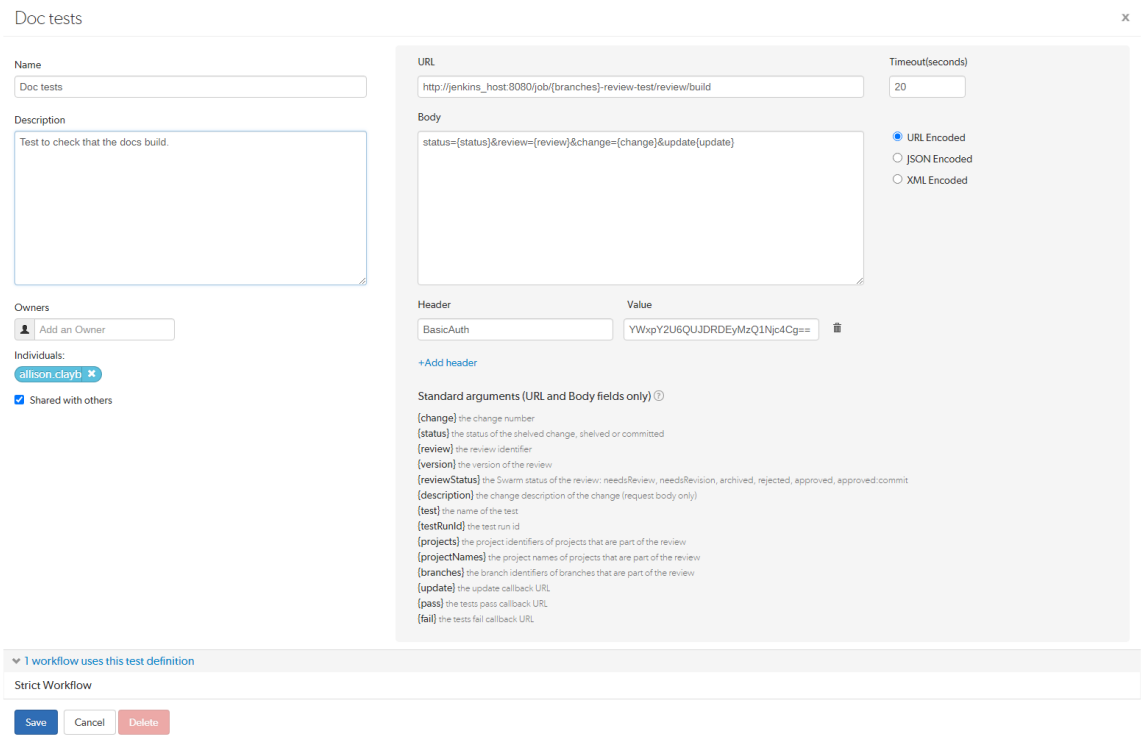
Viewing a test that you own

To view test details:

1. Click **Tests** in the menu.
2. Click on the name of the test you want to view.

   For information about editing the workflow, see "Add a test" on page 440.

   > **Note**
   > You can edit a test if you are the owner of the test, or if you have *super* user rights.



Viewing a shared test that you do not own

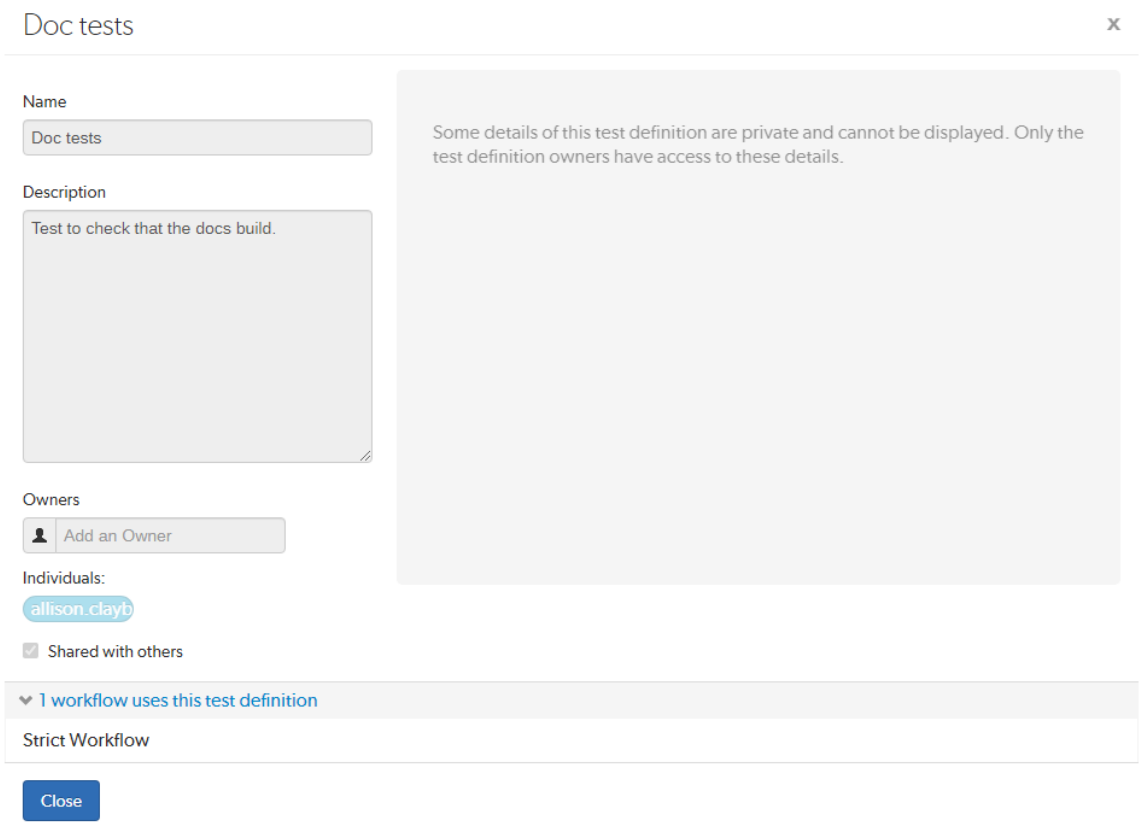
Shared tests can be viewed and used by all Swarm users.

To view test details:

1. Click **Test** in the menu.
2. Click on the name of the test you want to view.

   To avoid leaking sensitive information to users that do not own the test, configuration details are

hidden.



Notifications

Provided you have entered a working email address for your userid in Helix server, Swarm sends email notifications to you when various events take place. The table below shows the notifications sent if all of the project, group, and user notifications are enabled. This is the default behavior.

Note
The system-wide default notifications can be changed by the system owner. See "Global settings" on page 529 for details.

Tip
Many of the notifications can be disabled, this allows project, group, and user notifications to be customized to limit the number and type of notifications sent.

- Project notifications are configured on the project settings tab.
- Group notifications are configured on the group Notifications tab.
- User notifications are configured on the user Notifications tab.

Legend:

A = Author	**PM** = Project member or project member group	✔ = role user/group receives notification
R = Reviewer or reviewer group	**PF** = Project follower	✖ = role user/group does not receive notification
M = Moderator or moderator group	**AF** = Author follower	⊘ = role not included for this event
GM = Group member	**CA** = Comment author	

Event	Notification sent to one of these roles...								
	Event by you?	A	R	M	GM	PM	PF	AF	CA
A review is committed in Swarm	✖	✔	✔	✔ 3	✔4	✔ 1	✔1	✖	⊘
A review is committed outside of Swarm	✖	✖	✔	✔ 3	✔4	✔ 1	✔1	✖	⊘
A change is committed outside of Swarm (see "(1) Committed change notifications" on page 320)	✖	✖	⊘	✔ 3	✔ 4	✔ 1	✔1	✔1	⊘
A review is started (see "(2) Review start notifications" on page 320)	✔	✔	✔	✔ 3	✔4	✔ 2	✖	✖	⊘
A reviewer casts a vote	✔	✔	✔ 5	✖	✔4	✖	✖	✖	⊘

Event	Notification sent to one of these roles...								
	Event by you?	A	R	M	GM	PM	PF	AF	CA
A review's state changes	✗	✓	✓ 5	✗	✓4	✗	✗	✗	⊘
A review's reviewers are changed	✗	✓	✓ 5	✗	✓4	✗	✗	✗	⊘
A review's files are updated	✓	✓	✓ 5	✗	✓4	✗	✗	✗	⊘
A review's automated tests have finished (see "(7) Tests have finished" on page 322)	✗	✓	✓	✓	✓4	✗	✗	✗	⊘
A review's description is changed	✗	✗	✗	✗	✗	✗	✗	✗	⊘
A review comment is created (see "Comment notification delay" on page 273).	✗	✓	✓ 5	✗	✓4	✗	✗	✗	✗ 6
A review comment is edited (see "Comment notification delay" on page 273).	✗	✓	✓ 5	✗	✓4	✗	✗	✗	✗ 6
A committed change comment is created	✗	✓	⊘	✗	✓4	✗	✗	✗	✗ 6
A committed change comment is edited	✗	✓	⊘	✗	✓4	✗	✗	✗	✗ 6

Event	Notification sent to one of these roles...								
	Event by you?	A	R	M	GM	PM	PF	AF	CA
Someone joins or leaves a review	✗	✓	✓ 5	✗	✓4	✗	✗	✗	⊘
Someone likes a comment you wrote	✗	✗	✗	✗	✗	✗	✗	✗	✓5
You like a comment you wrote	✓	✗	✗	✗	✗	✗	✗	✗	✗

Any custom modules added to Swarm may also send notifications.

Important
Email delivery for events related to restricted changes is disabled by default. See "Restricted Changes" on page 565 for details on how to enable restricted change notifications.

@mention notifications

Users: @mention

Using an @mention in a review, changelist, or comment causes the referenced userid to receive a notification and be included in any future notifications regarding the associated file or review.

When a comment is added to a job, Swarm sends a notification to users listed in user fields in the job, users @mentioned in the job description, and the authors of any associated changes.

Note
Typically, you would not receive a notification when you @mention yourself. @mentions themselves do not trigger notifications; they inform who receives notifications. See "notify_self" on page 492 for details on how to enable self notification.

Groups: @@mention

Using an @@mention in a review, changelist, or comment causes the referenced groupid members to receive a notification and be included in any future notifications regarding the associated file or review.

When a comment is added to a job, Swarm sends a notification to groups listed in group fields in the job, groups @@mentioned in the job description, and the authors of any associated changes.

Note

- **Group mailing list enabled:** notifications are sent to the group email address.
- **Group mailing list disabled:** notifications are sent to the group members individual email addresses.

(1) Committed change notifications

Notifications for committed changes are sent by default, but can be disabled or require users to opt-in. Please see the notification configuration and adding a project for more details.

When committed change notifications are configured for opt-in, you need to copy the configuration's special depot path to the `Reviews:` field in your user spec within Helix server. Once you have done so, Swarm can send you committed change notifications for any change that matches a depot path specified in the `Reviews:` field.

Update Reviews with p4

1. Begin editing your user spec:

   ```
   $ p4 user
   ```

2. Edit the `Reviews:` field to include the special depot path, plus any other paths you want to receive notifications for.
3. Save the spec.

Update Reviews with P4V

1. Select **Connection > Edit Current User**.
2. Edit the `Reviews:` field to include the special depot path, plus any other paths you want to receive notifications for.
3. Click **OK**.

(2) Review start notifications

By default, notifications are sent when a review is started, but can be disabled for a project, group, or user.

Note
If a user or group is added as a reviewer when a new review is created, the reviewer will be notified that the review has been started. This initial notification cannot be muted.

(3) Moderator notifications

Moderators and moderator groups are associated with specific project branches. Moderators receive notifications for reviews and commits against the branches they are associated with.

(4) Group member notifications

Groups can be members of a project, moderators of a project branch, and reviewers of a review. The notifications that group members receive will depend on, the role the group has and the group notification settings that are configured for that group. Notifications received by group members also depend on whether the group has a group mailing list enabled, in this case additional notification settings are available. See *Group mailing list enabled* below for details.

Group mailing list disabled

Group members can be notified when a member of the group starts a review. Group members can also be notified when a change is committed by, or on behalf of, a changelist owner who is also a member of the group. These two notifications can be individually selected as required. See "Add a group" on page 393 for details.

Notifications are sent to the group members individual email addresses.

Group mailing list enabled

Group members can be notified when a member of the group starts a review. Group members can also be notified when a change is committed by, or on behalf of, a changelist owner who is also a member of the group. These two notifications can be individually selected as required.

Additional notifications are available if the group mailing list address is enabled. See "Add a group" on page 393 for details.

Notifications are sent to the group email address.

(5) Disable notifications

By default, notifications for events in a review are sent to all reviewers. Reviewers can disable notifications during a review to avoid further email notifications. Once notifications are disabled for a reviewer, they can be re-enabled when they are specifically @mentioned; reviewers can disable notifications again after they have been re-enabled. See "Disable notifications" on page 360 for details.

(6) Comment author notifications

By default, notifications are not sent to comment authors, but Swarm can be configured to send notifications of comments to comment authors. See "notify_self" on page 492 for details.

(7) Tests have finished

By default, notifications are sent when:

- Automated tests have failed for a review.
- The first time automated tests pass for a review after a test failure for that review.

Log in/Log out

When you are not logged into Swarm, certain features are unavailable to you, such as providing comments to changes or reviews, adding projects, and more.

Log in to Swarm

1. Click **Log in** on the right of the Swarm header.

   > **Tip**
   > If Helix Authentication Service is configured for your Helix server and Swarm you will be directed to the sign in process used by your Identity Provider (IdP).

2. Type in your username and password, appropriate for the Helix server that Swarm is configured to use.
3. Select the **Remember me** check box if you prefer to stay logged in between browser restarts.

   > **Note**
   > The Helix server can enforce maximum log in times. You may become logged out even if **Remember me** is selected. Swarm administrators can change the maximum log in time, see "Sessions" on page 563 for details.

4. Click the **Login** button.

Log out of Swarm

1. Click your userid, on the right of the Swarm header.
2. Select **Log Out** from the drop-down menu.

Tip

- If Helix Authentication Service is configured for your Helix server, logging out of Swarm will not invalidate your Identity Provider (IdP) login status. If you try to log back in to Swarm while your IdP status is still valid, you will not be prompted to complete the log in steps.

  If **`require_login`** is also enabled, Swarm will return you to the login and your IdP will automatically log you back in. In this case log out from your Identity Provider page before logging out from Swarm.

- If a custom redirect has been configured by your Swarm administrator, you are logged out of Swarm and then redirected to the URL specified by the administrator.

  The custom redirect can be set to any internal or external URL, for example:

  - Company intranet, extranet, internet, FTP, or Web-mail page
  - Industry news website
  - Identity Provider page to invalidate your IdP log in status

require_login

By default, Swarm requires users to log in, which prevents anonymous users from accessing a Helix server via Swarm. Users who have not logged in see a log in page immediately when visiting Swarm:

The steps to log in are identical to using the Log in dialog.

Swarm administrators can disable require_login to allow anonymous users to see commits, reviews, etc.

Note
service and *operator* users are not permitted to login. See User types in the *Helix Core Server Administrator Guide* .

Notable minor features

Quick URLs

Swarm handles URLs intelligently to reduce the amount of information you need to enter to quickly locate what you are looking for:

1. Enter a URL in the following format:

   `https://myswarm.url/<identifier>`

2. Swarm checks the **<identifier>** and redirects you to the first match it finds. Swarm checks for the identifier in following order:
   - Reviews
   - Changelists
   - Depot paths
   - Projects
   - Jobs
   - Users
   - Groups
   - Depot names
   - Git Fusion SHA1 (or fragments).

Note
- Changelist and review identifiers must be numeric.
- Git Fusion SHA1 identifiers (or a fragment of one) work when the SHA1 refers to a single changelist. If the SHA1 refers to more than one changelist, which can occur for Git branches, Swarm reports a **404 error**.
- If you enter an identifier that does not exist for any type of resource, Swarm displays a **Page Not Found** error.

Example usage

Display the latest version of a review

For example, to display review 124: enter the URL **https://*myswarm.url*/124**, Swarm redirects to **https://myswarm.url/reviews/124**.

Tip
- To display a specific version of a review, enter the full URL including **reviews** and append **/v<*n*>/** to the URL. For example, to display version 2 of review 124: enter **https://myswarm.url/reviews/124/v2/**
- To display a diff between two versions of a review, enter the full URL including **reviews** and append **/v<*n,n*>/** the URL. For example, to display the diff between version 2 and 4 of review 124: enter **https://myswarm.url/reviews/124/v2,4/**
- To open a review with a specific tab open, append **#<*tab name*>** to the end of the URL (**#<*tab name*>** must be the last item in the URL):

- **Files tab:`#files`** (default if **`#<tab name>`** is not specified)
- **Comments tab:`#comments`**
- **History tab:`#history`**

Display a changelist

For example, to display changelist 123: enter the URL **`https://myswarm.url/123`**, Swarm redirects to **`https://myswarm.url/changes/123`**.

Tip
To open a changelist with a specific tab open, append **`#<tab name>`** to the end of the URL:

- **Files tab:`#files`** (default if **`#<tab name>`** is not specified)
- **Comments tab:`#comments`**

Display the latest version of a file

Enter the URL **`https://myswarm.url/depot/alpha/readme.txt`**, Swarm redirects to **`https://myswarm.url/files/depot/alpha/readme.txt`**.

Tip

- To display a specific version of a file, enter the full URL including **`files`** and append **`?v=<n>`** to the URL. For example, to display version 2 of the **`depot/alpha/readme.txt`** file: enter **`https://myswarm.url/files/depot/alpha/readme.txt?v=2`**
- To open a file with a specific tab open, append **`#<tab name>`** to the end of the URL (**`#<tab name>`** must be the last item in the URL):
  - **View tab:`#view`** (default if **`#<tab name>`** is not specified)
  - **Commits tab:`#commits`**

Display the user profile page for jsmith

Enter the URL **`https://myswarm.url/jsmith`**, Swarm redirects to **`https://myswarm.url/users/jsmith`**.

@mention

When you write a changelist description, job description, comment, or review, use `@mention` to refer to projects, changelists, jobs, and users. Use `@@mention` to refer to groups. Swarm automatically creates a link for each @mention and @@mention, so it's easy to navigate to the specified resource. For example, when you include `@job12345` in a comment, Swarm turns that text into a link that, when clicked, displays the Jobs page for `job12345`.

> **Note**
> For jobs, the `@` character is optional: Swarm creates job links for any text that looks like a job identifier, such as `job012345`.
>
> Similarly, for changes and reviews, the `@` character is optional: Swarm creates change links for `change 1234`, or review links for `review 2345`.

When you start a code review, including a user `@mention`, or a group `@@mention` in the changelist description automatically makes them reviewers for that code review. During a code review, including a user `@mention` or group `@@mention` in a comment causes the mentioned user or group to receive notifications of code review events, even if they are not a member of your project or following you or your project.

Additional option for a user `@mention`:

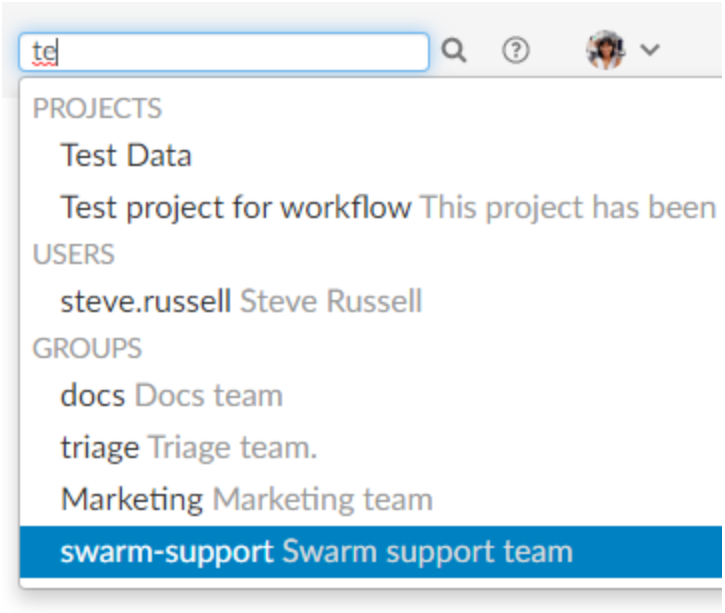
- `@*mention`: include an asterisk (`*`) before the userid to make the user a required reviewer. See "Required reviewers" on page 380 for details.

Additional options for a group `@@mention`:

- `@@*mention`: include an asterisk `*` before the groupid to make the group a required reviewer, **All votes required**. See "Required reviewers" on page 380 for details.
- `@@!mention`: include an exclamation mark `!` before the groupid to make the group a required reviewer, **One vote required**. See "Required reviewers" on page 380 for details.

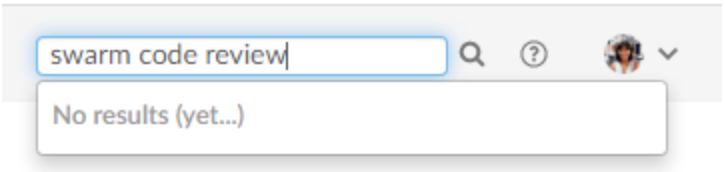
> **Note**
> - **Group mailing list enabled:** notifications are sent to the group email address.
> - **Group mailing list disabled:** notifications are sent to the group members individual email addresses.

Search



Swarm can search for users, groups, projects, and file paths. Enter keywords or path elements into the search box in the Swarm header. Navigate the results with the ↑ (up arrow) and ↓ (down arrow) keys. To display the details for a result, press **Enter**or click the search result.

Full-content searching is only available if your Swarm administrator installs the Helix server Search Tool. See "Search" on page 560 for details.



Swarm updates the search results as you type. Some results should appear within a second or two. You may have to wait a few seconds for final results to be incorporated into the results list. When Swarm does not yet have any results, it indicates such.

JIRA integration

Swarm ships with a JIRA module that can:

- create links to JIRA when Swarm displays JIRA issue identifiers in changelists, comments, jobs, reviews, etc.
- add links within JIRA back to Swarm, for JIRA issues associated with reviews or committed changes. These links reflect the current status of associated code reviews.

By default, the JIRA module is disabled. To enable JIRA integration, see "Enabling the JIRA module" on page 448.

Avatars

Each event in an activity stream includes an *avatar*, an image that represents the user responsible for the event. Avatars help to visually tie together various events and personalize the history presented in the stream.

Based on the email address entered in a user's or group's Helix server account, Swarm attempts to fetch an avatar from gravatar.com. Swarm selects an avatar from its collection of default avatars if an avatar is not returned from gravatar.com.

Hover your mouse over an avatar to display the fullname of the user or group.

Following

Whenever you see a **Follow** button, for example when you are viewing a project page or user profile, clicking the button causes Swarm to send you notifications whenever there is activity generated by the current resource.

This is useful if, in the case of a project, you are not a project member but want to know what's happening in the project. Or, in the case of a user, you want to see what activity that user generates.

To stop receiving notifications, visit the project page or user profile and click the **Unfollow** button.

Time

Swarm typically displays the time of an event, such as when a file was created, as about *X units ago*. Hover your mouse pointer over a time display to see a tooltip displaying the exact date and time of the event.

> **Tip**
> Configure Swarm to display the exact time and date of events by setting **Time Display** to **Timestamp** on your user Settings tab.

Keyboard shortcuts

Swarm provides the following keyboard shortcuts:

- **n** - While viewing "Diffs" on page 261, pressing **n** scrolls to the next difference.
- **p** - While viewing "Diffs" on page 261, pressing **p** scrolls to the previous difference.

- **ESC** - While viewing a dialog, pressing **ESC** closes the dialog.

  While entering text into a text area, pressing **ESC** stops text entry.

About Swarm

You can discover the version of Swarm you are using:

1. Log in to Swarm.
2. Click your userid, found at the right of the main toolbar, and select **About Swarm**.

   A dialog appears displaying the Swarm version.

Custom error pages

If Swarm encounters an error during processing, such as when a "Quick URLs" on page 323 is used that points to a non-existent resource, Swarm displays a custom error page.

Short links

Swarm provides a *short link* feature that creates shorter URLs than normal to make sharing specific views within Swarm easier. It is also possible to register or configure an alternate, shorter hostname to have even shorter URLs. See "Short links" on page 573 for details.

Conceptually, this is identical to http://tinyurl.com, or the Twitter feature http://t.co, but is restricted to Swarm URLs on a hostname you control.

Swarm displays a *bookmark* button  when viewing files or folders in the depot.

Click the button to display a popup containing the short link. Press **CTRL+C** (on Windows and Linux), or **Command+C** (on Mac OSX), to copy the short link. You can then paste the short link anywhere you'd like to share the current file or folder view in Swarm.

Markdown

You can use *Markdown* to add text styles to comments and project overview pages. Swarm can also display Markdown files stored in the Helix Core server. Markdown is not supported in the Swarm review description.

In this section:

Common Markdown styles

A number of the more common Markdown styles are shown below, for a more thorough list of the styles you can use, see the Markdown cheatsheet:

- **bold** and *italics* can be specified with `**bold**` or `*italics*`.
- Unordered lists can be specified with asterisk (`*`) markers. Plus (`+`) and minus (`-`) signs also work:

```
* This
* That
* Another
```

- Ordered lists can be specified with numbered markers:

```
1. First
2. Second
3. Third
```

- Hypertext links can be specified with [Link text](http://address/page). If you don't need to add anchor text, then a URL in the text without any markup is linkified.
- You can mark inline text as code using `backticks`.
- You can also mark blocks of code using three backticks on a line, like:

````
```
var text = "Some code text";
alert(text);
```
````

- Headers can be defined using hash (#) marks.

```
# Main heading
## Sub heading
### Lesser heading
```

- Images can be added as well. You can either provide the full Swarm *view* URL, or use a relative Swarm *view* URL to reference something in Swarm.

```
![alt text]
(https://swarm.company.com/view/depot/www/dev/images/jamgraph-jam.gif
"Title Text")
![alt text](/view/depot/www/dev/images/jamgraph-jam.gif "Title Text")
```

> **Tip**
> To find the Swarm **`view`** URL for an image file:
>
> 1. From Swarm, click on the **Files** tab and navigate to the image file.
> 2. Click on the image file so that it opens in the **View** tab.
> 3. Click the **Open** button for the file to open the image in you browser.
> 4. The URL in your browser address bar is the Swarm *view* URL for the image file.

- Tables are supported by using pipe (|) separators between columns and colons (:) for justification.

```
| Tables   | Look   | Like this |
| -------- | ----:  | :-------: |
| Left     | right  | center    |
```

- It is also possible to blockquote paragraphs using the greater than symbol (>).

```
> Blockquotes can be displayed like this, using the
> the greater than sign at the start of the line.

Normal text resumes here.
```

Markdown example

The following example shows how Markdown text can be used in a comment but it would display the same for a project overview page.

The source and the final result are shown:

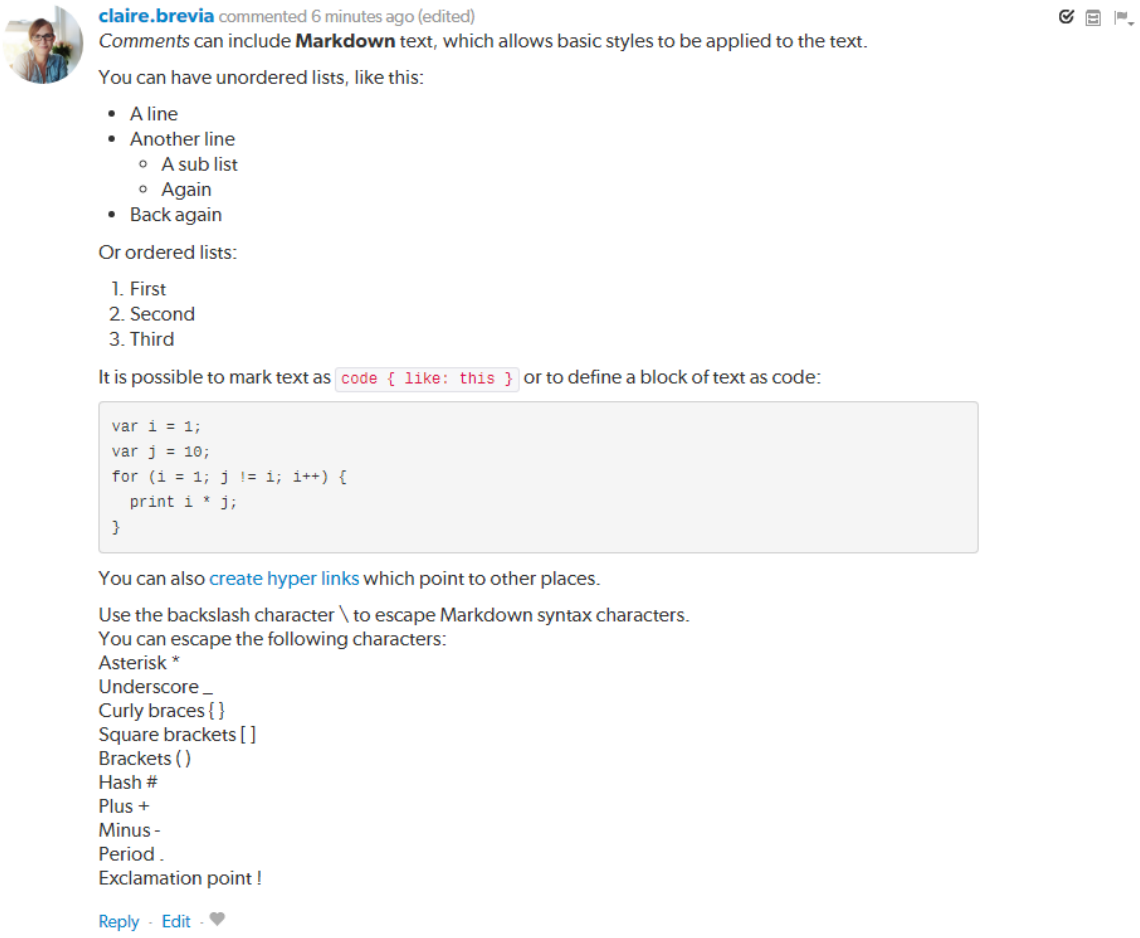
- "Source text" on the facing page
- "Rendered view of Markdown text" on page 333

Source text

````
*Comments* can include **Markdown** text, which allows basic styles to be
applied to the text.

You can have unordered lists, like this:
* A line
* Another line
  * A sub list
  * Again
* Back again

Or ordered lists:
1. First
2. Second
3. Third

It is possible to mark text as `code { like: this }` or to define a block
of text as code:
```
var i = 1;
var j = 10;
for (i = 1; j != i; i++) {
  print i * j;
}
```

You can also [create hyper links](http://www.perforce.com) which point to
other places.

Use the backslash character \ to escape Markdown syntax characters.
You can escape the following characters:
Asterisk \*
Underscore \_
Curly braces \{ \}
Square brackets \[ \]
````

```
Brackets \( \)
Hash \#
Plus \+
Minus \-
Period \.
Exclamation point \!
```

Rendered view of Markdown text



claire.brevia commented 6 minutes ago (edited)

Comments can include **Markdown** text, which allows basic styles to be applied to the text.

You can have unordered lists, like this:

- A line
- Another line
  - A sub list
  - Again
- Back again

Or ordered lists:

1. First
2. Second
3. Third

It is possible to mark text as `code { like: this }` or to define a block of text as code:

```
var i = 1;
var j = 10;
for (i = 1; j != i; i++) {
  print i * j;
}
```

You can also create hyper links which point to other places.

Use the backslash character \ to escape Markdown syntax characters.
You can escape the following characters:
Asterisk *
Underscore _
Curly braces { }
Square brackets [ ]
Brackets ( )
Hash #
Plus +
Minus -
Period .
Exclamation point !

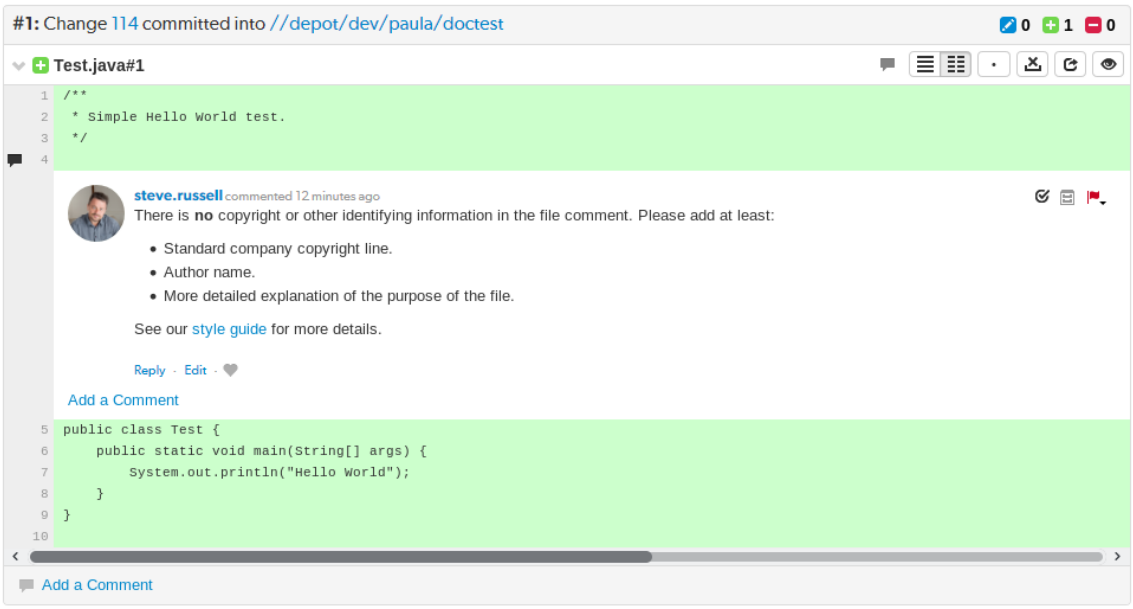
Reply · Edit ·

Markdown in comments

Markdown content is displayed in comments, but Markdown support is limited to prevent execution of raw HTML and JavaScript content.

If you use Markdown styles in your comments, Swarm renders them when you post the comment.

> **Tip**
> Try to avoid using Markdown styles like headers, images, and tables in comments, they can make comments harder to read this is especially true inline comments.



Markdown in projects

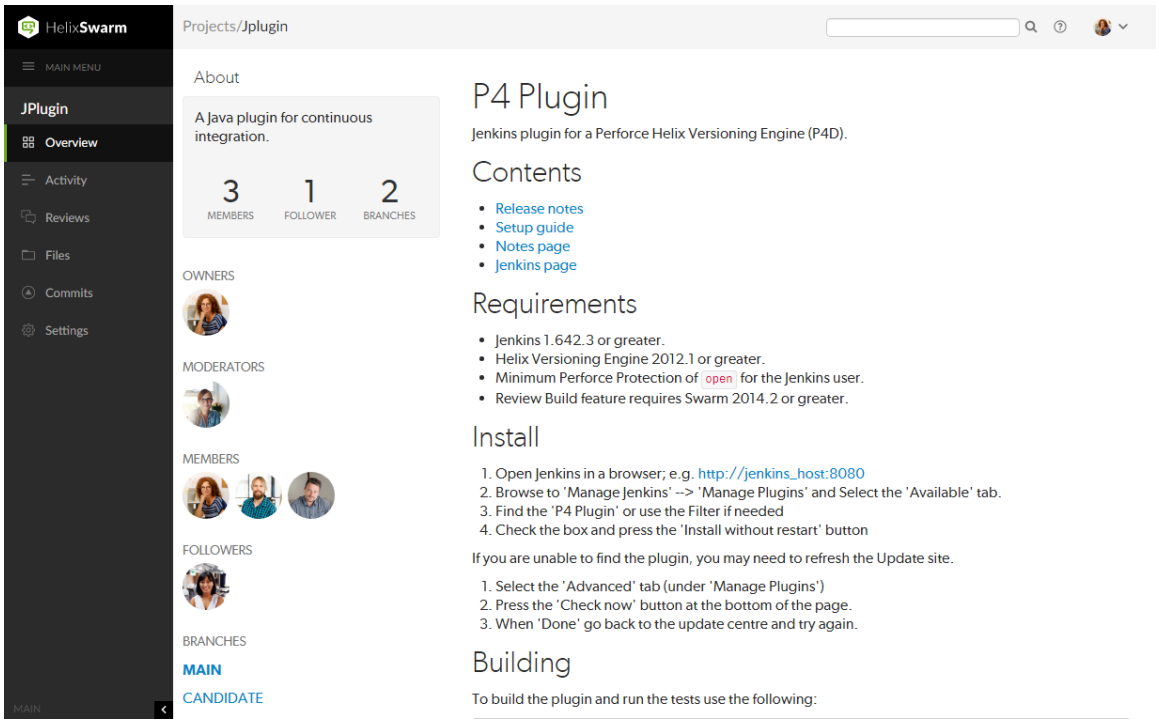
If a project has a `README` markdown file in the root of its `MAIN` branch, that `README` markdown file is displayed as the overview page for the project.

By default, Markdown content is displayed on the project overview page but is limited to prevent the execution of raw HTML and JavaScript content. This can be configured by the Swarm administrator.

- Markdown support for displaying the project overview page can be disabled by your Swarm administrator, see "Project readme" on page 538.
- The level of Markdown support can be configured by your Swarm administrator, see "Markdown" on page 512.

If you need to change which branch is considered MAIN, and therefore from where the `README` is read from, see "Mainline branch identification" on page 509.

> **Tip**
> Valid Markdown file extensions are: `md`, `markdown`, `mdown`, `mkdn`, `mkd`, `mdwn`, `mdtxt`, `mdtext`.
>
> If more than one `README` file is found, Swarm displays the first one it finds based on the order above.


HelixSwarm
Projects/Jplugin
MAIN MENU
JPlugin
Overview
Activity
Reviews
Files
Commits
Settings
About
A Java plugin for continuous integration.
3
MEMBERS
1
FOLLOWER
2
BRANCHES
OWNERS
MODERATORS
MEMBERS
FOLLOWERS
BRANCHES
MAIN
CANDIDATE
P4 Plugin
Jenkins plugin for a Perforce Helix Versioning Engine (P4D).
Contents
Release notes
Setup guide
Notes page
Jenkins page
Requirements
Jenkins 1.642.3 or greater.
Helix Versioning Engine 2012.1 or greater.
Minimum Perforce Protection of open for the Jenkins user.
Review Build feature requires Swarm 2014.2 or greater.
Install
1. Open Jenkins in a browser; e.g. http://jenkins_host:8080
2. Browse to 'Manage Jenkins' --> 'Manage Plugins' and Select the 'Available' tab.
3. Find the 'P4 Plugin' or use the Filter if needed
4. Check the box and press the 'Install without restart' button
If you are unable to find the plugin, you may need to refresh the Update site.
1. Select the 'Advanced' tab (under 'Manage Plugins')
2. Press the 'Check now' button at the bottom of the page.
3. When 'Done' go back to the update centre and try again.
Building
To build the plugin and run the tests use the following:

5 | Code reviews

A code review is a process in which other developers can see your code and provide feedback that can suggest ways to improve the code's structure, performance, maintainability, and interaction with other code.

Benefits

Some of the benefits of code review are:

- Enforcing coding standards: code reviews can catch code that does not meet your team's coding standards. This improves the readability and consistency of your codebase.
- Knowledge and experience sharing: your team can help you learn to code better. This is particularly useful for developers new to the team.
- Early defect detection: small errors can be caught before they become problems later on.
- Code sharing: code reviews spread knowledge of the current codebase, which helps both with maintaining a mental model of the overall project, as well as defending against developer absences.
- Better personal review: knowing that someone might catch a simple coding error often increases the review developers perform of their own code.

Swarm attempts to provide these benefits without adding onerous overhead for developers.

Facilities

Swarm provides the following code review facilities. In the list, the term *author* refers to the person who creates a change to be reviewed, *reviewer* refers to any authenticated Swarm user performing code review tasks, and *required reviewer* refers to a reviewer whose up-vote is required before a review can be approved.

- Authors can request reviews, and can designate reviewers and required reviewers.
- Reviewers can start a code review on existing changes.
- Reviewers can add themselves to a review to indicate that they are participating in the review and sharing responsibility for the review.
- Reviewers can add comments to a changelist, to a specific file in a changelist, or to a specific line in a file, using Markdown text.
- Reviewers can vote on a review, to indicate their approval or disapproval.
- Required reviewers can prevent a review from becoming approved until they up-vote the review.
- Project branches with assigned moderators limit review approval to one of the moderators.

- Reviews spanning multiple project branches with assigned moderators, limit review approval to one moderator from any one of the branches by default.

  > **Note**
  > Swarm can be configured to require that one moderator from each branch must approve the review. For more information about moderator behavior, see "Moderators" on page 379.

- Reviewers can mark changes as needing revision, approved, rejected, or to be archived for future consideration.
- Reviewers can commit approved changes if necessary.

Workflow

By default, Swarm's code review workflow is basically advisory in nature with very few restrictions imposed on the code review workflow. However, Swarm does provide a number of mechanisms to structure or restrict code review workflows, such as:

- A required reviewer, which is any user designated by a review author, project member or moderator, or is any authenticated user that joins a review and makes their vote required, can prevent reviews from being approved until they up-vote the review. See "Required reviewers" on page 380 for details.
- Branch moderators, when configured for one or more branches in a project, prevent reviews from being approved (or rejected) without their involvement. For more information about moderators, see "Moderators" on page 414.
- Administrators can optionally configure Swarm to prevent reviews with open tasks being approved or, approved and committed, see "Prevent Approve for reviews with open tasks" on page 553.
- **Workflow:** on by default:
  - Global workflow rules can be configured to enforce a minimum code review workflow on the entire Helix Core server. For more information on global workflows, see "Workflow basics" on page 425.
  - Global workflow rules can be configured to enforce a minimum code review workflow on projects, and branches that don't have an associated workflow. For more information on global workflows, see "Workflow basics" on page 425.
  - Individual workflows can be configured by users and can be associated with projects, and branches to enforce that code review workflow on them. For more information on workflow, see "Workflow basics" on page 425.

Agile development teams should find sufficient capability within Swarm to make code reviews a regular part of their workflow. Swarm's development team has been using it regularly during development of Swarm. If you have ideas and suggestions for improvement, please contact us.

Models

There are three code review models: pre-commit, post-commit, and the Git Fusion model. Which model you use for code reviews with Swarm is up to you.

Pre-commit model

The pre-commit model is possible due to Helix server's *shelving* feature. Shelving enables you to temporarily make copies of your files available to other users without committing the changes into the depot. Shelving can be a very handy way for developers to create a backup, or to handle local workspace changes that might otherwise lose work in progress, without having to commit code that might destabilize a codebase.

Swarm uses the shelving feature in Helix server to manage code reviews. Shelving allows reviewers to easily acquire a copy of the code to be reviewed, and allows updates to the reviewed code prior to submission.

For more information on shelving, see Shelve Changelists in *Helix Core Server User Guide*.

Post-commit model

The post-commit model can be used if your team's development processes preclude the use of shelving. Code must be committed to the Helix server before code review can begin, which reduces the opportunity to fix problems before, for example, a continuous integration system notices problems. However, code reviews can be started for any existing code regardless of how long it has been committed.

Git Fusion model

Perforce Git Fusion provides repo management for Git repositories, and provides workflows that enable Git and Helix server users to collaborate on the same projects using their preferred tools.

The Git Fusion model is similar to the pre-commit model; changes in your local repo can be pushed for review to a named Helix server branch in the Git fusion repo configuration, making your proposed changes available so that others can review and comment on them prior to committing them to the target branch. Git Fusion and Swarm work together to create a review branch and container for the pre-commit collaboration.

The Git Fusion model has several limitations that you should be aware of:

- The target branch for Git Fusion-created reviews must be a fully populated branch, and must be listed in the repo-specific Git Fusion configuration.

  See Setting up Repos in the *Git Fusion Guide* for details on converting a lightweight branch into a fully populated Helix server branch.
- Reviews created with Git Fusion can only be updated from Git Fusion.

- You cannot clean up history and then push your changes to the same review. If you perform a Git rebase, you should push your changes as a new review.
- A Git Fusion review does not currently display the individual task branch commits that make up the review. Only the merged commit diffs are shown.

For more information on Git Fusion, see the *Git Fusion Guide*.

Internal representation

Swarm-managed changelists

> **Note**
> **Helix server 2020.2 and later:** any new file being shelved that has the same content as an existing shelved file refers to the existing archive file instead of creating a duplicate archive file. No Helix server or Swarm configuration is required for this feature.
>
> This Helix server feature automatically reduces the space required for the Swarm-managed shelved review changelists. Swarm creates these changelists for its own internal use. Helix server only updates new shelves, it does not retrospectively update your existing shelves.

A code review consists of one or more shelved changelists that Swarm manages. A shelved changelist is a pending changelist that has a snapshot of its files on a shelf associated with the changelist.

When a review is started, Swarm creates a new changelist that becomes the review changelist. What happens afterwards varies:

- If the review contains uncommitted work (the pre-commit model), Swarm copies the shelved files from the user's changelist that initiated the review into the review's changelist.
- Any time that a user's changelist associated with the review has its shelved files updated, Swarm copies the shelved files into its review changelist and creates an archive changelist. An archive changelist is no different from any other pending changelist with shelved files, but it allows Swarm to provide versioning and diffs within a review.
- If the head version of a review is committed (the post-commit model), the review's changelist is emptied of files.

The review's changelist is never actually committed; this allows the review to be opened later with additional shelved changes.

> **Important**
> **Swarm's managed review changelists should only be deleted if you are uninstalling Swarm.**
>
> Swarm's review changelists maintain the history of a review and all of its feedback. The deletion of a Swarm shelved changelist causes instability and potentially data loss, and represents a scenario that can be very challenging to recover from, even with the engagement of Perforce consultants.

You can display a list of all of the Swarm-managed changelists using the **p4 changelists** command:

```
$ p4 changelists -u swarm
Change 1212285 on 2015/07/31 by swarm@swarm-96017af4-5615-9819-7af1-
6fc1fa537214 *pending* 'Add requirements and instructions'
Change 1212284 on 2015/07/31 by swarm@swarm-96017af4-5615-9819-7af1-
6fc1fa537214 *pending* 'Add requirements and instructions'
...
```

swarm is the userid with *admin*-level privileges within the Helix server that Swarm is configured to use. Use the appropriate userid when you run the **p4 changelists** command.

Swarm-managed workspaces

Whenever Swarm creates a changelist for a review, it uses a client workspace (or just workspace) associated with the configured Helix server userid that has *admin* privileges. Whenever a user commits a change via Swarm's user interface, Swarm uses a workspace associated with that user.

To learn more about workspaces, see Helix server as a version control implementation in the *Solutions Overview: Helix Version Control System* Guide.

The workspaces that Swarm creates and uses live in the *SWARM_ROOT*/**data/clients** folder.

Inside the clients folder, Swarm maintains a user-specific folder that contains any workspace folders that may be required. Each user-specific folder is named by converting their Helix server userid into hexadecimal to avoid any characters that would be problematic in the filesystem, such as slashes, accents, UTF-8 characters, etc. For example, the folder for the user steve.russell would be named **73746576652e72757373656c6c**.

Within the user-specific folder are the folders that become the root of each workspace. Each of these folders is named with a globally-unique identifier (GUID) prefixed with **swarm-**, for example **swarm-438d482b-f107-9a35-c06c-86ac68136b00**. Accompanying each folder is a lock file with the same name plus the .lock extension. Finally, the user-specific clients folder contains a management lock file called **manage.lock**.

Here is an example of the folder structure:

```
SWARM_ROOT/
        data/
                clients/
                  73746576652e72757373656c6c/
                        manage.lock
                        swarm-438d482b-f107-9a35-c06c-86ac68136b00/
                        swarm-438d482b-f107-9a35-c06c-86ac68136b00.lock
                        swarm-8388362a-233d-0cb9-3e90-895eaaa99f6c/
                        swarm-8388362a-233d-0cb9-3e90-895eaaa99f6c.lock
```

```
                    7061756c612e626f796c65/
                          manage.lock
                          swarm-da7de4b4-0ecb-12c8-1b35-f3e32bb18033/
                          swarm-da7de4b4-0ecb-12c8-1b35-f3e32bb18033.lock
```

Here are the steps Swarm takes when it needs to use a client:

1. Convert the current connection's userid to hexadecimal.
2. Check to see whether a user-specific folder exists within *SWARM_ROOT*`/data/clients`; if not, create the folder.
3. Within the user-specific folder, loop over any existing workspace folders and attempt to lock each in turn:

   If a lock is acquired skip to the next step. Otherwise, perform the following procedure.

   Create workspace procedure:

   a. Check if the max number of clients for the current user has been reached:
      - If so, wait a short amount of time (50 milliseconds), and start step 3 again.
      - If not, proceed to the next step.

   b. Take a lock on `manage.lock`.

   c. Check if the max number of clients for the current user has been reached:
      - If so, release the `manage.lock`, wait a short amount of time (50 milliseconds), and start step 3 again.
      - If not, proceed to the next step.

   d. Create a new workspace folder using a GUID-based filename, and take a lock on the folder.

   e. Release the `manage.lock` lock.
4. Perform the necessary file operations using the locked workspace folder.
5. Revert the file content within the workspace folder to avoid having constantly growing disk space use.

   > **Note**
   > There may occasionally be stray files left; Swarm is not aggressive about cleaning up.

6. Swarm releases the lock on the workspace folder.

Most users should only require 1-2 workspaces, and those are only required if they commit from Swarm. The *admin* user that Swarm is configured to use should only use one workspace per configured worker.

By default, the number of workspaces that could be active at any given instant is two times the number of configured workers. Since the default worker count is three, Swarm would use at most six workspaces simultaneously.

If the workspace limit is reached, further file processing is blocked until a workspace becomes available. Potentially, this means that users could encounter timeouts. Configuring Swarm to use more workers could solve that issue.

Removal considerations

Administrators might wish to remove Swarm-managed workspaces. There are a few considerations that should be assessed prior to removal:

- Ideally, you should stop the web server (taking Swarm out of service) before removing a Swarm-managed workspace from the Swarm server; this eliminates the risk of removing a workspace that is in use.

  If you do not stop the web server first, Swarm may encounter an error during a submit.
- Removal of a Swarm-managed workspace folder does not remove the client spec from the Helix server. Unless the client spec is removed, that workspace effectively becomes orphaned. Orphaned clients are, of themselves, not a big concern as the storage and performance impact is negligible.
- Removal of a Swarm-managed workspace's corresponding client spec in the Helix server can be done. However, **you should never remove a client spec that has associated shelved files**.

  Usually, the only client specs that should have associated shelved files belong to the *admin* account that Swarm is configured to use. All other workspaces that may exist for other users are primarily used for submitting changes, and so should not have shelved files associated.

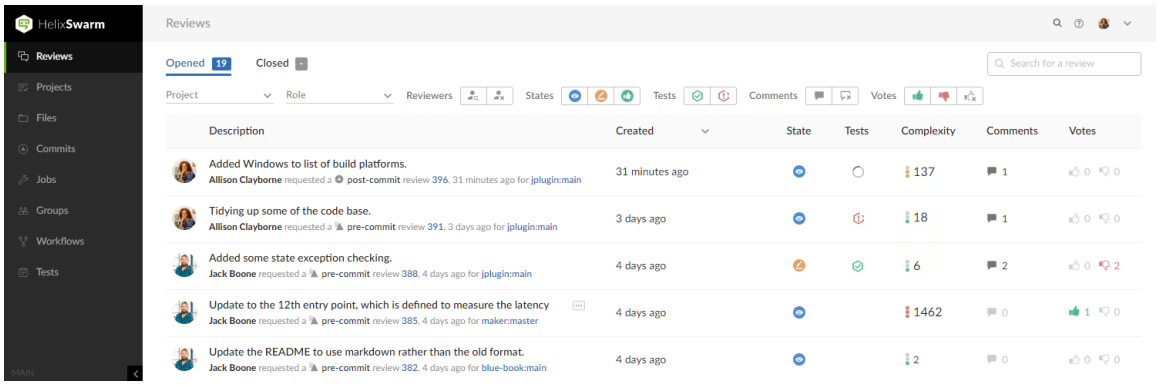
Reviews list

The code review list helps you keep track of code reviews that:

- Have been requested and are awaiting review
- Are underway
- Have been accepted, rejected, or archived

To see all available reviews, click **Reviews** in the menu.

The **Reviews** page lists open and closed reviews for all projects in the Helix Core server.



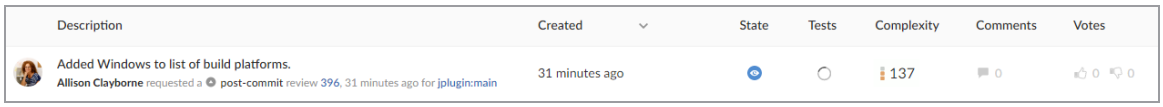
The **Opened** and **Closed** tabs display:

- **Opened** tab: displays a list of all code reviews that have started, are being reviewed, are awaiting revisions, or need to be committed.
- **Closed** tab: displays a list of all code reviews that have been approved and committed, rejected, or archived.

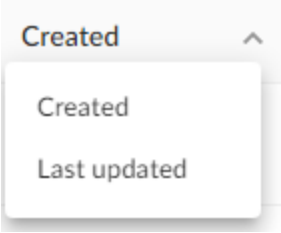
Tip

- The **Opened** and **Closed** tabs display the number of reviews in each tab. Initially this can be an over estimate which Swarm dynamically corrects as more information becomes available to it.
- The **Closed** tab initially displays a − character. The − is replaced by a number when you navigate to the **Closed** tab.

Each review displays the following information:



- **Avatar** displays the avatar of the review author, click to view the profile of the author
- **Description** contains:
  - first line of the review description, click to open the review. If the review description is too long it is truncated, click on the ellipsis ••• to expand it in the list page.
  - review author, click to view the profile of the author
  - review type, pre-commit or post-commit
  - review number and version, click to open the review
  - review creation date and time
  - review project branches, click to open the project
- **Created** or **Last activity:** when the review was created or when the review was last updated depending on the **Result order** button setting.



To change the sort order, click the **Result order** button and select **Created** or **Last activity** from the dropdown menu:

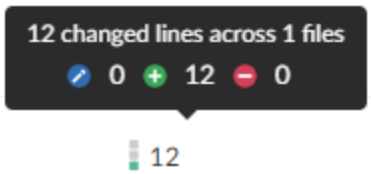
- **Created**: Reviews sorted by when they were created
- **Last activity**: Reviews sorted by when they were last updated

> **Note**
> - If the **Result order** button is not displayed reviews are sorted by when they were created.
> - **Result order** button display is a global setting controlled by the Swarm administrator. See "Reviews filter" on page 551 for details.
> - When review results are older than 24 hours they are displayed in numerical order within each day.

- **State:** shows the current review state
- **Tests:** shows the test suite state for the review, either tests in progress, tests passed, or tests failed.
- **Complexity:** a traffic light icon and number shows the relative complexity of the review and the total number of lines changed in the review. Complexity can be configured by your Swarm administrator but, by default:
  - **Red:** ≥ 300 changes
  - **Amber:** < 300 and >30 changes
  - **Green:** ≤ 30 changes

> **Tip**
> - Review complexity is only calculated for a review when the review is updated and the file content has changed.
> - Review complexity is only stored for the current revision of a review.

Hover over the complexity icon to display more detailed information:



- **Comments:** displays the number of open (non-archived) comments that are associated with the review. The icon is filled if there are comments on the review.
- **Votes:** displays the number of up votes and down votes for the review. The appropriate vote icon is filled if you have voted on the review, vote icons with only an outline show votes by others.

> **Note**
> Hover your mouse over any of the icons to see tooltips.

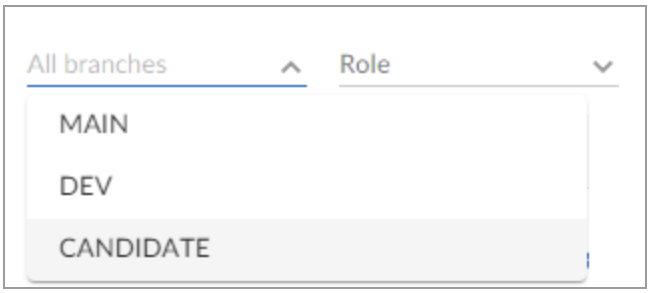
"Projects" on page 299 and "Groups" on page 291 have their own review lists that display reviews created by their members.

Review filters

> **Tip**
> Swarm updates the URL in your browser to reflect the filter options you have selected. This makes it easy to bookmark or share your review list filter settings with the URL.
>
> If you share a URL with **Comment** or **Vote** filters selected, those filters are applied to the user running the URL. This means that their reviews list page will contain different reviews to your reviews list page.

The following filtering options are available for code reviews:

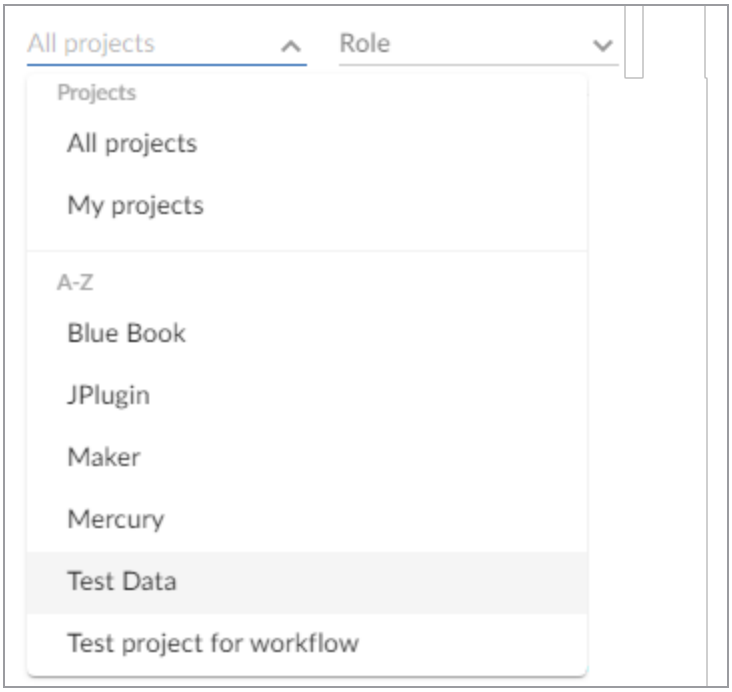
- **Branch** (only available on the project "Reviews page" on page 303): a dropdown menu that lets you filter which reviews to display based on the current project branches:



  - **All branches:** all reviews are displayed for all branches within the current project.
  - **Select branch:** selecting a branch name displays only reviews for that project branch.

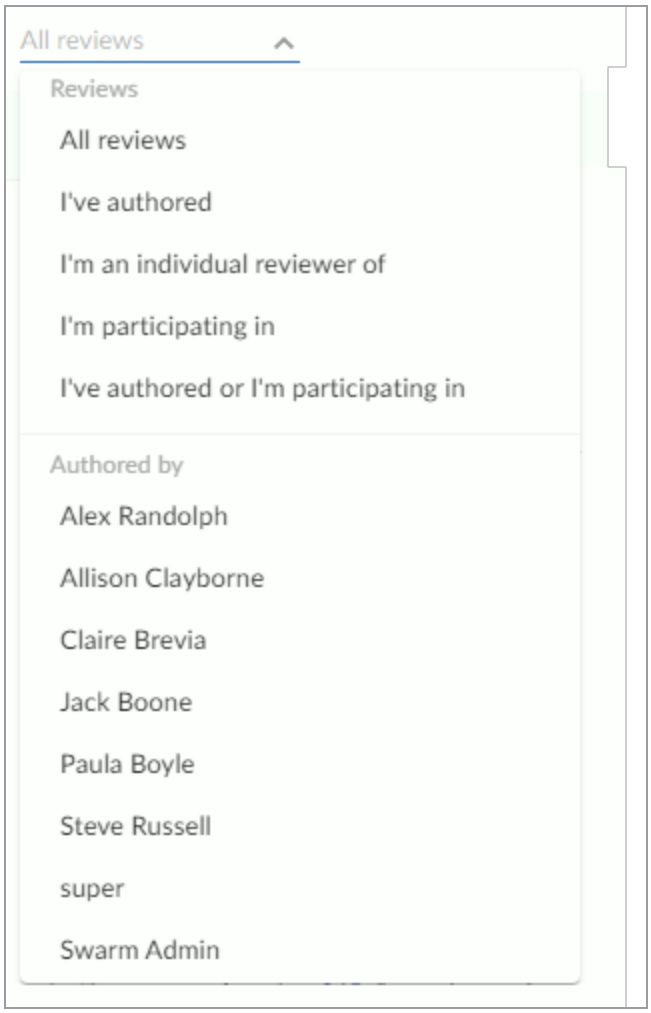
- **Branch search:** An auto-complete search field that allows you to choose one of the branches within the current project. Once specified, only reviews for the selected project branch are displayed.

- **Project:** a dropdown menu that lets you filter which reviews to display based on project:



  - **All projects:** all reviews are displayed for all projects.
  - **My projects:** all reviews for all of the projects you are participating in, as a member, owner, moderator, or follower.
  - **Select project:** selecting a project name displays only reviews for that project.
  - **Project search:** an auto-complete search field that allows you to choose one of the projects defined in Helix server. Once specified, only reviews for the selected project are displayed.

- **Role:** a dropdown menu that lets you filter which reviews to display based on user involvement:



- **All reviews:** displays all reviews.
- **Reviews I've authored**: displays reviews that you have authored.
- **Reviews I'm participating in:** displays reviews that you are a reviewer of, but not an author of.
- **Reviews I'm an individual reviewer of:** displays reviews that you are an individual reviewer of , but not a group reviewer of, or an author of.
- **Review I've authored or I'm participating In:** displays reviews that you have authored, or are a reviewer of.
- **Authored by:** an auto-complete search field that allows you to choose one of the user accounts defined in the Helix server. Once specified, only reviews authored by the user are displayed.

When you select one of the available options, the list of options updates to match the currently selected filter, and the **Role** dropdown indicates the current filter: **All**, **Author**, **Participant**, **Author or Participant**, **Individual Reviewer**, or **user**.

- **Reviewers** (**Opened** tab only):

  - **Has reviewers:** displays reviews that have one or more reviewers.
  - **No reviewers:** displays reviews that have no reviewers.

- **States** (**Opened** tab only):

  - **Needs Review:** the review's changes need to be reviewed.
  - **Needs Revision:** the review's changes have been reviewed, but further revisions are required before the review can be accepted.
  - **Approved:** the review's changes have been approved, and should be committed.

- **States** (**Closed** tab only):

  - **Approved:** the review's changes have been approved and committed.
  - **Rejected:** the review's changes have been rejected.
  - **Archived:** the review's changes have been put aside.

- **Tests:**

  - **Tests passed:** when automated tests are enabled for the associated project, and the test suite execution succeeds, Swarm updates the review accordingly.
  - **Tests failed:** similar to the Tests pass state, except that the test suite execution has failed. Check with your test suite to determine why the tests failed.

- **Comments:**

  - **I have commented on:** I have commented on the review.
  - **I have not commented on:** I am a participant but have not commented on the review.

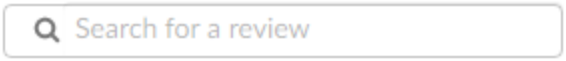
> **Note**
> Filters for commenting only apply to reviews which you are a participant of. Commenting on or voting on a review will automatically add you as a participant. If you leave the review after commenting on it, then this review will not be included in the list.

- **Votes:**

  - **I have voted up:** I have voted the review up.
  - **I have voted down:** I have voted the review down.
  - **I have not voted on:** I am a participant but have not voted on the review.

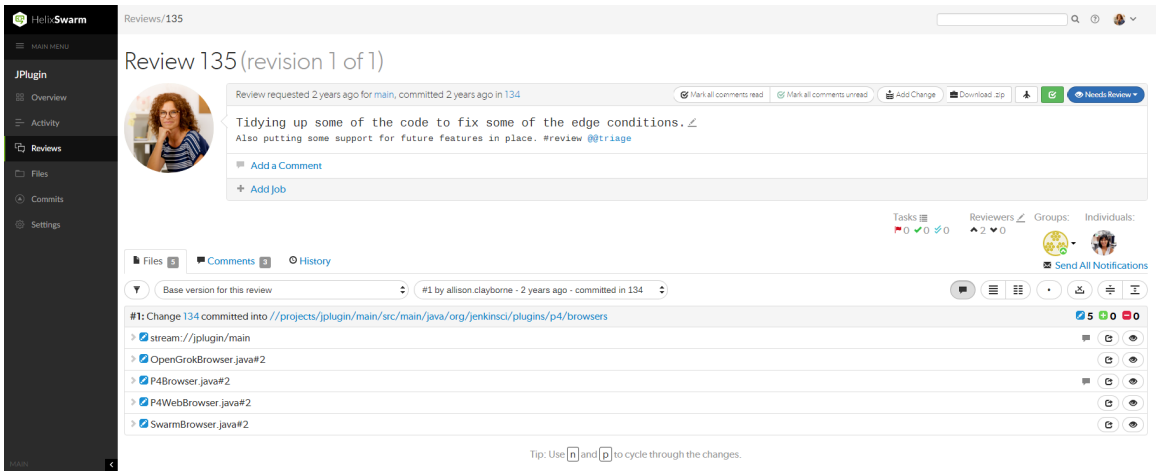
> **Note**
> Filters for voting only apply to reviews which you are a participant of. Commenting on or voting on a review will automatically add you as a participant. If you leave the review after voting on it, then this review will not be included in the list.

- **Search term:** search for a partial match of review description content and an exact match for *reviewid*, *userid*, and *projectid*.



Review display

During a code review, reviewers spend most of their time using the review interface.



Swarm supports stream specs in your workspace using the Private editing of streams feature. If a changelist or review contains a stream spec, it will be displayed first in **Files** with the prefix **stream: //**, for example: **`stream://MyStreamDepotName/MyStreamSpecLocationName`**. A changelist/review can only contain one stream spec.

The review interface is very similar to the changelist interface; and provides largely the same functionality, but has several notable differences that are described in the following sections.

Review ID

The Review ID is the unique number used to identify the Swarm review. The revision of the review being viewed and the total number of revisions available are shown in brackets.

Add Change button

Once a review has been started you can add a changelist to the review. It can be useful to add changelists to an existing review. For example, if follow up changes are made to files in a review or if you need to group a number of changelists under a single review.

The **Add Change** button in the review heading is used to add a changelist to an existing review. The button is not visible if the review is in a state that is protected from change by the review workflow rules. For details of the **On update of a review in an end state** rule, see "Workflow rules" on page 423.

The options available in the **Add Change** dropdown menu depend on whether the review is pre-commit or post-commit:

- **Pre-commit reviews:**
  - **Append Pending Changelist:** when you add a pending changelist to a review, the files in the changelist are appended to the existing files in the review, see "Append a pending changelist to a review" on page 373.

> **Note**
> A review can only contain 1 stream spec. If you append a changelist with a stream spec to a review that already contains a stream spec, the spec in the changelist replaces the original one in the review. If it is a different spec from the original spec in the review, Swarm cannot display the diff between them and displays **File content unchanged**.

- **Replace with Pending Changelist:** when you add a pending changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a pending changelist" on page 375.
- **Replace with Committed Changelist:** when you add a committed changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a committed changelist" on page 377.

> **Note**
> If you replace a pre-commit review with a committed changelist, the new version of the review will be a post-commit review.

- **Post-commit reviews:**
  - **Replace with Pending Changelist:** when you add a pending changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a pending changelist" on page 375.

> **Note**
> If you replace a post-commit review with a pending changelist, the new version of the review will be a pre-commit review.

  - **Replace with Committed Changelist:** when you add a committed changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a committed changelist" on page 377.

> **Tip**
> When the content of a review is changed, Swarm checks to see which branches are in the new revision of the review:
>
> - **If a new branch was added to the review:**
>   - Default reviewers on the new branch are added to the review.
>   - Moderators from the added branch become moderators for the review alongside the existing moderators.
>   - **Only if workflow is enabled:** if the new branch is associated with a workflow, the workflow is merged with the existing workflow. The most restrictive workflow is used for the review.
> - **If a branch is no longer part of the review:**
>   - Reviewers for the review are not changed.
>   - Moderators from the removed branch no longer moderate the review.
>   - **Only if workflow is enabled:** if the branch was associated with a workflow, the branch workflow is removed from the review.

Download .zip button

The **Download .zip** button is used to download a ZIP archive containing all of the files in the review. The version of the files downloaded is the most recent review revision selected in the review revision selector, see "Select review revisions to view" on page 361.

> **Note**
> The **Download .zip** button is not displayed if the zip command-line tool is not installed on the Swarm server. For information about installing, and configuring the zip command-line tool, see Zip archive.

When you click the **Download .zip** button, Swarm performs the following steps:

1. Scans the files/folders:
   - Checks that you have permission to access their contents, according to the Helix Core server protections.
   - Checks that the total file size is small enough to be processed by Swarm.
2. Syncs the file contents to the Swarm server from the Helix Core server.
3. Creates the ZIP archive by compressing the file content.
4. Starts a download of the generated ZIP archive.

> **Note**
> - You might not see all of the above steps; Swarm caches the resulting ZIP archives so that repeated requests to the same files/folders can skip the sync and compress steps whenever possible.
> - If an error occurs while scanning, syncing, or compressing, Swarm indicates the error.

Deployment status

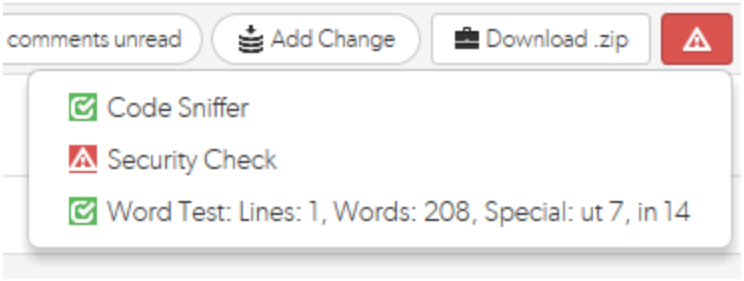
When automated deployment has been configured for a project, the deployment success  or failure  is indicated in the review's heading. If your deployment program can provide a URL that provides details of the deployment, the indicator becomes linked; click the indicator to see the deployment results.

Test status

When the review has an associated workflow with tests configured on it or when continuous integration is configured for the project, Swarm displays the current test status for the review revision in the review heading:

- **Tests in Progress:** displayed if any of the tests for the review revision are running.
- **Tests Passed:** displayed if all of the tests for the review revision have passed.
- **Tests Failed:** displayed if any of the tests for the review revision have failed.

To view test run information for tests associated with the review revision, click the **Test status** to open the dropdown list:



If your continuous integration system calls back to Swarm with a URL, the test run in the dropdown will be linked to the URL provided.

Review state button

The **Review state** button displays the current state of the review, and is also used to change the state of the review.

For information on the review states and changing the review state, see "States" on page 387 and "Change review state" on page 390.

Edit description

Click the **Edit Description** icon in the review description to update the description to reflect any updates have been made during the review.

Update pending changelist checkbox

Only available if you are editing the description of a pre-commit review and you are the original author of the changelist that created the review.

Select the **Update pending changelist** checkbox in the **Edit Description** dialog to also apply your review description changes to the original changelist description.

Tasks

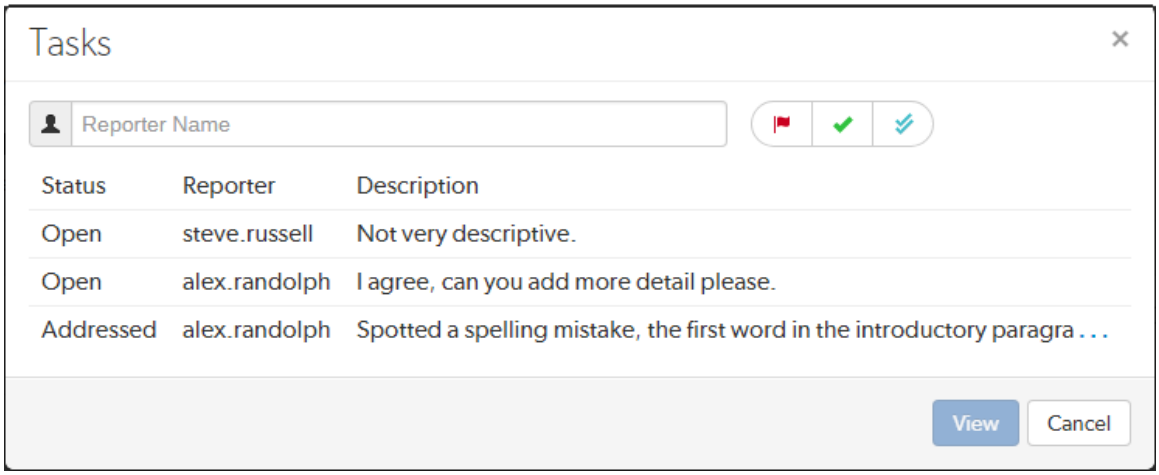
Comments can be flagged as tasks, for more details on working with tasks see Tasks.

Note
If Swarm is configured to prevent approval of reviews with open tasks and a review has open tasks, the **Approve**, and **Approve and Commit** options will not be available for the review. This option is configured by an administrator, see "Prevent Approve for reviews with open tasks" on page 553.

To approve, or approve and commit a review with open tasks, you must address the tasks first and then set them to **Task Addressed**, or **Not a Task**, see "Set a task to Task Addressed or Not a Task" on page 270 for details.

A summary of the number and status of comments flagged as tasks is displayed below the code review description. Archived comments that are flagged as tasks are not included in the summary.

Click the **Task list** button to display a dialog listing all tasks associated with the review:



Within the **Tasks** dialog, you can filter the tasks by the reporter (the userid of the user who created the task), and/or by task state:

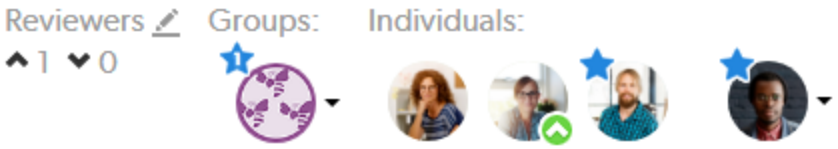
- Click the **Red Flag** button to display only open tasks (comments that need to be addressed).
- Click the **Green Check Mark** button to display only addressed tasks (comments that have been addressed).
- Click the **Blue Double-check Mark** button to display only verified tasks (comments that have been addressed and verified).

Note
Archived tasks do not appear in the Tasks dialog.

To view a particular task, select the task in the **Tasks** dialog, and click the **View** button. Alternately you can double-click on the task in the **Tasks** dialog. The **Tasks** dialog closes and Swarm adjusts the review display so that the chosen task is in view.

Reviewers

A *Reviewers* area appears below the review's description whenever a review has one or more reviewers, or you are logged in.

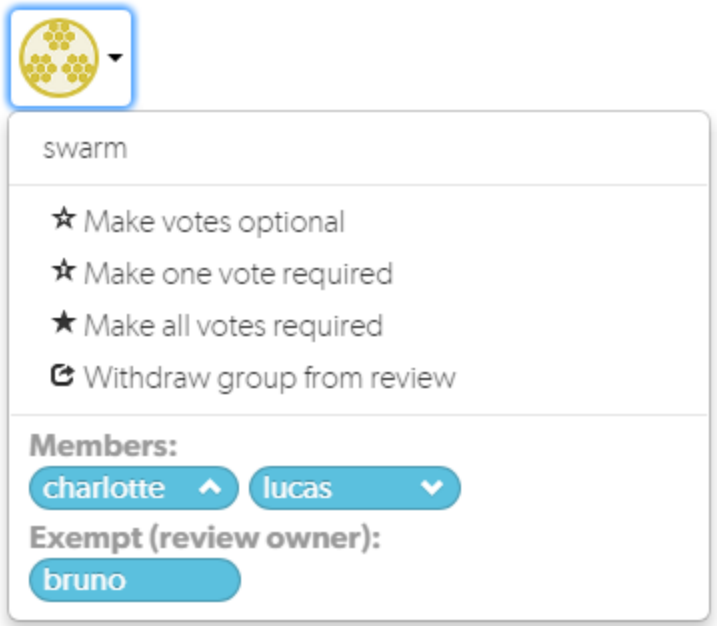


This area includes, from left to right, the edit reviewers button, the current up and down vote counts, the avatars of current reviewer groups, and the avatars of the current reviewers.

> **Note**
> By default, reviewer group members are not displayed in the *Individuals* area of the reviews page when they interact with a review (vote, comment, update, commit, archive, etc.). This avoids overloading the *Individuals* area with individual avatars if you have large reviewer groups.
>
> An exception to this behavior is when a member of a reviewer group is also an individual required reviewer, in this case their avatar will be displayed in the *Individuals* area.
>
> See "Expand group reviewers" on page 558 for details on displaying reviewer group members when they interact with a review.

Group reviewer

If you are a member of a group that is a reviewer on the review, click your individual avatar to vote on the review. See the "Individual reviewer" on page 357 table for details. Your vote is registered for the group, and is also displayed on your individual avatar. Click on the group avatar to see how individual members have voted on the review:



You must be the review author, a project member, a project moderator, or a user with super privileges to change the group settings:

> **Note**
> If the review includes content that is part of a project or branch with Retain default reviewers enabled, the following restrictions apply to the review:
>
> - The voting option for a retained default reviewer can only be changed to a stricter level, you cannot reduce the voting level.
> - Retained default reviewers cannot be removed from the review.

- **Make votes optional:**
  - If any group member votes down the review, the group avatar displays a badge indicating that the group has voted down the review.
  - If at least one group member votes up the review and **no** members of the group vote down the review, the group avatar displays a badge indicating that the group has voted up the review.
- **Make one vote required:** Indicated by a star badge with a 1 over the group avatar.
  - If any group member votes down the review, the group avatar displays a badge indicating that the group has voted down the review.
  - If at least one group member votes up the review and **no** members of the group vote down the review, the group avatar displays a badge indicating that the group has voted up the review. See Required reviewers for details.

- **Make all votes required:** Indicated by a star badge over the group avatar.
  - If any group member votes down the review, the group avatar displays a badge indicating that the group has voted down the review.
  - If all the group members vote up the review and **no** members of the group vote down the review, the group avatar displays a badge indicating that the group has voted up the review. See Required reviewers for details.
- **Withdraw group from review:** Removes the group from the review.

Individual reviewer

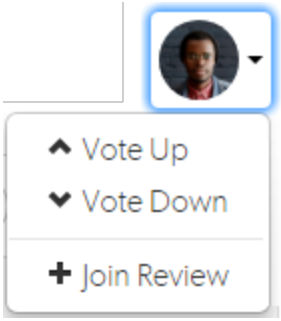
When an individual reviewer has voted on a review, their avatar displays a badge indicating whether they voted up or down. Required reviewers have a star badge over their avatar.

If you are logged in and viewing a review you did not author, your avatar appears to the right of any other reviewers and its appearance varies according to the following conditions:

> **Note**
> If the review includes content that is part of a project or branch with Retain default reviewers enabled, the following restrictions apply to the review:
>
> - The voting option for a retained default reviewer can only be changed to a stricter level, you cannot reduce the voting level.
> - Retained default reviewers cannot be removed from the review.

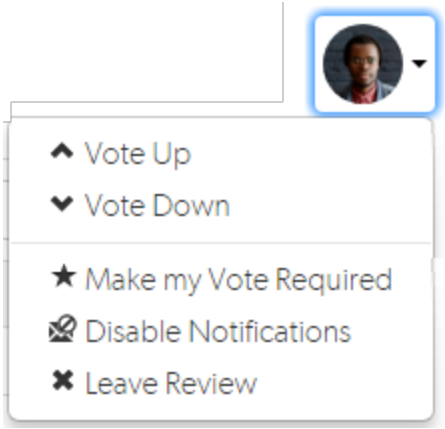
When you are not yet a reviewer, or you are a member of a reviewer group who has not yet voted: Clicking your avatar presents a menu allowing you to vote up or down and thereby become an individual reviewer, or simply join the review as an individual reviewer without voting.

When a member of a reviewer group votes they automatically become an individual reviewer as well.



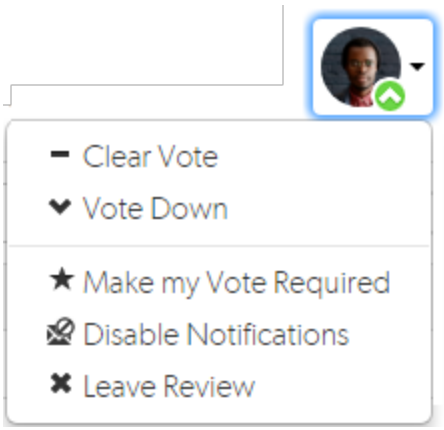
When you are an individual reviewer who has not yet voted: No badge appears on your avatar and clicking your avatar presents a menu allowing you to vote up or down, change whether your individual vote is required or not, or leave the review.

If you are also a member of a reviewer group and you click **Leave Review**, you will remain a member of the reviewer group.



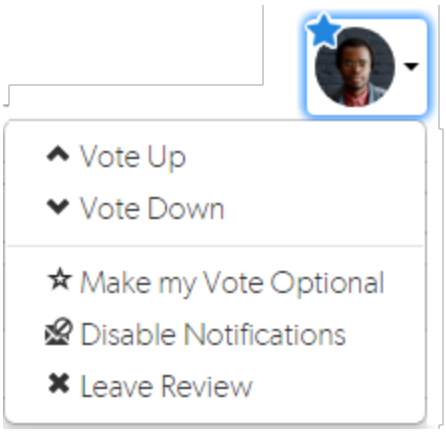
When you are an individual reviewer who has voted, or a member of a reviewer group who has voted: Your avatar displays a badge indicating your vote. Clicking your avatar presents a menu allowing you to clear or change your vote, change whether your individual vote is required or not, or leave the review.

If you are also a member of a reviewer group and you click **Leave Review**, your vote is cleared but you will remain a member of the reviewer group.



When you are a required reviewer, or a member of reviewer group that requires all votes: Your avatar displays a star badge indicating that the review cannot be approved until you vote up. Clicking your avatar presents a menu allowing you to change your vote, make your vote optional, or leave the review.

If you are also a member of a reviewer group and you click **Leave Review**, you will remain a member of the reviewer group. **Make my Vote Optional** is not available for members of "all votes required" reviewer groups.



Review approval

A review can be approved when the following requirements are met:

- There are no down votes on the review.
- All of the Required reviewers on the review have voted up.
- The **Minimum up votes** on the review has been satisfied for **each** of the projects and branches the review spans.

Use the **Review state** button to approve the review.

Tip

- If the review is moderated, only the moderator can approve the review. For more information about review moderation, see "Moderators" on page 379.

- **Automatic approval of reviews:**
  - If the **Automatically approve reviews** workflow rule is enabled for the review's project/branch, the review is automatically approved as soon as the review requirements are satisfied.
  - If the review has a moderator, the review will not be automatically approved. The moderator must approve the review manually.

  For more information about the **Automatically approve reviews** rule, see "Workflow rules" on page 423.

Stale votes

When a review is updated, if the review's list of files, file content, or file-types changes, any votes cast on the review become *stale*. The vote counts are reset, and the vote indicators become muted.

If you hover your mouse over a reviewer with a stale vote, a tooltip appears displaying the *userid*, how they voted, and on which version of the review; each version is represented as a point on the "Select review revisions to view" on the next page.

Note
Stale vote handling is not supported for Git-created reviews.

Disable notifications

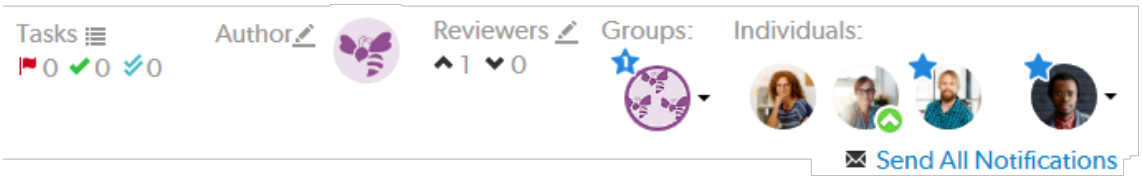
When you become a review participant, by joining the review or being @mentioned in a comment or in the review's description, you receive notifications for any events associated with the review. If you find that the notifications become more of a burden than benefit and you wish to continue being a review participant, you can disable notifications:

1. Click your avatar in the reviewers area to display your reviewer options.
2. Click **Disable Notifications**.

Once notifications are disabled, you no longer receive notifications. However, if you are @mentioned in a subsequent review comment, you do receive a notification for that comment; regular notifications remain disabled. This approach ensures that you don't miss anything that other reviewers or the review author deems important.

Changing the review author

If the configuration option to allow users to change the author of a review is enabled, then an extra icon is displayed to the left of the Tasks and Reviewers icons. By default this option is disabled, talk to your Swarm Administrator if you require this feature.



If the edit author button is displayed, clicking the edit button will open a dialog that allows the author of this review to be changed. This is useful if the original author is no longer available, or ownership has passed to a different developer.

Select review revisions to view

The review revision selectors are used to specify which revisions of a review you want to *diff*, they are located in the **Files** tab above the list of files.

- **Left dropdown selector:** the base version for the review is selected by default, base is the version of the file that was checked out of the depot before it was changed for this review. The current version in the depot, sometimes called Head can be selected, if there are multiple revisions of the review a specific revision can be selected.
- **Right dropdown selector:** the latest revision of the review is selected by default. If there are multiple revisions of the review, a specific revisions of the review can be selected.

> **Tip**
> If a review consists of one or more Swarm-managed changelists. When comparing versions of a review, Swarm is showing any differences between the selected versions, not the review author's personal changelist. See "Internal representation" on page 339 for details.

For information on viewing *diffs*, see "Diffs" on page 261.

Filter comments button

For involved reviews that have many revisions and many comments, it can sometimes be difficult to determine whether the points expressed in comments have been addressed. To help with such situations, click the **Filter comments** button to limit the displayed comments to the current review version.

Files tab

The file listing header displays:

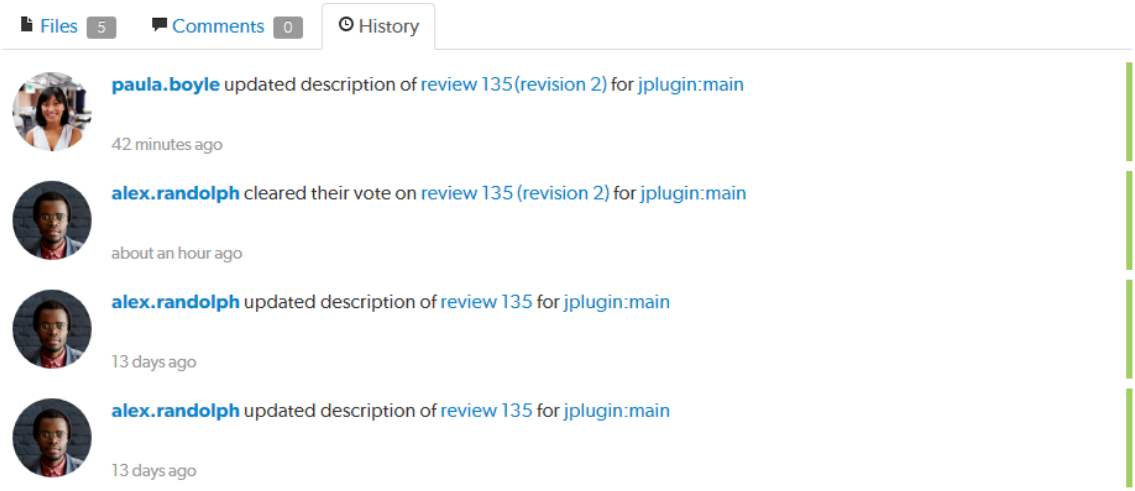
- The current version of the files in the review.
- Which changelist contains a shelved copy of the review's files.
- The common path for all of the files in the review.

In diff mode, the file listing header displays:

- Which two versions of the review's files are being compared.
- Which changelists contain the files being compared.
- The common path for all of the files in both versions of the review.

History tab

The **History** tab presents a list of the events that affect this review.



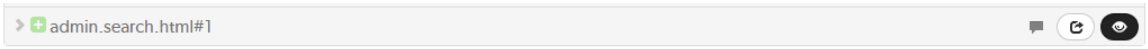
Events include:

- When the review was started
- When a new reviewer joins the review
- When the review's state changes
- When the review's files are updated
- When a reviewer votes on the review
- When someone comments on the review, or one of its files
- When tests pass or fail, provided continuous integration is configured

Mark file as read

Beside each file in a review is a **Mark file as read** button  , which help you keep track of which files you have reviewed. The read flag is remembered independently for each user. If the content of a file is changed in an update to the review, the read flag automatically clears. This is particularly useful when a code review consists of many files.

When clicked, the button color inverts and the associated file is visually muted, to make it easy to distinguish read files from unread files:



If a file has been marked as read, click the button a second time to reset the status to unread.

Identifying reviews created with Git Fusion

Git Fusion-initiated reviews include the Git logo beside the main review identifier. This indicator is important because Helix server users cannot update Git Fusion-initiated reviews.

Activities

This section describes the major activities that affect code reviews, including starting a review, updating a review, and fetching a review's files.

Start a review

> **Important**
> If your Helix server is configured as a commit-edge deployment, and your normal connection is to an edge server, Swarm refuses to start reviews for shelved changes that have not been promoted to the commit server.
>
> Within Swarm, this means that the **Request Review** button does not appear for unpromoted shelved changes. Outside of Swarm, attempts to start reviews for unpromoted shelved changelists appear to do nothing. Ask your Helix server administrator for assistance if you cannot start a review.
>
> An administrator of the Helix server can automatically promote shelved changes to the commit server by setting the configurable `dm.shelve.promote` to `1`.

> **Tip**
> If your changelist only contains a stream spec and its location in the Helix server is not associated with a Swarm project, the review that you create will not have any default reviewers or workflow rules. To associate the review with a project so that it has default reviewers and obeys the project workflow rules, include a file change from the project path in the changelist when you create the review.

1. Use the P4D command line or a P4D client to create the shelved or committed changelist.
2. Start a code review by using one of the following approaches:

Use Swarm:

1. Use Swarm to view a shelved or submitted changelist.

   > **Tip**
   > To view a shelved or submitted changelist, use a Quick URL. For example, if your change is **54321**, visit the URL: `https://myswarm.url/54321`.

2. Click the **Request Review** button to request a review of that changelist.

   Requesting a review on a shelved changelist uses the pre-commit model and requesting a review on a submitted changelist uses the post-commit model.

Use P4D command line:

When you are about to shelve or submit files:

1. Include `#review` within your changelist (separated from other text with whitespace, or on a separate line).

   Once the review begins, Swarm replaces `#review` with `#review-12345`, where `12345` is the review's identifier.

   > **Note**
   > The `#review` keyword is customizable. For details, see "Review keyword" on page 546.

2. At this time, you can add reviewers to the code review by using @mention for users, and @@mention for groups in the changelist description for each desired reviewer.

   If your `@mention` or `@@mention` includes an asterisk (`*`) before the *userid* or *groupid*, for example `@*userid`, that user or all of the group members become required reviewers. If your `@@mention` includes an exclamation mark (`!`) before the *groupid*, for example `@@!groupid`, the members of that group become required reviewers but only one member of the group is required to vote. See "Required reviewers" on page 380 for details.

3. Complete your shelve or submit operation.

   > **Warning**
   > If you shelve a changelist and subsequently edit the description to include `#review`, a review is **not started**. You must re-shelve the files after adding `#review`.

Use Git Fusion:

1. When you are using Git Fusion, you can start a review by pushing your changes to a target branch using the following command:

   ```
   $ git push origin task1:review/master/new
   ```

   `task1` is the name of the current Git task branch, and *`master`* is the target branch that the proposed changes are intended for.

   > **Important**
   > The target branch must be mapped to a named Helix server branch in the Git Fusion repo configuration.
   >
   > See the Converting a lightweight branch into a fully-populated branch section of the *Git Fusion Guide* for details.

2. When the command completes, the output indicates the *`review id`* that has been created:

   ```
   remote: Perforce: Swarm review assigned: review/master/1234
   ```

where `1234` is the review id that was just created.

For more information on Git Fusion, see the *Git Fusion Guide*.

Tip
You can also start a Swarm review with P4V, P4VS, and P4Eclipse. See below for details:

- **P4V:** see the Swarm integration features section of the *P4V User Guide*.
- **P4VS:** see the Managing files chapter of the *P4VS User Guide*.
- **P4Eclipse:** see the Reviewing changes chapter of the *P4Eclipse User Guide*.

Note
If you are using P4V and its Swarm integration, and you encounter the error **`Host requires authentication`**, ask your Helix Core server administrator for assistance. See "P4V Authentication" on page 477 for details.

Update a review

To update a code review, use one of the following approaches:

- For a pre-commit review that you authored:
  1. Edit the files
  2. Shelve the files

  You can repeat these steps as many times as necessary.
- For a post-commit review, or a review where you are not the author:
  1. Fetch the review's files into a new changelist
  2. Edit the files
  3. Update the changelist's description to include **`#review-12345`** (separated from other text with whitespace, or on a separate line)
  4. Shelve the changelist's files

  Once these steps are complete, further updates involve editing the files, and then shelving the changelist's files.

  Warning
  If you use an invalid review identifier, it will appear that nothing happens. Swarm is currently unable to notify you of this situation. If the review has not been correctly updated, use the **Add Change** button in the Swarm review heading to add the changelist to the review, see "Add a changelist to a review" on page 372.
- When you are working with Git Fusion:

> **Important**
> You can only update Git Fusion-initiated reviews using Git Fusion.

In the following example, the current Git task branch is ***task1***, the target branch is ***master***, the review id is ***1234***, the Git Fusion hostname is ***gfserver***, and the remote repo name is ***p4gf_repo***.

1. Fetch the review's head version:

   ```
   $ git fetch --prune origin
   From gfserver:p4gf_repo
   * [new_branch]      review/master/1234 ->
   origin/review/master/1234
   x [deleted]         (none)     -> origin/review/dev/new
   ```

   The **--prune** option lets the local Git repo delete the unwanted **review/*master*/new** reference created by the initial **git push origin *task1*:review/*master*/new** command.

2. Check out the review's head version:

   ```
   $ git checkout review/master/1234
   ```

3. Edit the files as required.
4. Add the edited files to the index of files, in preparation for the next commit.

   There are several ways to do this. For example, to add all modified files to the index, run:

   ```
   $ git add -A
   ```

5. Commit the files in Git:

   ```
   $ git commit -m "made some changes"
   ```

6. Push the Git changes to the review:

   ```
   $ git push origin review/master/1234
   ```

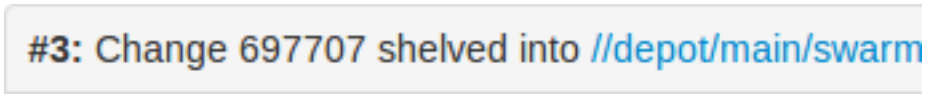
   > **Note**
   > If you get review feedback that is better expressed as a Git rebase and cleaned up history, you can make your changes and push them as a new review.
   >
   > You **cannot** clean up history and then push your changes to the *same* review.

   For more information on Git Fusion, see the *Git Fusion Guide*.

Fetch a review's files

First, determine the changelist containing the review's files:

1. Visit the review's page.
2. The current review version's changelist appears in the file list heading:

   In this example, the changelist is `697707`. You use the identified changelist in place of shelved changelist below.
3. Decide whether you will use P4, P4V, or Git Fusion to fetch the files, and follow the instructions in the appropriate section below.

Using P4

1. For a **shelved changelist**, use a command-line shell and type:

   ```
   $ p4 unshelve -s shelved changelist
   ```

2. For a committed changelist, use a command-line shell and type:

   ```
   $ p4 sync @committed changelist
   ```

> **Note**
> Your client's view mappings need to include the changelist's path.

Using P4V

For a shelved changelist:

1. Select **Search > Go To**.
2. Change the select box to **Pending Changelist**.
3. Type in the *shelved changelist* number and click **OK**.
4. Select the files in the **Shelved Files** area.
5. Right-click and select **Unshelve**.
6. Click **Unshelve**.

For a committed changelist:

1. Select **Search > Go To**.
2. Change the select box to **Submitted Changelist**.
3. Type in the *submitted changelist* number and click **OK**.

4. Select the files in the **Files** area.
5. Right-click and select **Get this Revision**.
6. Click **Close**.

Using Git Fusion

In the following example, the current local task branch is ***task1***, the target branch is ***master***, the review id is ***1234***, the Git Fusion hostname is ***gfserver***, and the remote repo name is ***p4gf_repo***.

1. Fetch the review's head version:

   ```
   $ git fetch --prune origin
   From gfserver:p4gf_repo
   * [new_branch]      review/master/1234 ->
   origin/review/master/1234
   x [deleted]         (none)     -> origin/review/dev/new
   ```

   The **--prune** option lets the local Git repo delete the unwanted **review/*master*/new** reference created by the initial git **push origin *task1*:review/*master*/new** command.

2. Check out the review's head version:

   ```
   $ git checkout review/master/1234
   ```

Important
You can only update Git Fusion-initiated reviews using Git Fusion.

For more information on Git Fusion, see the *Git Fusion Guide*.

Deleting shelves

If you have a number of unused shelves, you can delete them if you want to tidy up your workspace.

Important
By default, when you delete files from a shelved changelist, the files are not removed from the associated review.

However, Swarm can be configured to remove files from a review when they are deleted from an associated shelf, see "Process shelf file delete when" on page 555.

Do not use the Swarm user that is configured in the Swarm configuration file when deleting shelves, or deleting files from shelves. The Swarm logic processes the *shelve-delete* trigger event, if the event is invoked by the Swarm user it is rejected. The delete operations will fail.

To delete a shelf from a changelist without removing the shelved files from the associated review, use P4 or P4V to delete the entire shelf in one operation as described below. This enables you to delete a shelved changelist in your workspace without affecting the associated Swarm review even if Swarm is configured to remove files from a review when they are deleted from a shelf.

Using P4

To delete a shelved changelist without removing the files from the associated review:

1. Use a command-line shell and type:

   ```
   $ p4 shelve -d -c changelist
   ```

   > **Important**
   > Do not delete the files from the shelf individually. Deleting the entire shelf ensures that files are not removed from an associated review even if Swarm is configured to do so.

2. The shelved files are deleted from the pending changelist.

Using P4V

To delete a shelved changelist without removing the files from the associated review:

1. In P4V, right-click on the pending changelist.
2. Select **Delete Shelved Files**.

   > **Important**
   > Do not delete the files from the shelf individually. Deleting the entire shelf ensures that files are not removed from an associated review even if Swarm is configured to do so.

3. The shelved files are deleted from the pending changelist.

Edit reviewers

A review author can edit the reviewers for a review by using the **Edit reviewers** button  on the review page. Reviewers are able to join or leave reviews, or to change whether their vote is required or optional.

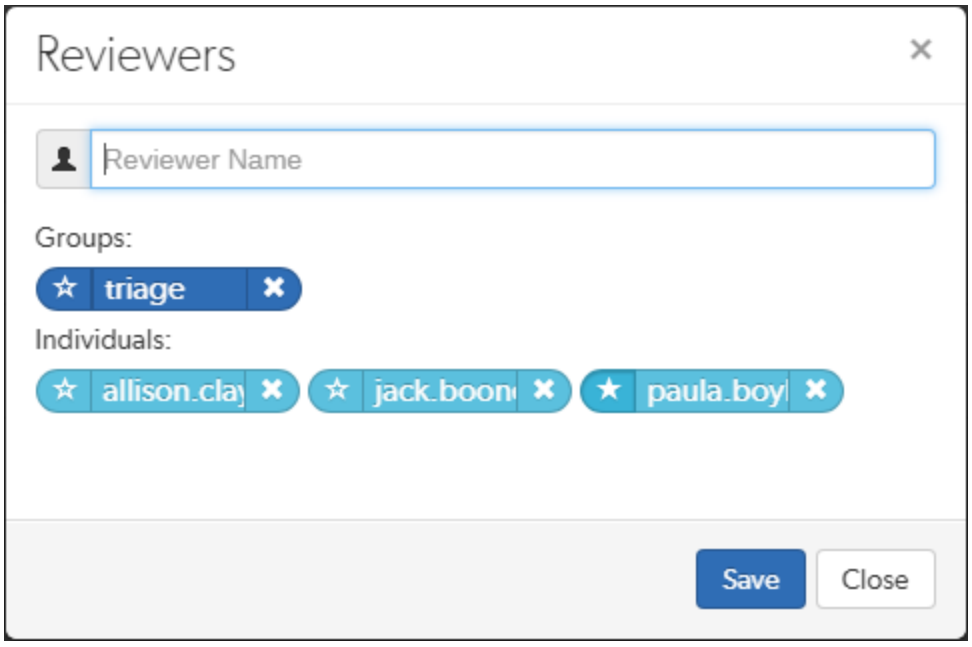
> **Note**
> If the review includes content that is part of a project or branch with Retain default reviewers enabled, the following restrictions apply to the review:
>
> - The voting option for a retained default reviewer can only be changed to a stricter level, you cannot reduce the voting level.
> - Retained default reviewers cannot be removed from the review.

Additionally, the following individuals can edit reviewers:

- Users with *admin* or *super* privileges.
- If the review is moderated, the moderators.
- If the review is part of a project, but not moderated, all project members.
- If the review is not part of a project, any authenticated user.

To edit reviewers for a review:

1. Navigate to a review.
2. Click the edit reviewers button, which appears just to the left of reviewer avatars. The **Reviewers** dialog is displayed.



3. Add or remove reviewers, or change the vote requirement.

   Use the reviewer search field to find users and groups by name, *userid*, *groupid*. The field auto-completes as you type, click on the user or group to add to the review.

   - **Groups:** click the star icon to the left of the *groupid* , and select whether the group is a required reviewer (one vote), a required reviewer (all votes), or an optional reviewer. A solid star means that all group member votes are required to approve a review, a solid star with a 1 inside means at least one group member must vote up and no group members vote down to approve a review, and the outlined star means that the group vote is optional.
   - **Users:** click the star icon to the left of the *userid* to toggle whether their vote is required or not. A solid star means that their vote is required to approve a review, whereas the outlined star means that their vote is optional.

Click the **X** icon to the right of the *userid* or *groupid* to remove that reviewer from the review.

4. Click the **Save** button to save any changes made.

Add a changelist to a review

Once a review has been started you can add a changelist to the review. It can be useful to add changelists to an existing review. For example, if follow up changes are made to files in a review or if you need to group a number of changelists under a single review.

The **Add Change** button in the review heading is used to add a changelist to an existing review. The button is not visible if the review is in a state that is protected from change by the review workflow rules. For details of the **On update of a review in an end state** rule, see "Workflow rules" on page 423.

The options available in the **Add Change** dropdown menu depend on whether the review is pre-commit or post-commit:

- **Pre-commit reviews:**
  - **Append Pending Changelist:** when you add a pending changelist to a review, the files in the changelist are appended to the existing files in the review, see "Append a pending changelist to a review" on the next page.

    > **Note**
    > A review can only contain 1 stream spec. If you append a changelist with a stream spec to a review that already contains a stream spec, the spec in the changelist replaces the original one in the review. If it is a different spec from the original spec in the review, Swarm cannot display the diff between them and displays **File content unchanged**.

  - **Replace with Pending Changelist:** when you add a pending changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a pending changelist" on page 375.
  - **Replace with Committed Changelist:** when you add a committed changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a committed changelist" on page 377.

    > **Note**
    > If you replace a pre-commit review with a committed changelist, the new version of the review will be a post-commit review.

- **Post-commit reviews:**
  - **Replace with Pending Changelist:** when you add a pending changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a pending changelist" on page 375.

    Note
    If you replace a post-commit review with a pending changelist, the new version of the review will be a pre-commit review.

  - **Replace with Committed Changelist:** when you add a committed changelist to a review, all of the files in the review are replaced with the files in the changelist you are adding to the review, see "Replace review with a committed changelist" on page 377.

Tip
When the content of a review is changed, Swarm checks to see which branches are in the new revision of the review:

- **If a new branch was added to the review:**
  - Default reviewers on the new branch are added to the review.
  - Moderators from the added branch become moderators for the review alongside the existing moderators.
  - **Only if workflow is enabled:** if the new branch is associated with a workflow, the workflow is merged with the existing workflow. The most restrictive workflow is used for the review.
- **If a branch is no longer part of the review:**
  - Reviewers for the review are not changed.
  - Moderators from the removed branch no longer moderate the review.
  - **Only if workflow is enabled:** if the branch was associated with a workflow, the branch workflow is removed from the review.

Append a pending changelist to a review

The **Append Pending Changelist** option is used to add a pending changelist to a pre-commit review.

Example:

- Original review contains the following files: $A^{\#1}$, $B^{\#2}$, $C^{\#1}$, $D^{\#1}$, and $E^{\#4}$
- Changelist contains the following files: $A^{\#2}$, $C^{\#3}$, $E^{\#4}$ (marked for delete), and $F^{\#1}$
- Appending the changelist to the original review generates a new revision of the review that contains the following files: $A^{\#2}$, $B^{\#2}$, $C^{\#3}$, $D^{\#1}$, $E^{\#4}$ (marked for delete), and $F^{\#1}$

Important

Committing a review with Swarm (recommended): Swarm automatically commits the files in the approved revision of the review.

Committing a review outside of Swarm:

Before you commit the review:

1. Unshelve the review into the pending changelist associated with the review.
2. Reshelve the files in the pending changelist.
3. Commit the pending changelist.
4. This process ensures that all of the files in the approved revision of the review are committed.

   Workflow feature enabled **(default):**

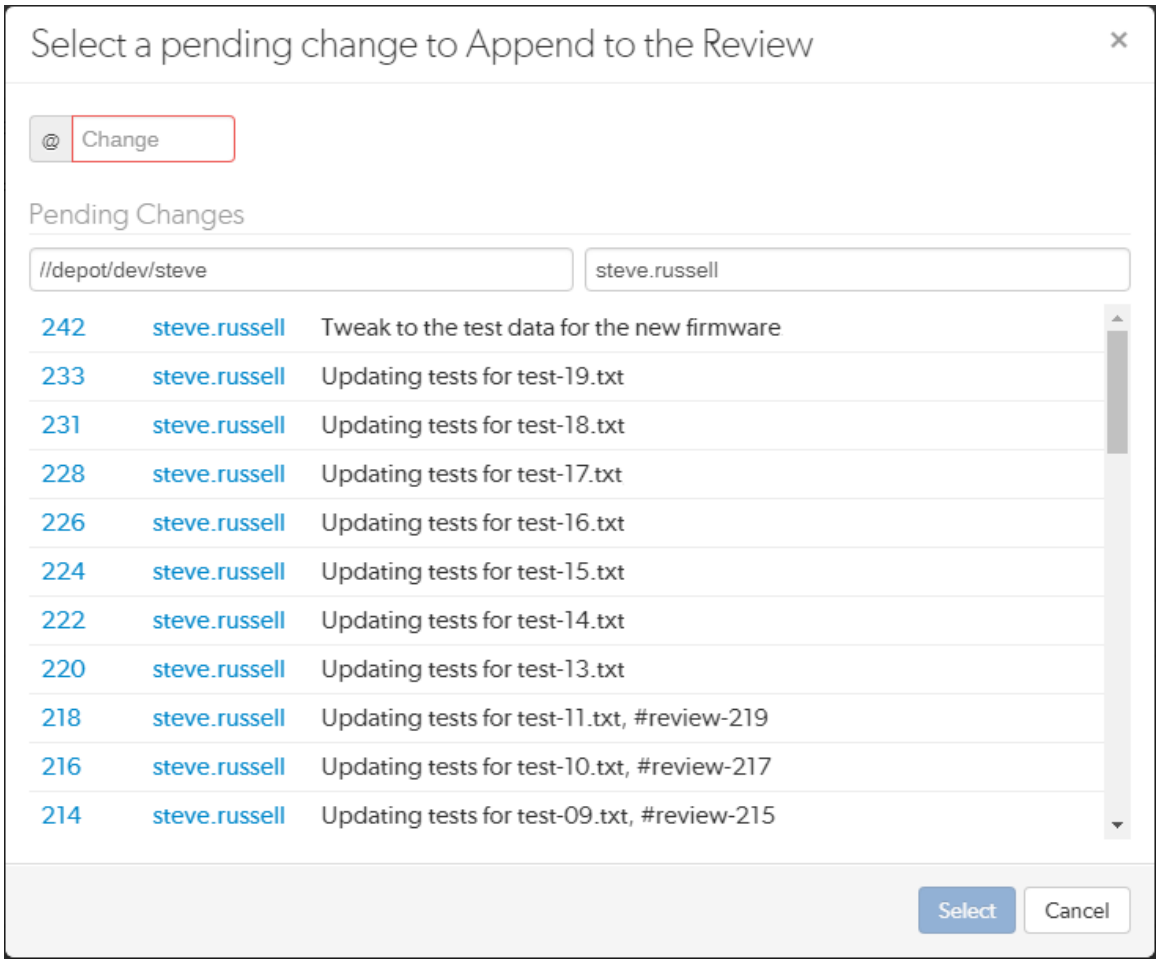
   Swarm can be configured to automatically check that the files being committed match the files in the approved revision of the review by using the **On commit with a review** workflow rule. For information about the **On commit with a review** rule, see "Workflow rules" on page 423.

   Workflow feature disabled**:**

   Swarm can be configured to automatically check that the files being committed match the files in the approved revision of the review by using the strict trigger option.

To append a changelist to a review:

1. Click **Add Change** from the review heading.
2. Select **Append Pending Changelist** from the dropdown menu and the append review dialog is displayed with a list of recent pending changelists:



3. Enter the pending changelist number directly in the **@ Change** box, filter by depot path and user, or select a changelist from the list of recent changelists.

> **Note**
> The changelist must not be part of another review, if it is Swarm will reject it.

4. Click **Select** to append the changelist to the review, double-click a changelist to do the same.
5. The changelist is appended to the review and the review version is updated.

Replace review with a pending changelist

The **Replace with a Pending Changelist** option is used to replace the files in a review with the files in a pending changelist.

> **Note**
> If you replace a post-commit review with a pending changelist, the new version of the review will be a pre-commit review.

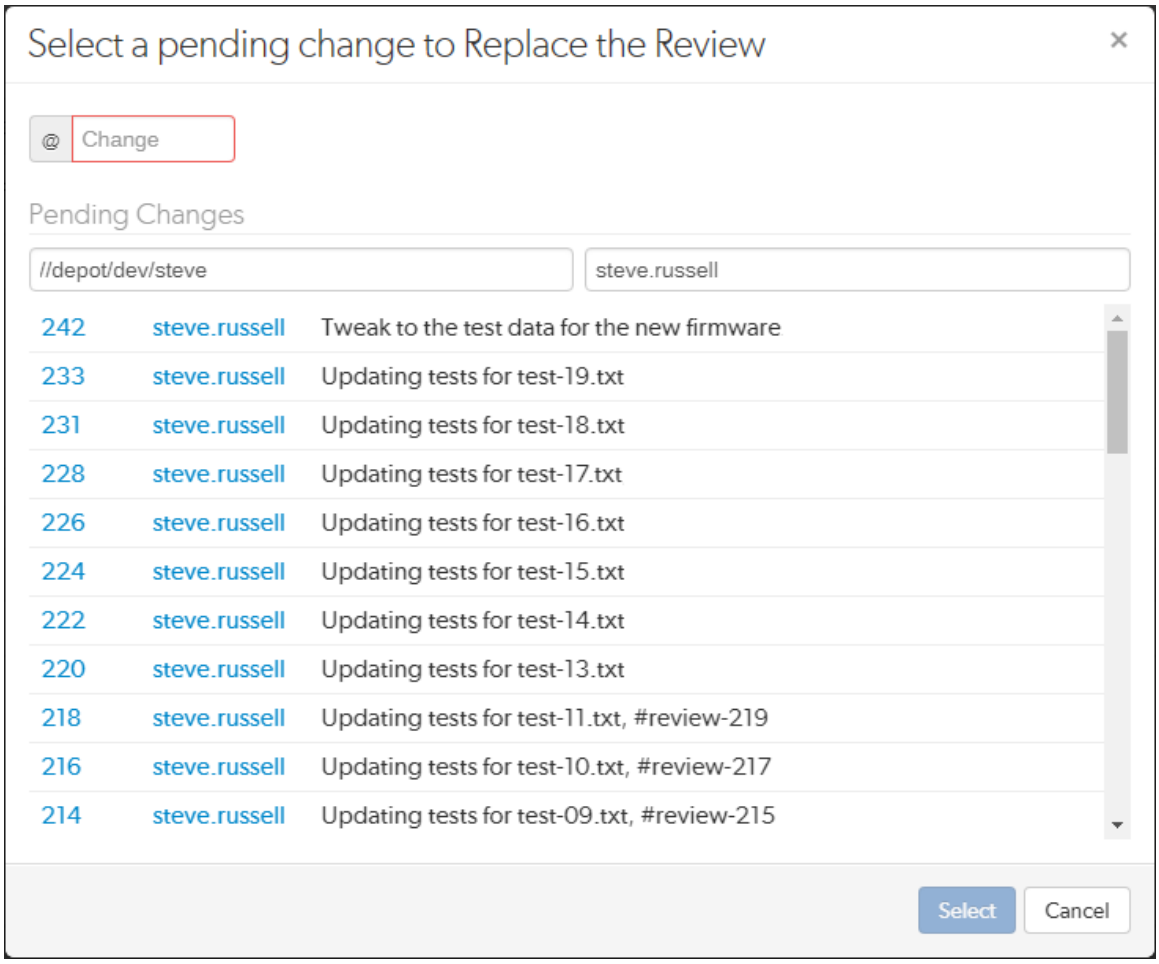
Example:

- Original review contains the following files: A#1, B#2, C#1, D#1, and E#4
- Changelist contains the following files: A#2, C#3, E#4 (marked for delete), and F#1
- Replacing the original review with the changelist generates a new revision of the review that contains the following files: A#2, C#3, E#4 (marked for delete), and F#1

> **Tip**
> When you replace a review with a changelist, the base versions of the files in the new revision of the review are the base versions of the files in the replacement changelist.

To replace the files in a review with the files in a changelist:

1. Click **Add Change** from the review heading.
2. Select **Replace with a Pending Changelist**from the dropdown menu and the replace review dialog is displayed with a list of recent pending changelists:



3. Enter the pending changelist number directly in the **@ Change** box, filter by depot path and user, or select a changelist from the list of recent changelists.

> **Note**
> The changelist must not be part of another review, if it is Swarm will reject it.

4. Click **Select** to replace the review with the changelist, double-click a changelist to do the same.
5. The changelist files replace the files in the review and the review version is updated.

Replace review with a committed changelist

The **Replace with a Committed Changelist** option is used to replace the files in a review with the files in a committed changelist.

Note
If you replace a pre-commit review with a committed changelist, the new version of the review will be a post-commit review.

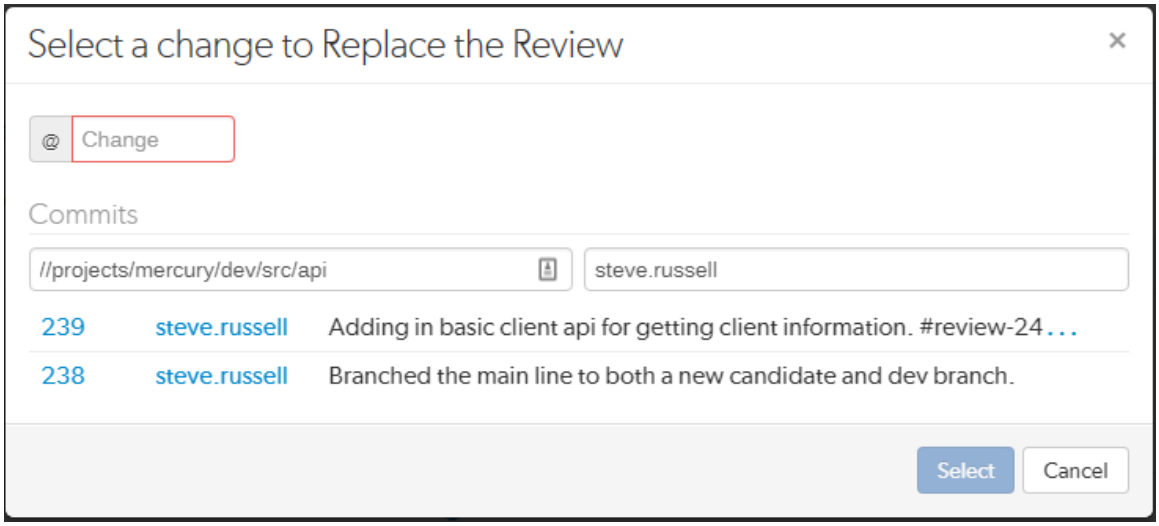
Example:

- Original review contains the following files: A#1, B#2, C#1, D#1, and E#4
- Changelist contains the following files: A#2, C#3, E#4 (marked for delete), and F#1
- Replacing the original review with the changelist generates a new revision of the review that contains the following files: A#2, C#3, E#4 (marked for delete), and F#1

Tip
When you replace a review with a changelist, the base versions of the files in the new revision of the review are the base versions of the files in the replacement changelist.

To replace the files in a review with the files in a changelist:

1. Click **Add Change** from the review heading.
2. Select **Replace with a Committed Changelist** from the dropdown menu and the replace review dialog is displayed with a list of recent committed changelists:



3. Enter the committed changelist number directly in the **@ Change** box, filter by depot path and user, or select a changelist from the list of recent changelists.

Note
The changelist must not be part of another review, if it is Swarm will reject it.

4. Click **Select** to replace the review with the changelist, double-click a changelist to do the same.
5. The changelist files replace the files in the review and the review version is updated.

Responsibility

Initially, code reviews have no reviewers.

User and group default reviewers can be set for individual projects and project branches. Each time a new review is created in the project or project branch, the default reviewers will be added to the review. See default reviewers for details.

Code review authors can designate users as reviewers by including an @mention for each desired reviewer, and an @*mention for each required reviewer. They can also designate groups as reviewers by using @@mention for each desired group reviewer, an @@*mention for each required reviewer group (all members required), and an @@!mention for each required reviewer group (one vote required). Code review authors can also designate users or groups as reviewers by using the **Edit Reviewers** dialog which allows reviewers to be added, removed, and to configure the reviewer type.

Other users can show their interest in participating in the code review by clicking their own avatar in the **Reviewers** area of the code review and selecting **Join review**, or by commenting on a review or one of its files. Once a user shows such interest, they are added to the review's list of reviewers and share in the responsibility of performing the code review. Later on, reviewers can change whether their vote is required or optional, or leave a review (perhaps to prevent further notifications).

Looking at a review list can help you determine which reviews have likely not been started, using the **No Reviewers** filter. Once a review has reviewers, it is considered to be active and appears in the review list with the state **`Has Reviewers`**.

Review participation is *advisory* by default, and is used to inform your team that a code review is being conducted. The disposition of the review is reflected in the review's current state, the badges that may appear over each reviewer's avatar, and any comments reviewers might add.

Moderators

A project moderator is a user assigned to moderate reviews for a specific branch associated with a project.

When the **Only Moderators can approve or reject reviews** restriction is enabled for a project branch, that branch is *moderated*. See Add a project for details on adding moderators to project branches.

Changing the state of any review associated with a moderated branch is restricted as follows:

- Only moderators can approve or reject the review. Moderators can also transition a review to any other state.

> **Important**
> Moderators prevent the automatic approval of reviews, for more information about automatically approving reviews using workflow rules see "Workflow rules" on page 423.

> **Note**
> By default, when a review spans multiple branches that have different moderators, only one moderator from any one of the branches needs to approve the review.
>
> Swarm can be configured to require that one moderator from each branch must approve the review, this is a global setting. If a moderator belongs to more than one of the branches spanned by the review, their approval will count for each of the branches they belong to. For instructions on how to configure moderator behavior, see "Moderator behavior when a review spans multiple branches" on page 553.

- The review's author, when not a moderator, can change the review's state to **Needs Review**, **Needs Revision**, **Archived**, and can attach committed changelists.

  Normally, the review's author cannot change the review's state to **Approved** or **Rejected** on moderated branches. However, authors that are also moderators have moderator privileges, and may approve or reject their own review.

  When `disable_self_approve` is enabled, authors who are moderators (or even users with admin privileges) cannot approve their own reviews.
- Project members can change the review's state to **Needs Review** or **Needs Revision**, and can attach committed changelists. Project members cannot change the review's state to **Approved**, **Rejected**, or **Archived**.
- Users that are not project members, moderators, or the review's author cannot transition the review's state.
- For the review's author and project members, if a review is not in one of their permitted states, for example if the review's state is **Rejected**, they cannot transition the review to another state.

These restrictions have no effect on who can start a review.

Required reviewers

Reviews can optionally have required reviewers. When a review has required reviewers, the review cannot be approved until all required reviewers and required reviewer groups have up-voted the review. If the review is associated with a project that has assigned moderators, even the moderators cannot approve the review without up-votes from all required reviewers (but they can reject the review).

When a group is a required reviewer, it can be set to operate in one of two ways:

- **All votes required:** all members of the group must up-vote the review to allow the review to be approved.

- **One vote required:** at least one member of the group must up-vote the review to allow the review to be approved. If any member of the group down-votes the review, the review cannot be approved.

Required reviewers are expected to take greater care while performing a review than non-required reviewers, as their votes affect whether a review can be approved or not.

To edit the reviewers for a review, and to change whether a reviewer is required or not, see "Edit reviewers" on page 370.

> **Note**
> If a review involves a branch with assigned moderators, only a moderator can approve the review, even if all required reviewers have up-voted the review.
>
> See Add a project for details on adding moderators to project branches.

Add yourself as a reviewer

1. Visit the review's page.
2. Log in, if you have not already done so.
3. Click your avatar in the **Reviewers** area of the review display, which should be grayed out since you are not yet a reviewer. A dropdown menu appears.
4. Select **+ Join Review**. Alternatively, you can select **^ Vote Up** or **v Vote Down** if you approve or disapprove of the review, respectively; either will cast your vote and make you a reviewer.

Your avatar is no longer grayed out, and you are now a reviewer.

Remove yourself as a reviewer

1. Visit the review's page.
2. Log in, if you have not already done so.
3. Click your avatar in the **Reviewers** area of the review display, which should not be grayed out since you are already a reviewer. A dropdown menu appears.
4. Select **X Leave Review**.

Your avatar is now grayed out, and you are no longer a reviewer.

Review workflow

There are many possible code review workflows. The section describes typical scenarios for a code review that Swarm can handle.

Basic review workflow

This section describes a basic workflow when reviewing code with Swarm.

Another developer reviews your code

1. You request a code review with a shelved change.
2. Another developer, Bill, sees the email notification from Swarm, clicks the review link in the email, and begins looking at the diffs in the files belonging to the review. Curious about an implementation detail, Bill clicks the line he's curious about and adds his query in a Swarm comment.
3. You receive an email notification regarding Bill's query. His query prompts you to clarify the code, say by renaming some variables and adding some better descriptive text in the surrounding code comments. You then update the review with your changes.
4. Bill sees the email notification that you have updated the review. He checks out the change, likes what he sees, and marks the review **Approved**.
5. You see the email notification that Bill has approved your review, so you commit your code.

You review the code from another developer

1. Another developer, Charlie, requests a code review with a shelved change.
2. You receive an email notification from Swarm, click the review link in the email, and begin looking at the diffs in the files in the review. You don't like what you see, as Charlie has tried to fix a bug using a technique you have already tried previously and know to be incorrect. You add comments to the code that needs attention, flag your comments as tasks, and mark the review **Needs Revision**.
3. Charlie receives an email notification regarding your review, but disagrees with you, and adds his own comments justifying his implementation.
4. You receive an email notification regarding Charlie's comment. The technique is somewhat complicated, so rather than attempt to describe how it is incorrect, you unshelve the review's code to your own workspace, change Charlie's code, and shelve your changes. Swarm updates the review with your new code.
5. Charlie receives an email notification regarding your updates to the review. He's still unconvinced, but he unshelves your changes to try them in his local workspace. He finds that your implementation works better, but sees a couple of areas where there could be improvements. He reshelves his latest work to update the review.
6. You receive an email notification regarding Charlie's updates, check out his changes, and realize that Charlie's work is now moot because the customer has revised his plans. You add a comment to the review reporting that fact, and **Reject** the review.

Additional workflow tasks

This section describes additional workflow tasks that can be used with the basic workflow described above.

Remove a file from a review

> **Important**
> Swarm must be configured to use this function, see "Process shelf file delete when" on page 555.

When configured, Swarm will automatically remove files from a review when they have been deleted from an associated shelved changelist and the review is in a specified state. Typically Swarm is configured to remove files from a review when the review state is **Needs Review** and **Needs Revision** but not when it is any other state.

To remove a file from a review using P4V:

1. Right-click the shelved file you want to remove from the review.
2. Select **Delete**, and click **Yes** when prompted to confirm the deletion.
3. The file is removed from the associated review if the review is in one of the states specified in the Swarm configuration.

> **Tip**
> If you want to tidy up your workspace by deleting an unused shelf without removing the files from an associated review, see "Deleting shelves" on page 369.

Revert the content of a review to match an earlier review revision

If the content of the latest revision of a review contains changes you want to rollback, or is created by mistake, you can revert the review content to match an earlier revision of the review. This section describes the process for reverting the content of a review.

> **Tip**
> - Reverting the content of a review increments the review revision, it does not change the review revision to the revision you are reverting to.
> - It is good practice to update the review description, or add a comment, so that users know that you have reverted the review with files from an earlier revision.

To revert the content of a review:

1. Find the changelist number for the review revision you want to revert to.

   > **Tip**
   > The changelist numbers in a review can be found using the review revision selector on the review page, see Select review revisions to view.

2. Unshelve the changes from the changelist you found in the previous step.
3. Create a new changelist and shelve the changes into the new changelist.
4. Use Swarm to navigate to the review you want to revert.
5. Click **Add Change** in the review heading, and select **Replace with Pending Changelist** from the dropdown list.
6. The **Select a pending change to Replace the Review** dialog is displayed.
7. Select the new changelist you shelved the files in and click **Select**.
8. The new changelist files replace the files in the review and the review revision is updated.

Fix a review if it has been replaced with the wrong changelist

If you or another user has replaced the files in your review with the wrong changelist, you can fix the mistake so that the review contains the correct files.

For example:

1. Create a pending changelist with files $A^{\#1}$, and $B^{\#1}$, this is changelist 250.
2. Request a review of changelist 250 using Swarm by clicking the **Request Review** button on the changelist page.
   - Revision 1 of review 251 is created containing files $A^{\#1}$, and $B^{\#1}$.
3. Go to review 251 in Swarm, click the **Add Change** button, select **Replace with Committed Changelist**, and select changelist 241. Changelist 241 contains files $X^{\#1}$, and $Y^{\#1}$.
   - This replaces the files in the review with the files in changelist 241.
   - Revision 2 of review 251 is created, containing only files $X^{\#1}$, and $Y^{\#1}$.
4. You realize that you have made a mistake, you meant to append changelist 242 to review 251.

Fix the mistake:

Review-251 should contain files $A^{\#1}$, and $B^{\#1}$from changelist 250 and files $C^{\#1}$, and $D^{\#1}$ from changelist 242.

Tip
It is good practice to update the review description, or add a comment, so that users know why you have changed the files in the review.

1. Go to review 251 in Swarm, click the **Add Change** button, select **Replace with Pending Changelist**, and select changelist 242. Changelist 242 contains files C#1, and D#1.
   - This replaces the files in the review with the files in changelist 242.
   - Revision 3 of review 251 is created, containing files C#1, and D#1.
2. Edit the description of changelist 250 to change `#review-251` to `#append-251`, this changes the add mode for review-251 to append.
3. Shelve your files in pending changelist 250. Changelist 250 contains files A#1, and B#1. This appends the files to the review because that is the default add mode for review-251.
   - Revision 4 of review 251 is created, containing files A#1, B#1, C#1, and D#1.

Review creation, and modification outside of Swarm

You can create a review and add a changelist to a review from outside of Swarm by adding keywords to the changelist description. The keyword is processed by Swarm when the changelist is shelved for review, when files in a pending changelist that is associated with a review are reshelved, and when a changelist is committed. For details on using a keyword to create a review and add a changelist to a review, see "Create a review" on page 548 and "Add a changelist to a review" on page 548.

Note
Swarm acts on the first valid keyword it finds in the changelist description, Swarm then ignores any further valid keywords it finds in the description.

Tip
When the content of a review is changed, Swarm checks to see which branches are in the new revision of the review:

- **If a new branch was added to the review:**
  - Default reviewers on the new branch are added to the review.
  - Moderators from the added branch become moderators for the review alongside the existing moderators.
  - **Only if workflow is enabled:** if the new branch is associated with a workflow, the workflow is merged with the existing workflow. The most restrictive workflow is used for the review.

- **If a branch is no longer part of the review:**
  - Reviewers for the review are not changed.
  - Moderators from the removed branch no longer moderate the review.
  - **Only if workflow is enabled:** if the branch was associated with a workflow, the branch workflow is removed from the review.

The following examples show how adding keywords to the changelist descriptions works in practice. In each case, you are collaborating with other users to produce a single review in Swarm:

Basic workflow using #review, and #append in the changelist description

1. You create a review with files A#1, B#1, and C#1 by using `#review` in the changelist description.
2. Review-xxxx is created and `#review` is automatically replaced with `#review-xxxx` in the changelist description.
3. User-2 appends their changelist that contains files C#2, D#1, and E#1 to review-xxxx by using `#append-xxxx` in the changelist description.
   - Review-xxxx now contains files A#1, B#1, C#2, D#1, and E#1.
   - The default add mode for review-xxxx is now append because User-2 added the changelist to the review using `#append-xxxx`.
4. You edit files A#1, and B#1 in your changelist so they become A#2, and B#2. The changelist description still contains `#review-xxxx`.
5. You shelve your pending changelist which has `#review-xxxx` in the changelist description, this appends the files to the review because that is the default add mode for review-xxxx.
   - Review-xxxx now contains files A#2, B#2, C#2, D#1, and E#1.

Advanced workflow using #review, and #append in the changelist description

1. You create a review with files A#1, and B#1 by using `#review` in the changelist description.
2. Review-xxxx is created and `#review` is automatically replaced with `#review-xxxx` in the changelist description.
3. User-2 appends their changelist that contains file C#1 to review-xxxx by using `#append-xxxx` in the changelist description.
   - Review-xxxx now contains files A#1, B#1, C#1.
   - The default add mode for review-xxxx is now append because User-2 added the changelist to the review using `#append-xxxx`.

4. User-3 adds their changelist that contains files A#2, and D#1 to review-xxxx by using `#review-xxxx` in the changelist description. This appends the files to the review because that is the default add mode for review-xxxx.
    - Review-xxxx now contains files A#2, B#1, C#1, and D#1.
5. You edit files A#2, and B#1 in your changelist so they become A#3, and B#2 and you add file E#1. The changelist description still contains `#review-xxxx`.
6. You shelve your pending changelist which has `#review-xxxx` in the changelist description, this appends the files to the review because that is the default add mode for review-xxxx.
    - Review-xxxx now contains files A#3, B#2, C#1, D#1, and E#1.

Advanced workflow using #review, #append, and #replace in the changelist description

1. You create a review with files A#1, and B#1 by using `#review` in the changelist description.
2. Review-xxxx is created and `#review` is automatically replaced with `#review-xxxx` in the changelist description.
3. User-2 appends their changelist that contains file C#1 to review-xxxx by using `#append-xxxx` in the changelist description.
    - Review-xxxx now contains files A#1, B#1, C#1.
    - The default add mode for review-xxxx is now append because User-2 added the changelist to the review using `#append-xxxx`.
4. User-3 replaces review-xxxx with their changelist that contains files A#2, and D#1 to review-xxxx by using `#replace-xxxx` in the changelist description.
    - Review-xxxx now contains files A#2, and D#1.
    - The default add mode for review-xxxx is now replace because User-3 added the changelist to the review using `#replace-xxxx`.
5. You edit files A#2, and B#1 in your changelist so they become A#3, and B#2 and you add file E#1. The changelist description still contains `#review-xxxx`.
6. You shelve your pending changelist which has `#review-xxxx` in the changelist description, this replaces the files in the review because that is the default add mode for review-xxxx.
    - Review-xxxx now contains files A#3, B#2, and E#1.

States

Reviews can be in one of several states. The biggest differentiator is whether the review's files have any outstanding, uncommitted changes or not.

Whenever a review state changes, an email notification is sent to all review participants, including:

- The review author
- Any user who comments on the review or its files
- Any user who has changed the review's state previously
- Any user who is @mentioned, or a member of a group that is @@mentioned in the review's description or comments.

The review state is indicated by the **Review state** button on the Review display page. The review state button is used to change the state of a review, see "Change review state" on page 390.

Code reviews can be in one of the following states:

- **Needs review:** The review has started and the changes need to be reviewed.
- **Needs revisions:** The changes have been reviewed and the reviewer has indicated that further revisions are required.
- **Approved:** The review has completed. The changes may need to be committed. If the changes have been committed then this review will be Approved and closed, otherwise it will be Approved and open. See the note below.
- **Rejected:** The review has completed. The changes are undesirable and should not be committed.
- **Archived:** The review has completed for now. However, it is neither rejected nor approved; it is simply put aside in case it is needed in the future.

> **Note**
> By default, when an **Approved** review is committed or updated, Swarm changes the state to **Needs Review** if the files have been modified since the review was approved. Files are considered modified if the list of involved files changes, or if the file content or file-type changes.
>
> If one or more files in a review has the filetype +k (**`ktext`**), this behavior is undesirable because the files will appear to be modified when the Helix server replaces RCS keywords with their current values. See "Unapprove modified reviews" on page 596 to see how to disable this behavior.

Self-approval by review authors

By default, review authors can approve their own reviews. This behavior is based on Swarm's advisory nature.

Self-approval by authors can be prohibited on a project-by-project basis by specifying moderators for project branches (see "State change restrictions with moderation" on the next page). However, authors who are moderators can self-approve their own reviews.

Administrators can configure Swarm to prevent all self-approval by review authors. See "Disable self-approval of reviews by authors" on page 552.

State change restrictions with moderation

Typically, any authenticated user can change the state of a review (remember that the review state is merely advisory in most cases).

When the **Only Moderators can approve or reject reviews** restriction is enabled for a project branch, that branch is *moderated*. See Add a project for details on adding moderators to project branches.

Changing the state of any review associated with a moderated branch is restricted as follows:

- Only moderators can approve or reject the review. Moderators can also transition a review to any other state.

  Important
  Moderators prevent the automatic approval of reviews, for more information about automatically approving reviews using workflow rules see "Workflow rules" on page 423.

  Note
  By default, when a review spans multiple branches that have different moderators, only one moderator from any one of the branches needs to approve the review.

  Swarm can be configured to require that one moderator from each branch must approve the review, this is a global setting. If a moderator belongs to more than one of the branches spanned by the review, their approval will count for each of the branches they belong to. For instructions on how to configure moderator behavior, see "Moderator behavior when a review spans multiple branches" on page 553.

- The review's author, when not a moderator, can change the review's state to **Needs Review**, **Needs Revision**, **Archived**, and can attach committed changelists.

  Normally, the review's author cannot change the review's state to **Approved** or **Rejected** on moderated branches. However, authors that are also moderators have moderator privileges, and may approve or reject their own review.

  When `disable_self_approve` is enabled, authors who are moderators (or even users with admin privileges) cannot approve their own reviews.

- Project members can change the review's state to **Needs Review** or **Needs Revision**, and can attach committed changelists. Project members cannot change the review's state to **Approved**, **Rejected**, or **Archived**.
- Users that are not project members, moderators, or the review's author cannot transition the review's state.
- For the review's author and project members, if a review is not in one of their permitted states, for example if the review's state is **Rejected**, they cannot transition the review to another state.

These restrictions have no effect on who can start a review.

Required reviewers

Reviews can optionally have required reviewers. When a review has required reviewers, the review cannot be approved until all required reviewers and required reviewer groups have up-voted the review. If the review is associated with a project that has assigned moderators, even the moderators cannot approve the review without up-votes from all required reviewers (but they can reject the review).

When a group is a required reviewer, it can be set to operate in one of two ways:

- **All votes required:** all members of the group must up-vote the review to allow the review to be approved.
- **One vote required:** at least one member of the group must up-vote the review to allow the review to be approved. If any member of the group down-votes the review, the review cannot be approved.

Required reviewers are expected to take greater care while performing a review than non-required reviewers, as their votes affect whether a review can be approved or not.

To edit the reviewers for a review, and to change whether a reviewer is required or not, see "Edit reviewers" on page 370.

> **Note**
> If a review involves a branch with assigned moderators, only a moderator can approve the review, even if all required reviewers have up-voted the review.
>
> See Add a project for details on adding moderators to project branches.

Change review state

The **Review state** dropdown button on the "Review display" on page 349 page is used to manually change the state of a Swarm review.

To change the state of the review:

1. Navigate to the review you want to update.
2. Click the **Review State** dropdown button.
3. Select the new state from the dropdown menu.
   The options available depend on the current state of the review and your user permissions:
   - **Needs Revision:** select to request changes to the files in the review.
   - **Needs Review:** select to request further review of the changes.
   - **Approve** (only available if the voting requirements for the review are satisfied. For information on voting requirements, see "Required reviewers" on page 380.): select to approve the review.

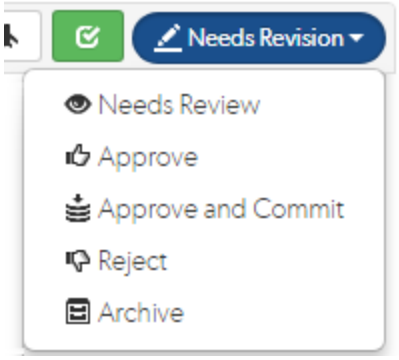
- **Commit** (only available for pre-commit reviews that have been approved): select to commit the review.
- **Approve and Commit** (only available for unapproved pre-commit reviews when the voting requirements for the review are satisfied. For information on voting requirements, see "Required reviewers" on page 380.): select to approve and commit the review in a single step, see "Approve and Commit" below.
- **Reject:** select to reject the review.
- **Archive:** select to archive the review.
- **Obliterate Review** (by default, only available for users with *admin* or *super* user rights): see "Obliterate Review" on page 533.

Approve and Commit

Approve and Commit is available in the dropdown menu for unapproved reviews, this enables you to approve and commit a review in a single step if required.

Note

- By default, any authorized user can commit a review. Swarm can be configured to restrict this behavior so that only the review author can commit the review, see "Disable commit" on page 565 for details.
- **Approve and Commit** is only available for unapproved pre-commit reviews when the voting requirements for the review are satisfied. For information on voting requirements, see "Required reviewers" on page 380.

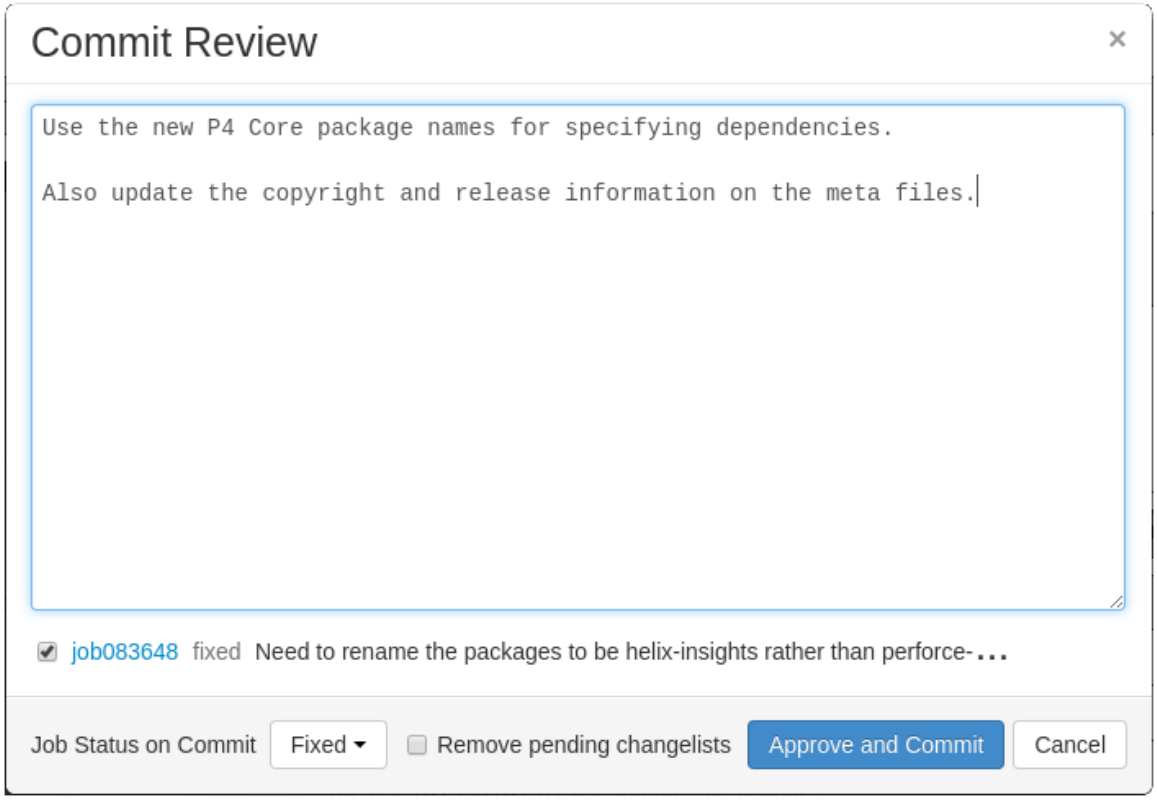


Note

If Swarm is configured to prevent approval of reviews with open tasks and a review has open tasks, the **Approve**, and **Approve and Commit** options will not be available for the review. This option is configured by an administrator, see "Prevent Approve for reviews with open tasks" on page 553.

To approve, or approve and commit a review with open tasks, you must address the tasks first and then set them to **Task Addressed**, or **Not a Task**, see "Set a task to Task Addressed or Not a Task" on page 270 for details.

1. Select **Approve and Commit** from the dropdown menu.
2. The **Commit Review** dialog is displayed.



3. Edit the review description if required.
4. Select which jobs should be associated with the review, and specify the job status on commit.
5. Select **Remove pending changelists**, Swarm will attempt to automatically clean up any changelists left behind after the review has been committed, including removing any shelved files. This option can be removed by an administrator, see "Review cleanup" on page 544 for details.
6. Click **Approve and Commit** to approve the review and commit the associated files.

> **Note**
> By default, Swarm records that you committed the review on behalf of the review's author. This can be configured by an administrator to only credit the committer and not the review author, see "Commit credit" on page 475 for details.

6 | Groups

Groups are a feature of Helix server that makes it easier to manage permissions for users. Swarm can use groups to coordinate review activities and responsibilities. This chapter covers how to manage groups in Swarm, including how to:

- "Add a group" below
- "Edit a group" on page 398
- "Delete a group" on page 398

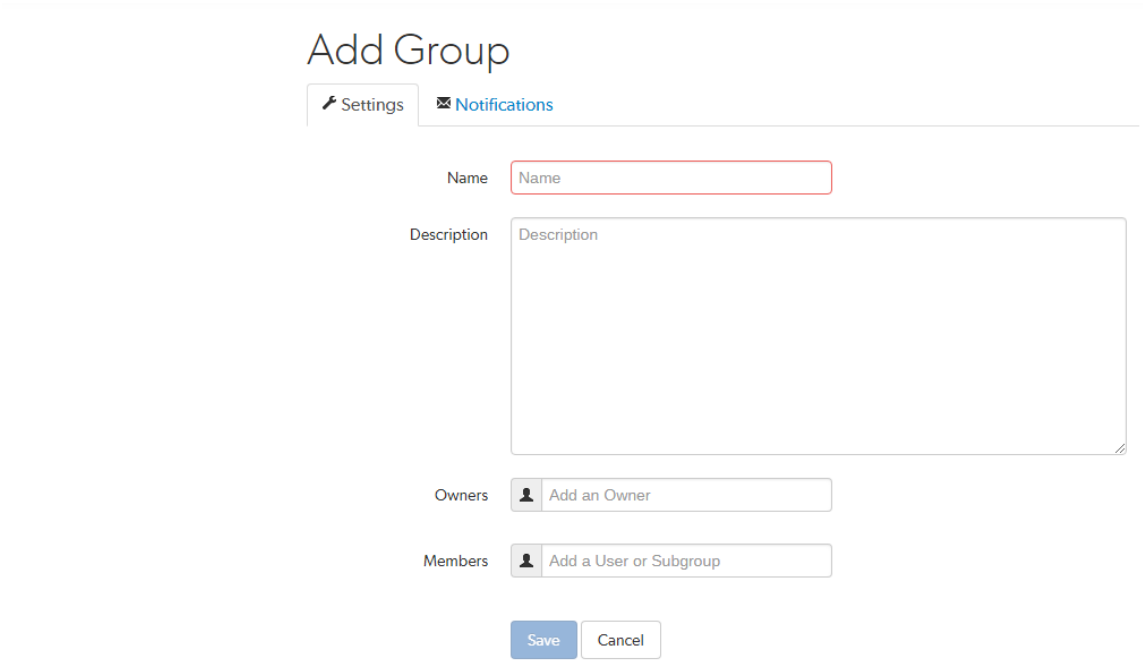
See "Groups" on page 291 for an introduction to Swarm groups.

Add a group

Important
You must have *super* privileges in Helix server (`p4d`), or have *admin* privileges in `p4d` version 2012.1 or later, to create a group. If you do not have sufficient permissions, Swarm does not display the **Add Group** button.

1. To view a list of groups, click **Groups** in the menu.
2. Click the **+ Add Group** button.

   The Add Group **Settings** tab is displayed:



3. Provide a name for the group.
4. **Optional:** provide a description.
5. **Optional:** specify an owner. This field auto-suggests users within Helix server as you type.

   Once specified, modifying the group's definition is restricted to group owners and users with *super* privileges in Helix server.

   If you do not specify an owner, you must specify at least one member (below).
6. **Optional:** specify group members. This field auto-suggests projects, groups, and users within Helix server as you type (up to a combined limit of 20 entries).

   If you specify a project, the project's members become members of the group. If you specify a group, that group becomes a sub-group of your new group, and all of its members (and members of any of its sub-groups) become members of your new group.

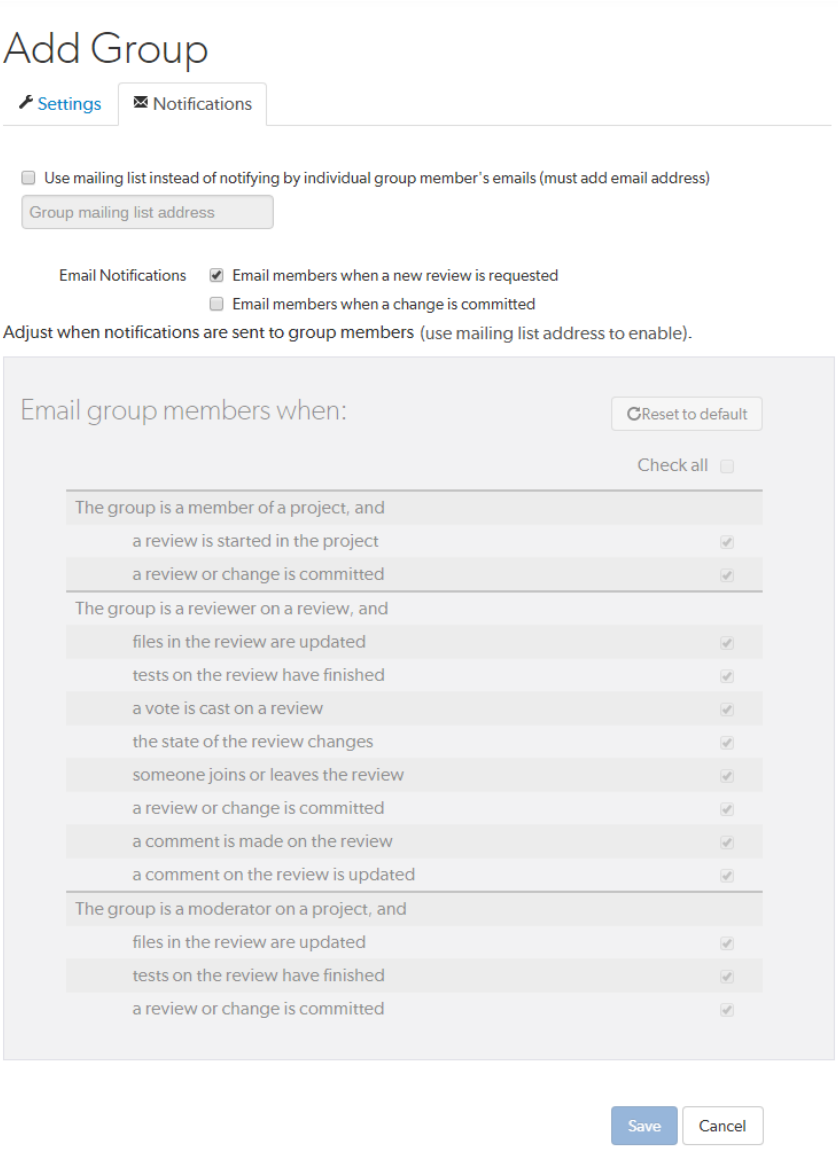
   If you do not specify any members, you must specify at least one owner (above).
7. You now have two options, either:
   - Click **Save** to finish adding the group. By default group members will be emailed when a new review is requested.

     > **Note**
     > The **Save** button is disabled if any required fields are empty.

or

- **Optional:** configure email notifications for the group in the Add Group, Notifications tab. See the next step for details.

8. **Optional:** click the **Notifications** tab to configure group email notifications.



9. **Optional:** add a group mailing list address by selecting **Use mailing list instead of notifying by individual group member's emails (must add email address)** and entering a valid group email address.

> **Note**
> - **Group mailing list enabled:** notifications are sent to the group email address.
> - **Group mailing list disabled:** notifications are sent to the group members individual email addresses.

The format of the email address is validated as you type.

10. Group members can be notified when a member of the group starts a review. Group members can be notified when a change is committed by, or on behalf of, a changelist owner who is also a member of this group. These settings are always available even if the group mailing list is not enabled.

    Select which actions send a notification to the group:

    - **Email members when a review is requested**: When any member of this group creates a review, this group will be notified.
    - **Email members when a change is committed**: When a change is committed into Perforce, if the owner of the changelist is a member of this group, this group will be notified.

      > **Tip**
      > When a user commits a changelist in Swarm, it is committed on behalf of the changelist owner. If the changelist owner is a member of this group, this group will be notified.

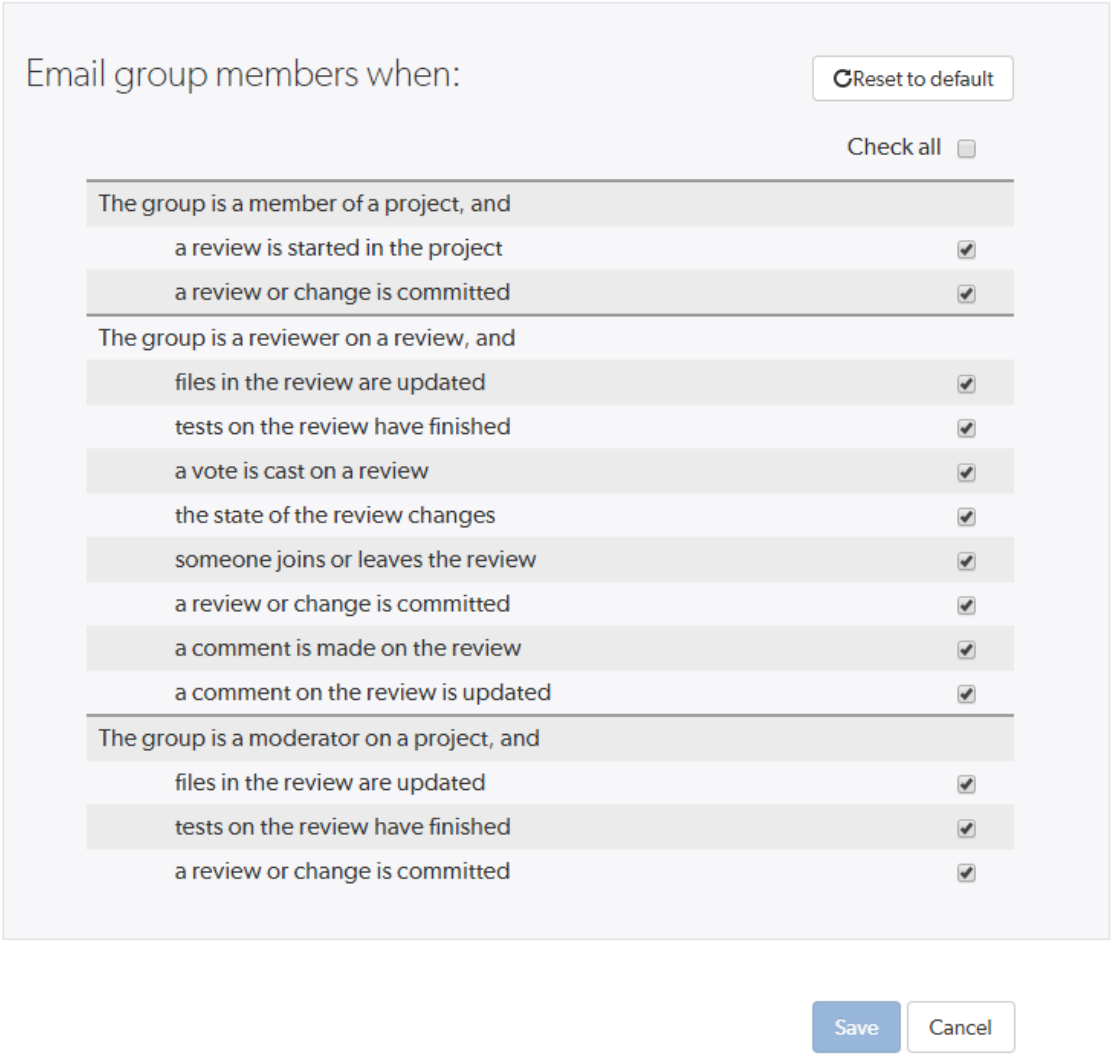
    > **Note**
    > Members of your group may receive notification emails even if group notifications are disabled as they may be members of a project, or follow a project or user, or Swarm's review daemon functionality may be enabled. See Notifications for details.

11. Group notification settings allow you to configure which notifications are sent to the group mailing list when events occur within Swarm (the group mailing list must be enabled). This allows you to limit the number of emails sent to the group mailing list. These settings apply across all projects.

    > **Note**
    > The following group notification settings are only available if the group mailing list address is configured.

Note
Defaults for these options are configured by the Swarm administrator, and they may force some of these options to on or off. See "Global settings" on page 529 for how this is configured.

Adjust when notifications are sent to group members.



Toggle notifications for each event on or off to control whether the group receives an email when that event occurs.

Clicking **Reset to default** resets the options back to system defaults.

12. Click **Save**.

> **Note**
> The **Save** button is disabled if any required fields are empty.

Edit a group

> **Important**
> You must be an owner of the group, or be a user with *super* privileges in Helix server, to edit the group.

1. Visit the group page you want to edit.
2. Click the **Settings** tab for the group to edit group details. If you do not have permission to edit a group, this tab does not appear.
3. Edit group details as required. See "Add a group" on page 393 for more details.
4. Click the **Notifications** tab for the group to edit group notifications. If you do not have permission to edit a group, this tab does not appear.
5. Edit group notifications as required. See "Add a group" on page 393 for more details.
6. Click **Save**.

> **Note**
> When a group is edited, the list of associated reviews is not immediately updated to match any changes in membership. The association with a review does not change until the review is updated.

Delete a group

> **Important**
> You must be an owner of the group, or be a user with *super* privileges in Helix server, to delete the group.

1. Visit the group page you want to delete.
2. Click the **Settings** tab for the group. If you do not have permission to edit a group, this tab does not appear.
3. Click **Delete**.

   A tooltip is displayed to confirm that you want to delete this group.
4. Click **Delete** to confirm.

7 | Projects

A Swarm project is a group of Helix server users who are working together on a specific codebase, defined by one or more branches of code, along with a job filter, and automated test integration. This chapter covers Swarm's project management capabilities, including how to:

- "Add a project" below
- "Edit a project" on page 411
- "Membership" on page 412
- "Delete a project" on page 419

See "Projects" on page 299 for an introduction to Swarm projects.

Add a project

1. On the Swarm **Projects** page, click the **+ Add Project** button.

> **Note**
> The ability to add projects can be limited to administrators only, or limited to members of specific groups. When limited, users who are not administrators, or a member of the specified group, will not see the **+ Add Project** button.

The **Add Project** page is displayed:

2. Provide a name for the project.
3. **Optional:** provide a description.
4. **Optional:** select the **Only Owners and Administrators can edit the project** checkbox. When checked, a field is displayed allowing you to add a new owner. The field auto-suggests users within Helix server as you type.

   Once specified, modifying the project's definition is restricted to project owners and administrators (users with *admin*-level or *super*-level privileges in Helix server).

   > **Note**
   > You cannot specify a group as an owner.

   > **Tip**
   > By default, if the **Only Owners and Administrators can edit the project** check box is selected for a project, only the project owners and administrators can view the project **Settings** page.
   >
   > This behavior can be changed by your Swarm administrator to allow project members that are not owners or administrators to view a read-only version of the project **Settings** page, see "Allow project members to view project settings" on page 539. The project **Automated Tests** and **Automated Deployment** details are hidden from project members unless they are an owner or an administrator. This enables project members to check the project settings but not change them.

5. Specify at least one team member. This field auto-suggests projects, groups, and users within Helix server as you type (up to a combined limit of 20 entries).

   > **Important**
   > During project creation, if you do not have *admin* privileges and you do not add yourself as a member or as an owner, you cannot edit this project's configuration later.

   See "Membership" on page 412 for more details.

6. **Default reviewers:**

a. **Optional:** specify **Default Reviewers** for the project.

This field auto-suggests users, and groups within Helix server as you type (up to a combined limit of 20 entries). Click on the user or group to add them as a default reviewer. Each time a new review is created, the default reviewers will be added to the review.

- **Users:** click the star icon to the left of the *userid* to toggle whether their vote is required or not. A solid star means that their vote is required to approve a review, whereas the outlined star means that their vote is optional.
- **Groups:** click the star icon to the left of the *groupid* , and select whether the group is a required reviewer (one vote), a required reviewer (all votes), or an optional reviewer. A solid star means that all group member votes are required to approve a review, a solid star with a 1 inside means at least one group member must vote up and no group members vote down to approve a review, and the outlined star means that the group vote is optional.

Click the **X** icon to the right of the *userid* or *groupid* to remove that default reviewer from the default reviewers list.

Important
When a review is part of multiple projects/project branches:

- The default reviewer lists for all of the projects and project branches the review is part of are combined and added to the review.
- If a default reviewer has different reviewer options set on projects and project branches that the review is part of, the strictest reviewer option is used for the review.

  Example: A review is created and it is part of **Project A**, **Project B**, and **Project Branch b**.

  Project A: default reviewer X is an Optional reviewer

  Project B: default reviewer X is an Optional reviewer

  Project Branch b: default reviewer X is a Required reviewer

  Result: default reviewer X is added to the review as a Required reviewer

Note
If users or groups are @mentioned in a new changelist description that includes `#review`, they will be added to the review as reviewers. If any of these reviewers are already specified as default reviewers they will not be added to the review again, the reviewer's most restrictive reviewer option is used for the review.

> **Note**
> If a default reviewer is deleted from Helix server they will not be added to new reviews.

b. **Optional:** click the **Retain default reviewers** checkbox to prevent default reviewers being removed from reviews associated with this project.

   For more information about retained default reviewers, see Retain default reviewers.

7. **Optional:** click the **Private** checkbox to make this project private. Private projects and their associated reviews are only visible to project owners, moderators, and members, plus users with *super* privileges in Helix server.

   For more information, see "Private projects" on page 306.

8. **Optional:** set the **Minimum up votes** required for reviews associated with this project.

   When **Minimum up votes** is set on a project, the project setting is used on the project branches unless **Minimum up votes** is set to **1** or higher on a branch. In this case the branch setting is used and the project setting is ignored.

   A review cannot be approved until all of the Required reviewers have voted up the review and the **Minimum up votes** specified has been satisfied.

   - If a review spans projects/branches, the Minimum up votes for **each** of the projects and branches must be satisfied before you can approve the review.
   - Required reviewers are included when up votes are counted.
   - When **Count votes up from** is set to **Members** for a workflow associated with a project/branch, only the up votes of members of the project contribute to satisfying the **Minimum up votes** for a project/branch. For more information about the **Count votes up from** rule, see "Workflow rules" on page 423.

     > **Important**
     > If the Workflow feature is disabled, all votes are counted not just votes from project members.

   > **Important**
   > If the **Minimum up votes** required is set higher than the number of reviewers that exist for a review, approval will be blocked for that review. This is true even if all the reviewers on the review have voted up the review.

9. **Optional** (not displayed if the Workflow feature is disabled): associate a **Workflow** with the project.

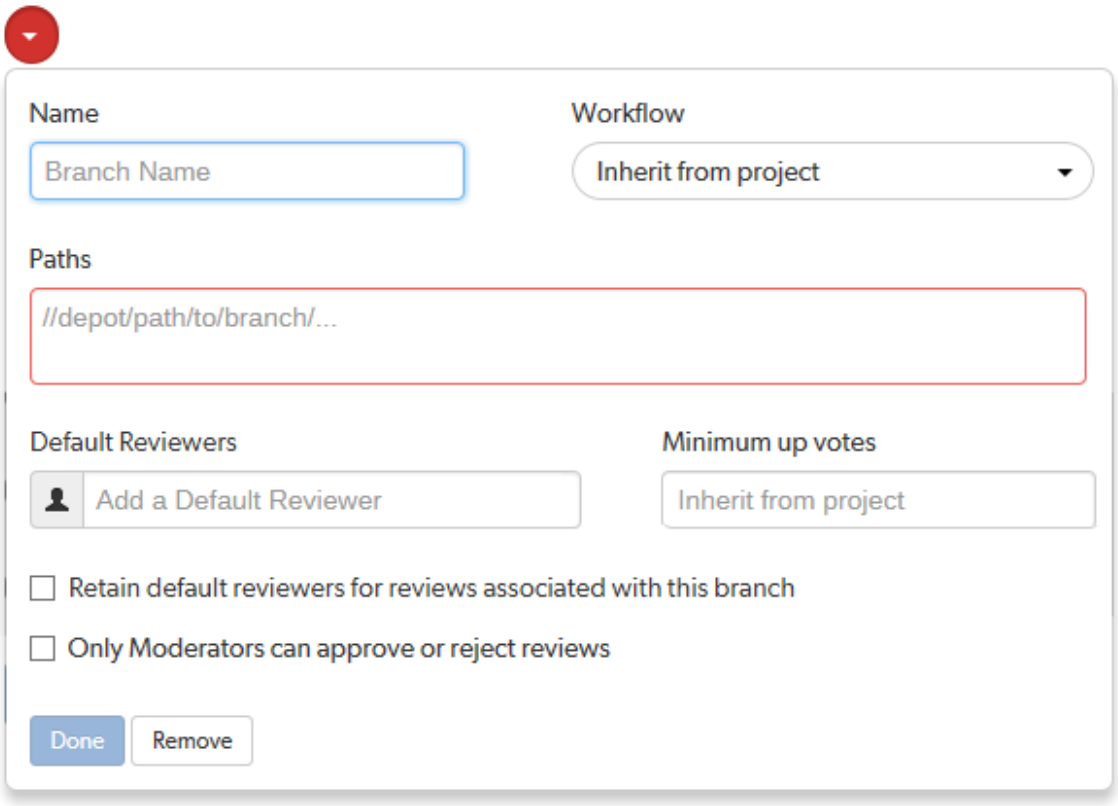
   To remove the existing workflow without replacing it with another workflow, select **No Workflow** from the **Workflow** dropdown list. This is the default when you create a new project.

To associate a workflow, select the workflow from the **Workflow** dropdown list, or enter the workflow name in the search field. The field auto-suggests workflows as you type. Only workflows that you own and shared workflows are shown in the dropdown list.

> **Tip**
> - When a workflow is associated with a project, the workflow is used for all of the branches in that project.
> - When a project branch is associated with a workflow, the workflow of the parent project is ignored and the branch workflow is used.

For more information about workflows and how project workflows interact with branch workflows, see "Workflow basics" on page 425.

10. **Optional:** click **+ Add Branch** to display the branch drop-down dialog.



a. Enter a short **Name** for your branch.

b. **Optional** (not displayed if the Workflow feature is disabled): associate a **Workflow** with the project branch.

To use the parent project workflow, select **Inherit from project** from the **Workflow** dropdown list. This is the default when you create a new branch. If the parent project is set to **No workflow**, the branch will use the global workflow rules.

To associate a workflow, select the workflow from the **Workflow** dropdown list, or enter the workflow name in the search field. The field auto-suggests workflows as you type. Only workflows that you own and shared workflows are shown in the dropdown list.

> **Tip**
> - When a workflow is associated with a project, the workflow is used for all of the branches in that project.
> - When a project branch is associated with a workflow, the workflow of the parent project is ignored and the branch workflow is used.

For more information about workflows and how project workflows interact with branch workflows, see "Workflow basics" on page 425.

c. Enter one or more branch paths in the **Paths** field using depot syntax, one path per line.

> **Tip**
> - Branch paths, and files can be excluded by putting a minus symbol **-** at the start of the path.
> - Branch paths are processed in order, starting with the first file path in the list.

For example:

```
//depot/main/swarm/...
-//depot/main/swarm/test/...
//depot/main/swarm/test/ResultSummary.html
```

The first path includes all of the directories and files under **`//depot/main/swarm/`** in the project branch.

The second path excludes all of the files in **`-//depot/main/swarm/test/`** from the project branch.

The third path includes the **`ResultSummary.html`** file from the previously excluded **`//depot/main/swarm/test/`** directory.

> **Note**
> - Wildcards should not be used; the only exception is that the branch path can end with the Helix server wildcard **`...`**
> - Branch paths are case sensitive.

- Branch paths, and files are not checked to see if they are valid when you save the branch:
  - If you enter an invalid path ending in the wildcard `...`, the path will not be displayed in the project file browser until the path is created. This allows you to specify a path before it has been created.
  - If you enter a path that ends with a file that has not been committed, or a non-existent file, Swarm displays a 404 error when you navigate the path to the file with the project file browser.

Note
The Project Commits tab can fail to show some Helix server commits in the top level view. Individual branch views display the commits correctly:

The **Project Commits** tab top level client view is made up of all of the branches of the project.

For example, Project Alpha:

Branch: QA:
`//depot/alpha/dev/QA/...`

Branch: Dev :
`//depot/alpha/dev/...`
`-//depot/alpha/dev/QA/...`

The project commits tab view is generated by processing the branches in the order that they were created in and from top to bottom for the paths in each of those branches.
For the project Alpha example above:
The first path includes all of the directories and files under `//depot/alpha/dev/QA/` in the project commits tab top level view.
The second path includes all of the directories and files under `//depot/alpha/dev/` in the project commits tab top level view.
The third path excludes all of the directories and files in `//depot/alpha/dev/QA/` from the project commits tab top level view.

Result: commits made to `//depot/alpha/dev/QA/` that should be shown for the **QA** branch are not displayed in the **Project Commits** tab top level view.

When you have a simple branch structure this can be avoided by considering this issue when you create your branches. In the example

above, creating the **Dev** branch first and then creating the **QA** branch avoids the problem because `//depot/alpha/dev/QA/` is not excluded from the **Project Commits** tab top level view.

d. **Optional:** specify **Default Reviewers** for the project branch.

This field auto-suggests users, and groups within Helix server as you type (up to a combined limit of 20 entries). Click on the user or group to add them as a default reviewer. Each time a new review is created, the default reviewers will be added to the review.

- **Users:** click the star icon to the left of the *userid* to toggle whether their vote is required or not. A solid star means that their vote is required to approve a review, whereas the outlined star means that their vote is optional.
- **Groups:** click the star icon to the left of the *groupid* , and select whether the group is a required reviewer (one vote), a required reviewer (all votes), or an optional reviewer. A solid star means that all group member votes are required to approve a review, a solid star with a 1 inside means at least one group member must vote up and no group members vote down to approve a review, and the outlined star means that the group vote is optional.

Click the **X** icon to the right of the *userid* or *groupid* to remove that default reviewer from the default reviewers list.

Important
When a review is part of multiple projects/project branches:

- The default reviewer lists for all of the projects and project branches the review is part of are combined and added to the review.
- If a default reviewer has different reviewer options set on projects and project branches that the review is part of, the strictest reviewer option is used for the review.

  Example: A review is created and it is part of **Project A**, **Project B**, and **Project Branch b**.

  Project A: default reviewer X is an Optional reviewer

  Project B: default reviewer X is an Optional reviewer

  Project Branch b: default reviewer X is a Required reviewer

  Result: default reviewer X is added to the review as a Required reviewer

> **Note**
> If users or groups are @mentioned in a new changelist description that includes **`#review`**, they will be added to the review as reviewers. If any of these reviewers are already specified as default reviewers they will not be added to the review again, the reviewer's most restrictive reviewer option is used for the review.

> **Note**
> If a default reviewer is deleted from Helix server they will not be added to new reviews.

e. **Optional:** click the **Retain default reviewers for reviews associated with this branch** checkbox to prevent default reviewers being removed from reviews associated with this branch.

   For more information about retained default reviewers, see Retain default reviewers.

f. **Optional:** set the **Minimum up votes** required for reviews associated with this branch.

   To inherit the parent project setting, leave **Minimum up votes** unset. This is the default when you create a new branch.

   To use the branch setting and ignore the parent project setting, set **Minimum up votes** to **1** or more on the branch.

   A review cannot be approved until all of the Required reviewers have voted up the review and the **Minimum up votes** specified has been satisfied.

   - If a review spans projects/branches, the Minimum up votes for **each** of the projects and branches must be satisfied before you can approve the review.
   - Required reviewers are included when up votes are counted.
   - When **Count votes up from** is set to **Members** for a workflow associated with a project/branch, only the up votes of members of the project contribute to satisfying the **Minimum up votes** for a project/branch. For more information about the **Count votes up from** rule, see "Workflow rules" on page 423.

     > **Important**
     > If the Workflow feature is disabled, all votes are counted not just votes from project members.

> **Important**
> If the **Minimum up votes** required is set higher than the number of reviewers that exist for a review, approval will be blocked for that review. This is true even if all the reviewers on the review have voted up the review.

g. **Optional:** check the **Only Moderators can approve or reject reviews** checkbox. When checked, a field is displayed allowing you to add a new moderator. The field auto-suggests groups and users within Helix server as you type.

   If a group is specified as a moderator, all of the members of that group have the same moderator privileges for that project branch as if they were added individually.

   Once the branch specification is complete and the project has been saved, changing the state of any review associated with this moderated branch is restricted as follows:

   - Only moderators can approve or reject the review. Moderators can also transition a review to any other state.

     Important
     Moderators prevent the automatic approval of reviews, for more information about automatically approving reviews using workflow rules see "Workflow rules" on page 423.

     Note
     By default, when a review spans multiple branches that have different moderators, only one moderator from any one of the branches needs to approve the review.

     Swarm can be configured to require that one moderator from each branch must approve the review, this is a global setting. If a moderator belongs to more than one of the branches spanned by the review, their approval will count for each of the branches they belong to. For instructions on how to configure moderator behavior, see "Moderator behavior when a review spans multiple branches" on page 553.

   - The review's author, when not a moderator, can change the review's state to **Needs Review**, **Needs Revision**, **Archived**, and can attach committed changelists.

     Normally, the review's author cannot change the review's state to **Approved** or **Rejected** on moderated branches. However, authors that are also moderators have moderator privileges, and may approve or reject their own review.

     When `disable_self_approve` is enabled, authors who are moderators (or even users with admin privileges) cannot approve their own reviews.

   - Project members can change the review's state to **Needs Review** or **Needs Revision**, and can attach committed changelists. Project members cannot change the review's state to **Approved**, **Rejected**, or **Archived**.

   - Users that are not project members, moderators, or the review's author cannot transition the review's state.

- For the review's author and project members, if a review is not in one of their permitted states, for example if the review's state is **Rejected**, they cannot transition the review to another state.

These restrictions have no effect on who can start a review.

h. Click the **Done** button to accept your branch specification.

Once the branch definition has completed, if any moderators were specified, the number of moderators for that branch is displayed in the list of branches:

> **Tip**
> Hover over the **X Moderators** text of a branch to view the moderators on that branch.

11. **Optional:** specify a job filter. The job filter allows you to specify criteria that are used to associate jobs with projects. For example, entering `Subsystem=ProjectA` associates jobs whose **subsystem** field is set to `ProjectA` with the current project.

> **Note**
> This job filter is simpler than the filters available in other Helix server clients. The filter must be expressed as `field=value` pairs; bare keywords are not permitted. The asterisk for wildcard matching is permitted, but no other filter expression syntax is permitted.

12. By default, project members and moderators are notified when a new review is started. Project members, moderators, and followers are notified when a change is committed.

    Select which actions send a notification:

    - **Email members and moderators when a new review is requested:** When a new review is requested for the project, all project members and moderators of the project are added to the email notification list.
    - **Email members, moderators and followers when a change is committed:** When a change is committed for the project, all project members, project moderators and project followers of this project are added to the email notification list.

Note
- When a group is a project member or project moderator, all of the members of that group are notified using the same logic as for individual project members and moderators.
- Any "@mention" on page 326 users and groups, or users and groups who are explicitly added to a review or changelist, will receive notifications even if new review/committed review notifications are disabled.

13. **Optional:** click the **Enable** checkbox beside **Automated Tests** to display the automated tests configuration fields.

    Specify a URL that triggers a test execution. Use the special arguments described in the dialog to help compose a URL that informs your test suite with important details. For more details, see "Integrate your test suite to inform review acceptance or rejection" on page 90
14. **Optional:** click the **Enable** checkbox beside **Automated Deployment** to display the automated deployment configuration fields.

    Specify a URL that triggers a deployment of the project's code. Use the special arguments described in the dialog to help compose a URL that informs your deployment program with important details. For more details, see "Automatically deploy code within a review" on page 97
15. Click **Save**.

Note
The **Save** button is disabled if any required fields are empty.

Important
It is possible to create a project that you cannot edit. This can happen if you have specified owners but not yourself as an owner, or if you have not specified yourself as a member. Swarm can detect some (but not all) such situations when you save a project; when it does detect such a situation, a warning dialog is displayed.

If you see this dialog, click **Continue** to save the project without your ownership/membership, or click **Cancel** within the dialog to continue editing the project. The project page's **Save** and **Cancel** buttons are disabled while this dialog is visible.

16. You can now use Swarm Workflow on reviews in this project.

Edit a project

1. Visit the project page you want to edit.
2. Click **Settings** in the project menu.
3. Edit the project's details as required. See "Add a project" on page 399 for more details.

> **Tip**
> If sharing is switched off for a workflow, any projects or branches that were associated with the workflow while it was shared will remain associated with it.
>
> If you edit a project or branch associated with that unshared workflow, you will still see the name of the workflow in the **Workflow** field, even if you don't own that workflow. If you remove that workflow from the project/branch you will not be able to see the workflow in the **Workflow** dropdown list unless you own it.

4. Click **Save**.

> **Note**
> By default, any member of a project can edit the project's configuration. Administrators can configure Swarm to prevent changes to the project's name and branch definition(s). See "Projects" on page 536 for details.

Membership

Membership in a Swarm project identifies users as belonging to the project, making them part of the team.

The are only a few notable differences between project members and non-members:

Difference	Member	Non-Member	Description
"Notifications" on page 316	✔	✖	Members receive project notifications; non-members do not.
"Avatars" on page 328	✔	✖	The project's home page features member's avatars.
"States" on page 387	✔	✖	Members can transition code review states; non-members cannot.

There are two ways to become a member of a project in Swarm:

1. Add a project and make yourself a member.
2. Ask a member of an existing project to add you as a member.

> **Note**
> If the project has any owners specified, you need to ask a project owner to add you as a member.

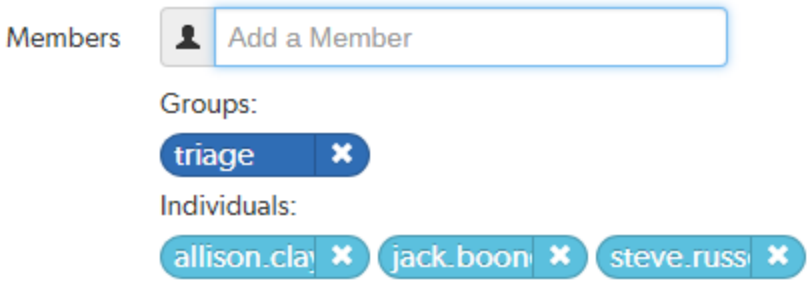
> **Note**
> Users with *super* privileges in Helix server can always adjust the settings for any project, including adjusting membership.

Add a member

If you are an owner of a project, or a member of a project without specified owners:

1. Visit the project page that needs the new member.
2. Click **Settings** in the project's toolbar.
3. The **Members** text field lets you specify a Swarm project, Helix server group, or Helix server user to add to the members for this project. The field auto-suggests *projectids*, *groupids*, and *userids* by matching what you have typed so far against the list of users in the Helix server.

   When you specify a project or group, all of the members of that project or group become members of this project. Swarm does not display all of the individual users, but it does provide a visual separation: project or group names are displayed first, with a darker blue background.

   When you hover your mouse over a member project or group, a tooltip appears displaying up to 100 of the *userids* of the project's or group's users.
4. Click **Save**.

Remove a member

If you are an owner of a project, or a member of a project without specified owners:

1. Visit the project page that has a member you want to remove.
2. Click **Settings** in the project's toolbar.

   Known members of the project are displayed beneath the **Members** text field, with a medium blue button representing projects or groups and a light blue button representing individual users.
3. Click the **X** next to the project id, group id, or userid you want to remove.
4. Click **Save**.

Warning
You are able to remove your own membership or ownership. Doing so could prevent you from managing the project.

Owners

A project *owner* is a Helix server user that controls the configuration for a project. An owner does not need to be a member of a project, but once the **Only Owners and Super Users can edit the project** check box has been selected, only an owner or user with *super* privileges in Helix server can edit any project settings.

Tip

- The **Owners** field is only displayed when the **Only Owners and Administrators can edit the project** check box is selected for the project.
- By default, if the **Only Owners and Administrators can edit the project** check box is selected for a project, only the project owners and administrators can view the project **Settings** page.

  This behavior can be changed by your Swarm administrator to allow project members that are not owners or administrators to view a read-only version of the project **Settings** page, see "Allow project members to view project settings" on page 539. The project **Automated Tests** and **Automated Deployment** details are hidden from project members unless they are an owner or an administrator. This enables project members to check the project settings but not change them.

Moderators

A project *moderator* is a user assigned to moderate reviews for a specific branch associated with a project. See how to specify moderators.

When **Only Moderators can approve or reject reviews** is set for a project branch, changing the state of any review associated with the moderated branch is restricted as follows:

- Only moderators can approve or reject the review. Moderators can also transition a review to any other state.

  Important
  Moderators prevent the automatic approval of reviews, for more information about automatically approving reviews using workflow rules see "Workflow rules" on page 423.

> **Note**
> By default, when a review spans multiple branches that have different moderators, only one moderator from any one of the branches needs to approve the review.
>
> Swarm can be configured to require that one moderator from each branch must approve the review, this is a global setting. If a moderator belongs to more than one of the branches spanned by the review, their approval will count for each of the branches they belong to. For instructions on how to configure moderator behavior, see "Moderator behavior when a review spans multiple branches" on page 553.

- The review's author, when not a moderator, can change the review's state to **Needs Review**, **Needs Revision**, **Archived**, and can attach committed changelists.

  Normally, the review's author cannot change the review's state to **Approved** or **Rejected** on moderated branches. However, authors that are also moderators have moderator privileges, and may approve or reject their own review.

  When `disable_self_approve` is enabled, authors who are moderators (or even users with admin privileges) cannot approve their own reviews.
- Project members can change the review's state to **Needs Review** or **Needs Revision**, and can attach committed changelists. Project members cannot change the review's state to **Approved**, **Rejected**, or **Archived**.
- Users that are not project members, moderators, or the review's author cannot transition the review's state.
- For the review's author and project members, if a review is not in one of their permitted states, for example if the review's state is **Rejected**, they cannot transition the review to another state.

These restrictions have no effect on who can start a review.

Default reviewers

User and group default reviewers can be set for individual projects and project branches. Each time a new review is created in the project or project branch, the default reviewers will be added to the review. See projects and project branches for adding default reviewers.

- A user can be set as a required reviewer or an optional reviewer.
- A group can be set as a required reviewer (one vote), a required reviewer (all votes), or an optional reviewer.

> **Important**
> **When a review is part of multiple projects/project branches:**

- The default reviewer lists for all of the projects and project branches the review is part of are combined and added to the review.
- If a default reviewer has different reviewer options set on projects and project branches that the review is part of, the strictest reviewer option is used for the review.

  Example: A review is created and it is part of **Project A**, **Project B**, and **Project Branch b**.

  Project A: default reviewer X is an Optional reviewer

  Project B: default reviewer X is an Optional reviewer

  Project Branch b: default reviewer X is a Required reviewer

  Result: default reviewer X is added to the review as a Required reviewer

Note
If users or groups are @mentioned in a new changelist description that includes **`#review`**, they will be added to the review as reviewers. If any of these reviewers are already specified as default reviewers they will not be added to the review again, the reviewer's most restrictive reviewer option is used for the review.

Note
If a default reviewer is deleted from Helix server they will not be added to new reviews.

Retain default reviewers

By default, default reviewers can be removed from an individual review by using the edit reviewers button on the review display page. Individual projects and branches can be configured to prevent default reviewers from being removed from individual reviews. For instructions on how to enable **Retain default reviewers** for a project or branch, see project or project branch.

Retain default reviewers basics

- Retained default reviewers can be added or removed by editing the project or project branch. Each time a review is updated in the project or branch, the list of default reviewers is checked.
  - Any new default reviewers are added to the review.
  - If a default reviewer has been removed from the project/branch, they will remain on the review. They are no longer a retained default reviewer and can be removed from the review if required.

- If a review is updated and it is no longer associated with the project/branch, the default reviewers for that project/branch will remain on the review. They are no longer retained default reviewers and can be removed from the review if required.

A review is checked when:

- The review state changes
- A new revision of the review is created
- A comment task state is changed
- A user votes on the review, or clears their vote
- Reviewers are edited on the review
- The review author is changed

- Retained default reviewers cannot be removed from a review by editing reviewers in the review.
- The retained default reviewer voting option cannot be reduced in a review by editing reviewers in the review.

  Example: if a user is a Required reviewer, the voting option cannot be reduced to a less strict option such as Optional reviewer.
- The retained default reviewer voting option can be made stricter for a review by editing reviewers in the review.

  Example: if a group is an Optional reviewer, the voting option can be increased to a stricter option such as Required reviewer (one vote).

Reviews spanning multiple projects and branches

Related projects and branches

When a review spans related projects and branches and a user/group is a retained default reviewer in one but a standard default reviewer in the other, the branch voting option is used for the user/group reviewer in the review.

Example:

- **Review 5678:** spans **Project A**, and **Branch A-1**.
- **Project A:**
  - **User-X:** Required reviewer
  - **Retain default reviewers:** Enabled.
    - **Branch A-1:** a child of **Project A**:
      - **User-X:** Optional reviewer
      - **Retain default reviewers:** Disabled.

- **Review:**
  - **User-X:** Optional reviewer, not a retained default reviewer.

    The project voting option is ignored and the branch voting option is used.

    There is no restriction on changing the default reviewer voting option.

    The default reviewer can be removed from the review.

Unrelated projects and branches

When an individual user/group default reviewer is retained for some of the projects and branches the review spans but not for others, the strictest voting option is used for a default reviewer on the review. The strictest retained default reviewer voting option is the minimum voting option for the user/group default reviewer in the review.

Example:

- **Review 1234:** spans **Project A**, **Project C**, and **Branch F-1**.
- **Project A:**
  - **Group-D:** Optional reviewer.
  - **Retain default reviewers:** Enabled.
- **Project C:**
  - **Group-D:** Required reviewer (all votes).
  - **Retain default reviewers:** Disabled.
- **Branch F-1:** not related to **Project A** or **Project C**:
  - **Group-D:** Required reviewer (one Vote)
  - **Retain default reviewer:** Enabled.
- **Review:**
  - **Group-D:** Required reviewer (All votes), retained default reviewer.

    The strictest voting option is used for the default reviewer.

    You can edit the voting option on the review. The minimum voting option is the strictest of the retained reviewer voting options. In this case that is, Required reviewer (one vote).

    The reviewer cannot be removed from the review.

Delete a project

Note

- Users with *super* or *admin* privileges in Helix server can delete projects.
- Project owners can delete projects that they own.
- If a project has no owners, any member of the project can delete the project.

When you delete a project:

- The deleted project is removed from the Swarm project list on the dashboard, and from project searches.
- The deleted project is removed from the profile page of the project owners, members, moderators, and followers.
- The project name is removed from the **Project** column on the "Reviews list" on page 342 page.
- Reviews that belong to the deleted project are not changed. The open and closed reviews remain accessible, their review states, comments, and tasks can be modified as normal.
- Reviews that belong to the deleted project, the project branch name link in the review heading is replaced with a link to the common depot location that contains the files included in the review.

Note

The deleted project name cannot be reused for a new project. This behavior is not case sensitive, this means that if you delete a project called **Project B** you cannot create a new project called **project b**.

Use the following steps to delete a project:

1. Visit the project page you want to remove.
2. Click **Settings** in the project menu.
3. Click **Delete**.

   A tooltip appears to confirm whether you want to delete this project.
4. Click **Delete** to confirm.

8 | Workflows

This chapter covers Swarm workflow management, including:

Related information:

- For instructions on how to list, search, and view workflows, see "Workflows" on page 307.
- For instructions on how to add a workflow to a project, see Add a workflow to a project.
- For instructions on how to add a workflow to a branch, see Add a workflow to a project branch.
- For a step-by-step walk-through of setting up a workflow, adding it to a project, and using it to progress through a review, see the Helix Swarm User Workflow Guide.
- **Administrators:** For instructions on how to configure the global workflow rules, see Workflow global rules.

Why should I use Swarm workflow?

By default Swarm's review process is basically advisory in nature with very few restrictions imposed on the review process. Team members can work in whatever way they want, but as projects grow in size and complexity, teams may prefer a stricter set of workflow rules over flexibility in order to improve productivity and quality. However, relying on people to stick to the agreed rules is not always enough.

The quality and speed of delivery of a product is heavily dependent on having a standardized and controlled workflow that follows best practice. It is also beneficial to automate the workflow where possible because the benefits of automating the workflow increases exponentially with scale. Teams within your organization will have varied workflow needs, it is important to maintain a balance between a company-wide policy and project specific needs. The Swarm workflow features have been designed to help you achieve this balance.

> **Tip**
> Additional Workflow rules and enhancements are on the road map for future Swarm releases.

The Swarm workflow feature enables you to enforce your rules and enables you to define specific rules for different projects and branches, providing you with flexibility and control over your review workflow. Workflows can be shared and they can be used on multiple projects/branches. Changing the workflow rules updates the workflow for all of the projects and branches it is associated with. This makes it easy to make changes in one place and change the workflow of multiple projects/branches. Specific workflow rules can be applied globally, ensuring basic company policies are followed by every project.

As a project owner you want to setup a project using the Mainline model with 'development', 'main' and 'release' branches. Each of these branches will have workflow rules that are tailored to its branch type.

Typically, you want your development branch to allow quick changes to the project content so that it can be populated quickly. Breaking test is acceptable at this point in the project. It is usually acceptable for a review to be raised after the content has been committed, this is the post-review commit model. Often you would only require one reviewer to vote up a review to trigger automatic approval. This minimal workflow configuration helps the development line to be built up quickly.

> **Tip**
> If your developers are pair programming, you might even allow them to commit changes without any review to further speed up the process. In the example shown below this is not the case and reviews are enforced by the workflow rules.

For your main branch you probably want to enforce pre-commit reviews, require more reviewers to vote up a review to trigger automatic approval, and restrict the up votes to project members only. This all helps to increase the stability of the main branch.

For a release branch you want even more protection for the content of the branch by only allowing critical changes to be committed to the branch. This is controlled by adding senior project members as moderators to the branch. Moderators are the only users that can approve or reject a review so they act as the gatekeepers for the branch. Moderators are added to the branch from the project branch settings dialog.

Development branch example:

A Development branch needs to be fast moving, allowing new features, collateral, and technologies to be added very quickly with issues fixed as they occur. In essence this is rapid development allowing a lot of work to be completed quickly but the branch is unstable. The workflow for your development branch would be configured to allow for this style of working, typically:

- The branch rules allow post-commit and pre-commit reviews as required:
  - Changes can be committed and a review can be requested later, this is a post-commit review. This approach is particularly useful if you are making large changes and you just want to get content into the depot and then check it.
  - Changes can be shelved with a review, this is a pre-commit review. This is useful when a user is adding standalone content or a feature that needs to be approved before it is committed to the depot.
- Only one reviewer needs to vote up a review to trigger automatic approval.

Main branch example:

A main branch needs to be more tightly controlled than the development branch as it is more stable as you move towards release. In essence, the main branch is used to fix any lingering issues and firm up the content and any tests you have in place. The workflow for your main branch would be configured to allow for this style of working, typically:

- Changelists must be shelved with a review, this is a pre-commit review.
- Up votes for reviews are only counted from project members.
- Two reviewers from the project are required to vote up a review to trigger automatic approval.
- The changelist can only be committed if the content of the changelist is identical to the content of the approved review.

Release branch example:

A Release branch typically needs to be very tightly controlled to protect the codeline with only a select set of users empowered to approve and commit reviews to the project before the product is released. In essence, the branch should be stable and you should not be making changes unless absolutely necessary. The workflow for your release branch would be configured to allow for this style of working, typically:

- Changelists must be shelved with a review, this is a pre-commit review.
- Senior members of the project are required to vote up the review before it can be approved.
- Moderators act as the final gatekeepers for review approval, only they have the power to approve a review once the required reviewers have voted the review up.
- The changelist can only be committed if the content of the changelist is identical to the content of the approved review.

These are three very different workflows, it is easy for team members to forget the rules especially when they are new to the project or team. Swarm workflows enable you to enforce your rules and control your branches more effectively. No more last minute changes breaking your release build and no more post-commit reviews in your pre-commit world.

Workflow overview

Note
The Swarm administrator can give individual groups and users special permission to ignore all of the Swarm workflow rules. Typically, these permissions are given to a small set of trusted groups and users, for example project owners and administrators. For more information about excluding users and groups from being bound by the workflow rules, see " Members or groups who can ignore ALL workflow rules" on page 610.

> **Tip**
> To understand the benefits of using the Swarm Workflow feature, see "Why should I use Swarm workflow?" on page 420

A workflow can be applied to a project or project branch to ensure that changelists and reviews in the project/branch follow the rules specified in that workflow. The "Workflow basics" on page 425 section gives an overview of global workflow rules, workflow rules and how they interact with each other.

Sections in this chapter:

Workflow rules

A workflow is made up of the following workflow rules:

On commit without a review:

This rule is applied when a changelist without an associated review is submitted from outside of Swarm.

Select one of the following options:

- **Allow:** the changelist is not checked, the changelist is submitted without a review.
- **Create a review:** the changelist is submitted and a review is created automatically for the changelist.
- **Reject:** the changelist submit is rejected.

> **Tip**
> The selected rule is also applied when a changelist is submitted with `#review` in the description. For more information about creating a review by including a keyword in the changelist description, see "Create a review" on page 548.

On commit with a review: This rule is applied when:

- A Swarm review is committed.
- A changelist with an associated review is submitted from outside of Swarm.

Select one of the following options:

- **Allow:** the changelist review state is not checked. The changelist is committed even if its associated review is not approved.

- **Reject unless approved:** the changelist is only submitted if its associated review is approved and the content of the changelist is identical to the content of the approved review.

> **Tip**
> The selected rule is also applied when a changelist is submitted with `#review-nnnnn`, `#replace-nnnnn`, or `#append-nnnnn` in the description (`nnnnn` = review ID). For more information about adding a changelist to a review by including a keyword in the changelist description, see "Add a changelist to a review" on page 548.

On update of a review in an end state:

Used to stop review content being automatically changed for reviews that are in specific states. By default, the protected end states are **Archived**, **Rejected**, and **Approve:Commit**. The end states are set by the Swarm administrator, see end_states.

> **Tip**
> When a review is in a protected end state, it can still be updated manually by a user from the Swarm UI.

This rule is applied when a changelist is added to a review.

Select one of the following options

- **Allow:** the review is not checked. The changelist is added to the review no matter what the review state is. This is the default setting.
- **Reject:** if the review is in one of the states specified in the `end_state` configurable:
  - The **Add Change** button is disabled for the review.
  - The changelist is rejected if it is added outside of Swarm using `#review-nnnnn`, `#replace-nnnnn`, or `#append-nnnnn` in the description (`nnnnn` = review ID).

Count votes up from:

By default, all of the up votes on a review are counted for the **Minimum up votes** value set on the project/branch the review is associated with. Limit the up votes that are counted to just the members of the project the review is associated with by using this rule.

This rule is applied when a user votes on a review.

Select one of the following options:

- **Anyone:** votes are counted for all reviewers on a review. This is the default setting.
- **Members:** only the up votes of members of the project the review is associated with are counted for the **Minimum up votes** set on projects/branches.

> **Tip**
> For instructions on how to set **Minimum up votes** for projects and branches, see Project minimum up votes and Branch minimum up votes.

Automatically approve reviews:

By default, reviews must be manually approved. Enable automatic approval of reviews with this rule.

This rule is applied when a user votes on a review, a required reviewer is added to a review, or a required reviewer is made an optional reviewer on a review.

Select one of the following options:

- **Never:** reviews are not automatically approved. This is the default setting.
- **Based on vote count:** reviews are automatically approved if:
  - There are no down votes on the review.
  - There are no moderators on the review. If a review has moderators it cannot be automatically approved.
  - All of the Required reviewers on the review have voted up.
  - The **Minimum up votes** on the review has been satisfied for **each** of the projects and branches the review spans.

> **Important**
> Moderators prevent the automatic approval of reviews. For more information about moderators, see "Moderators" on page 379.

> **Tip**
> After a review has been automatically approved it needs to be manually committed.

For information about creating a new workflow, see "Add a workflow" on page 433.

Workflow basics

- **Workflows:** can be associated with projects and/or project branches.
  - A workflow can be created by any Swarm user.
  - If a workflow is associated with any projects or project branches, that workflow cannot be deleted.
  - A workflow can be shared by the workflow owner.
  - A shared workflow can be viewed by any Swarm user.
  - A shared workflow can be applied to a project or project branch by any Swarm user that is authorized to edit the project.
  - If sharing is switched off for a workflow, any projects or branches associated with that workflow will remain associated with it.

- **Projects and branches:**
  - When a workflow is associated with a project, the workflow is used for all of the branches in that project.
  - When a project branch is associated with a workflow, the workflow of the parent project is ignored and the branch workflow is used. The project workflow is replaced by the branch workflow, the workflows are not merged. For examples of why you might use this, see "Restrictive project workflow with a less restrictive development branch workflow" on the next page, and "Project workflow with a more restrictive release branch workflow" on page 428.
  - When a branch is associated with a workflow and a global workflow rule is set to **Enforce on** by the administrator, the workflow of the parent project is ignored and the global workflow rule is merged with the branch workflow. The most restrictive rule is used.
- **Modifying a workflow:**
  - If a global workflow rule is set to **Enforce off** and that rule is modified, projects and branches that do not have an associated workflow will use the modified rule.
  - If a global workflow rule is set to **Enforce on** and the rule is modified, the entire Helix Core server will use the modified rule.
  - If a workflow is modified, any projects or branches associated with that workflow will use the modified workflow.
- **Global workflow rules:** configured by the Swarm administrator in Global workflow rules, each rule can be set in one of two modes:
  - **Enforce off:** the global workflow rule is used for projects and branches that do not have an associated workflow.
    - If a project or branch has an associated workflow, the global rule is ignored for that project or branch.
    - If projects are not used, the global workflow rule applies to the entire Helix Core server.
  - **Enforce on:** the global workflow rule is used for the entire Helix Core server. This allows a minimum workflow to be set for the Helix server.
    - The global workflow rule is merged with the workflow associated with a project or branch. The most restrictive rule is used.
- **Tests:**
  - If a test is associated with a workflow, it can be configured to run when a review associated with that workflow is either created/updated or submitted.
  - If a test is associated with the global workflow, it can be configured to run when a review is either created/updated or submitted.

Example workflows

This section contains examples of typical workflows that can be configured in Swarm.

Enforce a pre-commit review workflow

To enforce a pre-commit review workflow set the rules to:

- **On commit without a review:** set to **Reject**, ensures that changelist cannot be submitted without an approved review.
- **On commit with a review:** set to **Reject unless approved**, ensures that if a changelist already has an associated review, that review is approved and the content of the changelist is identical to the content of the approved review.

Allow a post-commit review workflow

To allow a post-commit review workflow set the rules to:

- **On commit without a review:** set to **Create a review**, ensures that all submitted changelists have a review.
- **On commit with a review:** set to **Allow**, enables changelists that already have an associated review to be submitted.

Restrictive project workflow with a less restrictive development branch workflow

The release and main branches need a restrictive workflow to protect the codelines. All submissions must have an associated approved review and the content of the changelist must be identical to the content of the approved review.

The development branch can be less restrictive because you want to get features in quickly and review them after they have been committed. In this scenario you would probably want to ensure that all submitted changelists have an associated review.

Project workflow:

- **On commit without a review:** set to **Reject**, ensures that changelist cannot be submitted without an approved review.
- **On commit with a review:** set to **Reject unless approved**, ensures that if a changelist already has an associated review, that review is approved and the content of the changelist is identical to the content of the approved review.
- **On update of a review in an end state:** set to **Reject**, ensures that the content of reviews that are considered complete cannot be changed. These end states are set by the Swarm administrator, see "Protected end states" on page 554.
- **Count votes up from:** set to **Members**, ensures that only the up votes of members of the project the review is associated with are counted for the **Minimum up votes** set on projects/branches.
- **Automatically approve reviews:** set to **Never**, ensures that reviews are not automatically approved.

Development branch workflow:

- **On commit without a review:** set to **Create a review**, ensures that all submitted changelists have a review.
- **On commit with a review:** set to **Allow**, enables changelists that already have an associated review to be submitted.
- **On update of a review in an end state:** set to **Reject**, ensures that the content of reviews that are considered complete cannot be changed. These end states are set by the Swarm administrator, see "Protected end states" on page 554.
- **Count votes up from:** set to **Anyone**, ensures that votes of all reviewers on a review are counted for the **Minimum up votes** set on projects/branches.
- **Automatically approve reviews:** set to **Based on vote count**, ensures that reviews are automatically approved when all of the review requirements have been satisfied.

Project workflow with a more restrictive release branch workflow

The project is set up with workflow that allows post commit reviews, this allows you to get features in quickly and review them after they have been committed. In this scenario it is a good idea make all submitted changelists have an associated review.

The release branch needs a more restrictive workflow to protect the codeline. All submissions must have an associated approved review and the content of the changelist must be identical to the content of the approved review.

Project workflow: defines the workflow for the project and its branches.

- **On commit without a review:** set to **Create a review**, ensures that all submitted changelists have a review.
- **On commit with a review:** set to **Allow**, enables changelists that already have an associated review to be submitted.
- **On update of a review in an end state:** set to **Allow**, allows the content of reviews that are considered complete to be changed.
- **Count votes up from:** set to **Anyone**, ensures that votes of all reviewers on a review are counted for the **Minimum up votes** set on projects/branches.
- **Automatically approve reviews:** set to **Based on vote count**, ensures that reviews are automatically approved when all of the review requirements have been satisfied.

Release branch workflow: defines the workflow for the release branch. The workflow of the parent project is ignored and the development branch workflow is used.

- **On commit without a review:** set to **Reject**, ensures that changelist cannot be submitted without an approved review.
- **On commit with a review:** set to **Reject unless approved**, ensures that if a changelist already has an associated review, that review is approved and the content of the changelist is identical to the content of the approved review.

- **On update of a review in an end state:** set to **Reject**, ensures that the content of reviews that are considered complete cannot be changed. These end states are set by the Swarm administrator, see "Protected end states" on page 554.
- **Count votes up from:** set to **Members**, ensures that only the up votes of members of the project the review is associated with are counted for the **Minimum up votes** set on projects/branches.
- **Automatically approve reviews:** set to **Never**, ensures that reviews are not automatically approved.

Merging multiple workflows

If a changelist spans multiple projects that have different workflows, the workflows are merged and the most restrictive workflow rules are used for the changelist. All of the tests associated with the merged workflows are run. If there are any duplicate tests they are only run once.

Example 1: Changelist 1234 spans Project A, Project B, and Project C

- **Project A:** //depot/jam/lib/...
- **Project B:** //depot/jam/docs/...
- **Project C:** //depot/common/src/...
- **Changelist 1234** contains the following files:
  - //depot/jam/lib/tests/checkLoad.sh
  - //depot/jam/docs/basics.workflow.html
  - //depot/common/src/index.html

The files span all three locations so the changelist is subject to the combined workflow rules for the three locations. The most restrictive workflow is used for the changelist:

	Changelist without a review	Changelist with a review	On update of a review in an end state	Count votes up from	Automatically approve reviews	Workflow Tests
Project A workflow	Allow	Reject unless approved	Allow	Anyone	Based on vote count	Smoke Test A
Project B workflow	Allow	Reject unless approved	Reject	Members	Never	Smoke Test B

	Changelist without a review	Changelist with a review	On update of a review in an end state	Count votes up from	Automatically approve reviews	Workflow Tests
Project C workflow	Create a review	Allow	Allow	Anyone	Based on vote count	Smoke Test B Smoke Test C
Changelist 1234 workflow	Create a review	Reject unless approved	Reject	Members	Never	Smoke Test A Smoke Test B Smoke Test C

Tip
The three workflows are combined and the most restrictive workflow rules are used for the changelist. All of the tests associated with the three workflows are run but **Smoke Test B** on **Project A** and **Project B** is only run once.

Example 2: Changelist 6789 spans Project M, Project N, and Project branch N-1

- **Project M:** //depot/thirdparty/lib/…
- **Project N:** //depot/thirdparty/lib/tests/…
- **Project branch N-1:**
  - //depot/projectN/…
  - //depot/thirdparty/lib/…
- **Changelist 6789** contains the following files:
  - //depot/thirdparty/lib/tests/checkReturns.sh
  - //depot/projectN/src/index.html
  - //depot/projectN/tests/pageLoadTest.php

The changelist files span all three locations so the changelist is subject to the combined workflow rules for the three locations. The most restrictive workflow is used for the changelist:

	Changelist without a review	Changelist with a review	On update of a review in an end state	Count votes up from	Automatically approve reviews	Workflow Tests
Project M workflow	Create a review	Reject unless approved	Allow	Anyone	Never	Smoke Test M
Project N workflow	Reject	Reject unless approved	Reject	Members	Never	Smoke Test N
Project branch N-1 workflow	Create a review	Allow	Allow	Anyone	Based on vote count	Smoke Test N-1
Changelist 6789 workflow	Create a review	Reject unless approved	Allow	Anyone	Never	Smoke Test M Smoke Test N-1

Tip
It is important to understand that the rules for **Project N** are ignored because **Project branch N-1** is a branch of **Project N** and it has its own workflow rules. This is why the result for **Changelist without a review** is **Create review**, the result for **On update of a review in an end state** is **Allow**, and the result for **Count votes up from** is **Anyone**.

Example 3: Changelist 3456 spans Project X, and Project Y

- **Project X:** //depot/main/product/…
- **Project Y:** //depot/main/product/docs/…
- **Changelist 3456** contains the following files:
  - //depot/main/product/tests/fileSelector.sh
  - //depot/main/product/docs/admin.html

The changelist files span both locations so the changelist is subject to the combined workflow rules for the two locations, and the Global workflow rules set by the administrator. The most restrictive workflow is used for the changelist:

	Changelist without a review	Changelist with a review	On update of a review in an end state	Count votes up from	Automatically approve reviews	Workflow Tests
Project X workflow	Create a review	Allow	Allow	Anyone	Based on vote count	Smoke Test X
Project Y workflow	Reject	Allow	Reject	Anyone	Based on vote count	Smoke Test Y
Global workflow rules	Create a review (**Enforce** on)	Reject unless approved (**Enforce** off)	Allow (**Enforce** on)	Members (**Enforce** on)	Never (**Enforce** off)	Global Smoke Test Global Full Test
Changelist 3456 workflow	Reject	Allow	Reject	Members	Based on vote count	Smoke Test X Smoke Test Y Global Smoke Test Global Full Test

Tip

- Global workflow rules:
  - When a global workflow rule is set to **Enforce** off, the rule is only used if the project/branch does not have an associated workflow. If the project/branch has an associated workflow, the global rule setting is ignored. This is why the result for the **Changelist with a review** rule is **Allow**, and the **Automatically approve reviews** is **Based on vote count**.
  - When a global workflow rule is set to **Enforce** on, the rule is merged with project/branch workflow and the most restrictive rule setting is used. This is why the result for the **Changelist without a review** rule is **Reject**, the result for the **On Update of a review in an end state** rule is **Reject**, and the result for **Count votes up from** is **Members**.

  The global workflow rule mode determines how the global rule interacts with the workflow rules. For more information about global rule modes, see "Workflow basics" on page 425.

- If one or more of the project branches has a moderator, the moderator prevents automatic approval of the review. For more information about moderators, see "Moderators" on page 379.

Add a workflow

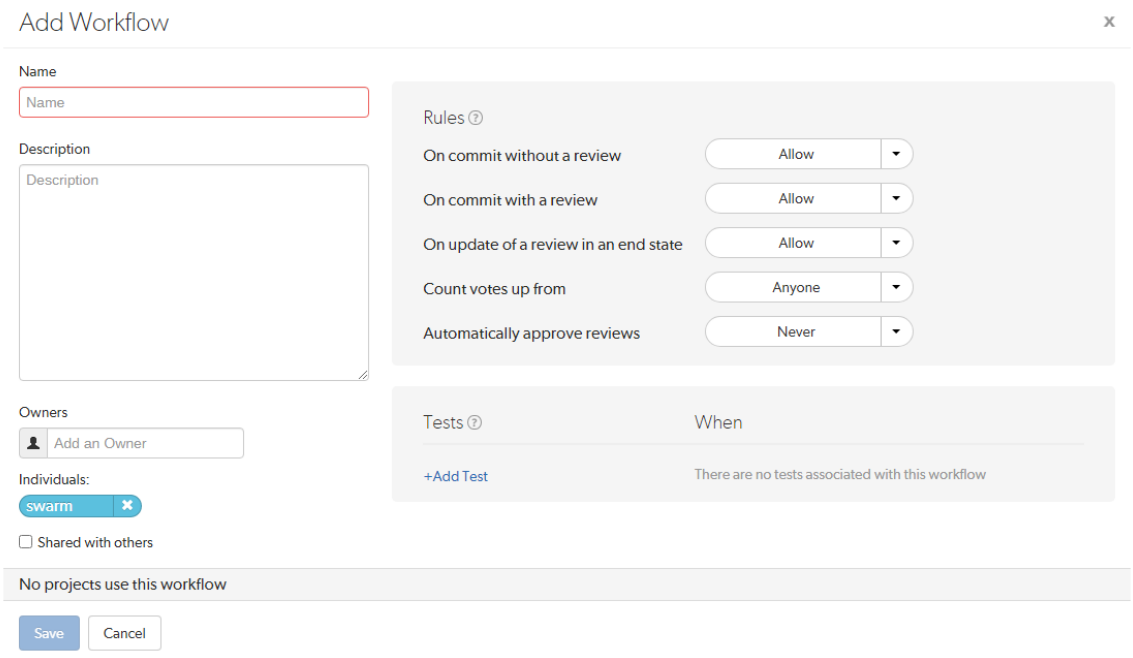
Note

- Swarm workflows can be created by any Swarm user.
- Shared Swarm workflows can be viewed by any Swarm user.
- Shared Swarm workflows can be applied to a project or project branch by any Swarm user that is authorized to edit the project.
- The global workflow can be viewed by any Swarm user but can only be edited by a Swarm user with *super* or *admin* privileges, or users that have been made an owner.

To add a workflow:

1. On the Swarm **Workflows** page, click the **+ Add Workflow** button.

   The **Add Workflow** page is displayed:



2. Enter a name for the workflow.
3. **Optional:** provide a description for the workflow.

4. You are automatically added as the owner of the workflow.

   Optional: add more owners if required. This field auto-suggests groups, and users within Helix server as you type (up to a combined limit of 20 entries).

   > **Important**
   > - A workflow must have at least one owner.
   > - If you remove yourself as an owner, you cannot edit this workflow configuration later unless you have *super* user rights.

5. **Optional:** if you want other Swarm users to be able to use this workflow, select the **Shared with others** checkbox.

   > **Tip**
   > Leave the checkbox unselected until you have proved that the workflow rules work as expected. This keeps the workflow private and stops other Swarm users using the workflow until you are happy with it. Once you are happy with the workflow, select the checkbox to share the workflow with other users.

6. **Rules:**

   On commit without a review:

   This rule is applied when a changelist without an associated review is submitted from outside of Swarm.

   Select one of the following options:

   - **Allow:** the changelist is not checked, the changelist is submitted without a review.
   - **Create a review:** the changelist is submitted and a review is created automatically for the changelist.
   - **Reject:** the changelist submit is rejected.

     > **Tip**
     > The selected rule is also applied when a changelist is submitted with `#review` in the description. For more information about creating a review by including a keyword in the changelist description, see "Create a review" on page 548.

   On commit with a review: This rule is applied when:

   - A Swarm review is committed.
   - A changelist with an associated review is submitted from outside of Swarm.

   Select one of the following options:

- **Allow:** the changelist review state is not checked. The changelist is committed even if its associated review is not approved.
- **Reject unless approved:** the changelist is only submitted if its associated review is approved and the content of the changelist is identical to the content of the approved review.

> **Tip**
> The selected rule is also applied when a changelist is submitted with `#review-nnnnn`, `#replace-nnnnn`, or `#append-nnnnn` in the description (`nnnnn` = review ID). For more information about adding a changelist to a review by including a keyword in the changelist description, see "Add a changelist to a review" on page 548.

On update of a review in an end state:

Used to stop review content being automatically changed for reviews that are in specific states. By default, the protected end states are **Archived**, **Rejected**, and **Approve:Commit**. The end states are set by the Swarm administrator, see end_states.

> **Tip**
> When a review is in a protected end state, it can still be updated manually by a user from the Swarm UI.

This rule is applied when a changelist is added to a review.

Select one of the following options

- **Allow:** the review is not checked. The changelist is added to the review no matter what the review state is. This is the default setting.
- **Reject:** if the review is in one of the states specified in the `end_state` configurable:
  - The **Add Change** button is disabled for the review.
  - The changelist is rejected if it is added outside of Swarm using `#review-nnnnn`, `#replace-nnnnn`, or `#append-nnnnn` in the description (`nnnnn` = review ID).

Count votes up from:

By default, all of the up votes on a review are counted for the **Minimum up votes** value set on the project/branch the review is associated with. Limit the up votes that are counted to just the members of the project the review is associated with by using this rule.

This rule is applied when a user votes on a review.

Select one of the following options:

- **Anyone:** votes are counted for all reviewers on a review. This is the default setting.
- **Members:** only the up votes of members of the project the review is associated with are counted for the **Minimum up votes** set on projects/branches.

> **Tip**
> For instructions on how to set **Minimum up votes** for projects and branches, see Project minimum up votes and Branch minimum up votes.

Automatically approve reviews:

By default, reviews must be manually approved. Enable automatic approval of reviews with this rule.

This rule is applied when a user votes on a review, a required reviewer is added to a review, or a required reviewer is made an optional reviewer on a review.

Select one of the following options:

- **Never:** reviews are not automatically approved. This is the default setting.
- **Based on vote count:** reviews are automatically approved if:
  - There are no down votes on the review.
  - There are no moderators on the review. If a review has moderators it cannot be automatically approved.
  - All of the Required reviewers on the review have voted up.
  - The **Minimum up votes** on the review has been satisfied for **each** of the projects and branches the review spans.

> **Important**
> Moderators prevent the automatic approval of reviews. For more information about moderators, see "Moderators" on page 379.

> **Tip**
> After a review has been automatically approved it needs to be manually committed.

7. **Tests**

   Optional: add tests to the workflow, the tests are run when a review associated with the workflow is either created/updated or submitted. Selected from the **When** dropdown for the test.

> **Tip**
> - Tests are only saved on the workflow when you click the **Save** button.
> - If a review is associated with multiple workflows, all of the tests on all of those workflows are run for the review. If the same test is on more than one of the associated workflows, it is only run once.

Add a test:

a. Click +Add Test in the Tests area.

b. Select the test to run from the **Tests** dropdown list.

c. Select when you want the test to run from the **When** dropdown list.

> **Tip**
> Review content change is when the files or content of files in a review change. A change to the review description does not trigger a test.

- **On Update** the test will run when:
  - a pre-commit review is created.
  - a review is submitted and the review content has changed compared to the previous review revision.
  - a review is updated and the review content has changed since the previous review revision.
  - a review is updated and the review content has not changed since the previous review revision, but this test failed or is still running on the previous revision. Any other tests for the review that passed for the previous review revision are not rerun and their test runs are copied to the new revision of the review.
- **On Submit** the test will run when a pre-commit review is submitted or a post-commit review is created.

d. Click the **Accept** ✔ button to add the test to the workflow.

Edit a test on the workflow:

a. Click the **Edit** button next to the test you want to edit.

b. The **Test** and **When** dropdown lists are displayed for the test.

c. Make changes as required.

d. Click the **Accept** ✔ button to confirm your changes.

Remove a test from the workflow:

Click the **Delete** button next to the test to remove it from the workflow.

The test is removed immediately, no confirmation is requested. You must save the workflow to complete the test removal.

8. Click **Save**.

> **Note**
> The **Save** button is disabled if any required fields are empty.

Edit a workflow

> **Important**
> Changes made to a workflow will change the workflow for projects and project branches that use that workflow.

> **Note**
> - A workflow can only be edited by the owner of the workflow or a user with *super* user rights.
> - To edit the **Global Workflow**, see "Workflow global rules" on page 605.

1. Visit the **Workflow** page you want to edit.
2. Edit the workflow details as required, see "Add a workflow" on page 433 for more details.
3. Click **Save**.

Delete a workflow

> **Note**
> - A workflow can only be deleted by the owner of the workflow, or a user with super user rights.
> - The **Global Workflow** cannot be deleted.

To delete a workflow:

1. Navigate to the workflow you want to delete.
2. Make sure that there are no projects, or branches associated with the workflow.

   If projects, or branches are associated with the workflow, the **Delete** button is muted, and the workflow cannot be deleted.

   For instructions on how to remove the workflow from a project, or branch, see Associate a workflow with the project and Associate a workflow with the project branch.

> **Note**
> Due to bug in Swarm 2019.3 and earlier, a workflow might falsely report that it is linked to a project when it is not. To remove the false link, navigate to the Settings page of the project reported as linked and click the **Save** button.

3. Click **Delete** to delete the workflow.

9 | Tests

Important
The **Tests** page is only available if Workflow is enabled, workflow is enabled by default.

This chapter covers Swarm test management, including:

- "Add a test" below
- "Edit a test" on page 445
- "Delete a test" on page 446
- To add a test to a workflow, see "Add a workflow" on page 433.

Related information:

- For instructions on how to list, search, and view tests, see "Tests" on page 312.
- **Administrators:** For instructions on how to add tests the global workflow, see "Global workflow" on page 605.

Add a test

Note
- Swarm tests can be created by any Swarm user.
- Shared Swarm tests can be viewed by any Swarm user.
- Shared Swarm tests can be added to a workflow by any Swarm user that is authorized to edit the workflow.
- Shared Swarm tests can be added to the global workflow by any Swarm user that is authorized to edit the global workflow.

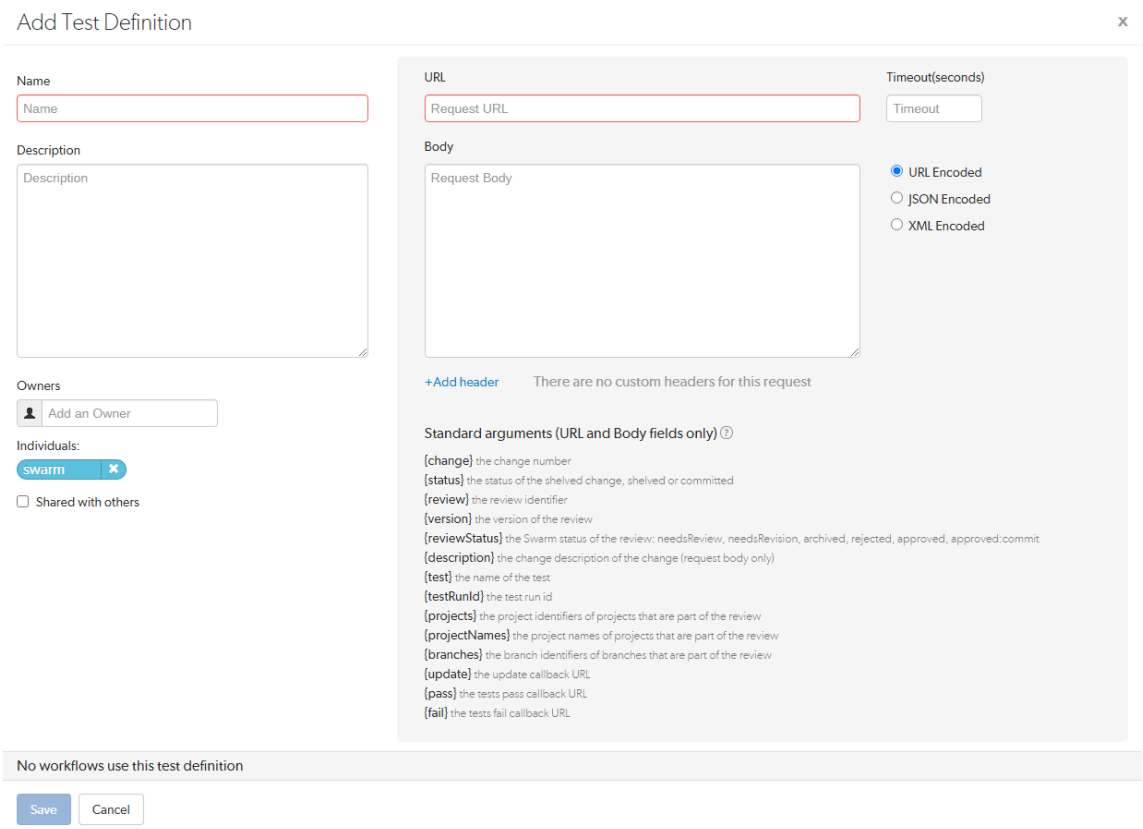
Note
In earlier versions of Swarm, global tests were set in the Swarm `config.php` file. From Swarm 2020.1, tests are added to the global workflow on the Swarm **Workflows** page so that they operate as global tests. If you upgrade from an earlier version of Swarm, your global tests are automatically migrated. Each global test is migrated as follows: the owner is set as the Swarm admin user (not shared), the name is set to the original test name, the description is set to the original test title, and it is added to the **Global Workflow**.

Associate a test with a workflow to ensure that the test is run when a review associated with that workflow is either created/updated or submitted. Associate a test with the global workflow to ensure that the test is run whenever a Swarm review is either created/updated or submitted. This ensures that the global tests are enforced for all changes even if they are not part of a project.

To add a test:

1. On the Swarm **Tests** page, click the **+ Add Test Definition** button.

   The **Add Test Definition** page is displayed:



2. Enter a name for the test. This should be a unique name that is 1 to 32 characters in length.
3. **Optional:** provide a description for the test.
4. You are automatically added as the owner of the workflow.

   Optional: add more owners if required. This field auto-suggests groups, and users within Helix server as you type (up to a combined limit of 20 entries).

> **Important**
>
> - A test must have at least one owner.
> - If you remove yourself as an owner, you cannot edit this test configuration later unless you have *super* user rights.

5. **Optional:** if you want other Swarm users to be able to use this test, select the **Shared with others** checkbox.

   > **Tip**
   > Leave the checkbox unselected until you have proved that the test works as expected. This keeps the test private and stops other Swarm users using the test until you are happy with it. Once you are happy with the test, select the checkbox to share the test with other users.

6. Enter the test request **URL** that will trigger your test suite in the URL text box.

   Special arguments are also available to pass details from Swarm to your test suite, see "Pass special arguments to your test suite" on the next page.

   > **Note**
   > **Helix Plugin for Jenkins 1.10.11 and later:** Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Body** and select **URL Encoded**.

7. **Optional:** set a **Timeout (in seconds)** as an integer. The timeout is used when Swarm connects to your test suite. If a timeout is not set, the `http_client_options` **`timeout`** setting is used. By default, this is 10 seconds.
8. **Optional:** specify parameters that your test suite requires in the request **Body**, select the encoding method using the radio buttons:
   - **URL Encoded**: POST parameters are parsed into name=value pairs. This is the default.
   - **JSON Encoded**: parameters are passed raw in the POST body.
   - **XML Encoded**: parameters are passed in XML format in the POST body.

   Special arguments are also available to pass details from Swarm to your test suite, see "Pass special arguments to your test suite" on the next page.
9. **Headers**

   Optional: enables you to specify name=value pairs to pass to the test suite.

   > **Tip**
   > Headers are only saved on the test when you click the **Save** button.

   Add a header name=value pair:

   a. Click +Add header in the headers area.

   b. Enter the name=value pair in the empty **Header** and **Value** boxes.

   c. The new header is saved when you click **Save** on the test page.

Edit a header name=value pair:

a. Edit the name=value pair in the **Header** and **Value** boxes.

b. The changes to the header are saved when you click **Save** on the test page.

Remove a header name=value pair from the test:

Click the **Delete** button next to the header to remove it from the test.

The header is removed immediately, no confirmation is requested. You must save the test to complete the header removal.

10. Click **Save**.

> **Note**
> The **Save** button is disabled if any required fields are empty.

Pass special arguments to your test suite

You can include special arguments in the request **URL** and **Body** to pass Swarm information about the change that triggered the test run to your test suite. Swarm automatically replaces the arguments with the relevant Swarm information when it calls the test:

> **Note**
> The curly braces `{}` are part of the arguments and must be used.

- `{change}` the change number
- `{status}` the status of the shelved change, shelved or committed
- `{review}` the review identifier
- `{version}` the version of the review
- `{reviewStatus}` the Swarm status of the review: **needsReview**, **needsRevision**, **archived**, **rejected**, **approved**, **approved:commit**
- `{description}` the change description of the change used to generate this update. **{description}** cannot be used in the **URL**, it can only be used in the request **Body**.
- `{test}` the name of the test
- `{testRunId}` the test run id
- `{projects}` the project identifiers of projects that are part of the review, null if the review is not part of a project. Comma separated if more than one project is involved in the review.
- `{projectNames}` the project names of projects that are part of the review, null if the review is not part of a project. Comma separated if more than one project is involved in the review.
- `{branches}` the branch identifiers of branches that are part of the review. Branch identifiers are prefixed by a project identifier (with a colon `:` separator by default) if the branch is part of a project,

null if the branch is not part of a project. Comma separated if more than one branch is involved in the review.

> **Tip**
> Some CI systems, such as Jenkins, escape the default `:` character resulting in a failed test request. If your CI system needs a different separator character, your Swarm *admin* can configure Swarm to use that character.
>
> Be careful when selecting your separator character, some characters can cause issues when calling tests. For example, a `#` character in a URL call means the branch is treated as an anchor and a `%` character in a JSON body encoding is escaped. Both of these will result in a failed test request.
>
> For instructions on configuring the separator character, see "project_and_branch_separator" on page 589.

- **`{branchName}`** the branch name(s) impacted by the review, comma-separated
- **`{update}`** the update callback URL. You can include any or all of the following when calling the update url to update the test run: status, messages, and a url in the body that links to the CI system for that run. They should be formatted in JSON in the body of the POST request. **Status:** valid status values are **`running`**, **`pass`**, and **`fail`**. **Messages:** you can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated. **`{update}`** is the preferred option for Swarm 2019.3 and later.

> **Important**
> If your test system cannot POST to Swarm, you cannot use **`update`** and you must use **`pass`** and **`fail`** instead.

> **Note**
> **When using {update}:**
>
> - If your build script has access to any messages related to the test execution, pass the messages to Swarm using the **`{update}`** URL. Swarm uses the provided message(s) to add to the test results.
> - If your build script has access to the results of test execution, include a POST parameter called **`url`** when calling the update URL. Swarm uses the provided url to link reviews to the test results. Valid test status values are **`running`**, **`pass`**, and **`fail`**.
> - The **`{update}`** callback url accepts a JSON body where you can specify any or all of the following: messages, url, and status.
>
> ```
> {
>     "messages" : ["My First Message", "My Second Message"],
>     "url" : "http://jenkins_host:8080/link_to_run",
>     "status": "pass"
> }
> ```

- **`{pass}`** the tests pass callback URL. From Swarm 2019.3 and later, **`{update}`** is preferred. For more details, see the note below.
- **`{fail}`** the tests fail callback URL. From Swarm 2019.3, **`{update}`** is preferred. For more details, see the note below.

> **Note**
> - **`{update}`**, **`{pass}`**, and **`{fail}`** are composed automatically by Swarm, they include Swarm's own per-review authentication tokens.
> - **When using `{pass}` and `{fail}`:** if your build script has access to the results of test execution, include a GET or POST parameter called **`url`** when calling the pass or fail URLs. Swarm uses the provided url to link reviews to the test results.

> **Tip**
> Swarm 2019.3 and later still supports **`{pass}`** and **`{fail}`**, however **`{update}`** is preferred because you can also include a message with the test status.

Edit a test

> **Important**
> Changes made to a test will change the test for workflows that use it.

> **Note**
> A test can only be edited by the owner of the test or a user with *super* user rights.

1. Navigate to the **Test** page you want to edit.
2. Edit the test details as required, see "Add a test" on page 440 for more details.
3. Click **Save**.

Delete a test

> **Note**
> A test can only be deleted by the owner of the test or a user with super user rights.

To delete a test:

1. Navigate to the test you want to delete.
2. Make sure that there are no workflows associated with the test.

   If workflows are associated with the test, the **Delete** button is disabled, and the test cannot be deleted.

   For instructions on how to remove the test from a workflow, see "Add a workflow" on page 433.
3. Click **Delete** to delete the test.

10 | Integrations

Swarm integrates with a variety of other applications and processes to provide important functionality, such as automated testing and deployment of code in reviews, issue tracking, file previewing, and downloading archives. This chapter describes each of the integrations available with Swarm.

Included integrations:

- JIRA
- LibreOffice
- Zip archive
- Automated deployment for reviews
- Automated testing for reviews
- P4V

JIRA

Swarm's JIRA integration allows code reviews and committed changes to be associated with JIRA issues, making it easy to reference associated issues, and see the state of a code review or committed change within JIRA.

To associate a code review with a JIRA issue, include a JIRA issue identifier in the review's description, for example `SW-1234`. Swarm links to the JIRA issue and creates a link within the JIRA issue back to the code review in Swarm. Multiple JIRA issues can be included in the changelist description.

As a code review progresses, Swarm updates each associated JIRA issue with the review's current status.

> **Note**
> Swarm fetches JIRA project identifiers using a worker, once per the worker process' lifetime. See "Worker configuration" on page 603. New JIRA projects will not auto-link to JIRA until the project identifiers have been updated. By default, project identifiers are updated once every ten minutes.

Swarm can also be configured to add a link to a Perforce job within a JIRA issue. Perforce job links are created when a new JIRA or job is created or updated. This gives you direct access to the Perforce job from a link in the **Issue Links** section of the JIRA issue.

By default, JIRA module and Perforce job links are disabled.

See "Enabling the JIRA module" on the next page and "Enabling Perforce job links in JIRA issues" on the next page for details on configuring these features.

For a summary of JIRA integration and Perforce job link workflow, see "JIRA integration and Perforce job link workflow" on page 450.

Prerequisites for JIRA integration and job links in JIRA

- The Perforce Defect Tracking Gateway (P4DTG) must be configured for Swarm, JIRA, and P4D, see Installing and configuring the Helix Defect Tracking Gateway in the *Helix Defect Tracking Gateway Guide*.
- The Swarm workers must be working, see "Worker status" on page 603.
- The JIRA module must be enabled, see "Enabling the JIRA module" below.
- To add Perforce job links to JIRA issues, job links must be enabled and the JIRA **`job_field`** must be set, see "Enabling Perforce job links in JIRA issues" below.

Enabling the JIRA module

When enabled, the JIRA module links JIRA issues referenced in change descriptions, job descriptions, comments, and reviews to your local JIRA service. By default, the JIRA module is not enabled.

To enable the JIRA module, add the following configuration block to your *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'jira' => array(
        'host'      => '', // URL for your installed JIRA web interface
        'user'      => '', // the username required for JIRA API access
        'password'  => '', // the password required for JIRA API access
        'job_field' => '', // optional, if P4DTG is replicating JIRA issue
IDs
                           // to a job field, list that field here
    ),
```

Note
If your JIRA web interface uses HTTPS, you may need to configure "HTTP client options" on page 568 so that Swarm can connect successfully.

Enabling Perforce job links in JIRA issues

When a Perforce job is created, a link to that job is added to the associated JIRA issue. When P4DTG replication detects a change to an issue, such as description, status, etc. a job link is added to the associated JIRA issue if it does not already exist.

Important
If Swarm fails to add a Perforce job link to a JIRA issue because that JIRA issue does not exist, the failure is logged by Swarm. Swarm will not try to add the job link to that JIRA issue again.

Note
If a Perforce job is updated and the **`DTG_DTISSUE`** field value is changed, a job link will be added to the new JIRA issue. The Perforce job link will not be removed from the original JIRA issue.

Prerequisites for Perforce job links in JIRA issues:

- The JIRA module must be enabled
- The JIRA **`job_field`** must be set
- P4DTG must be configured

To enable Perforce job links in JIRA issues, add the following configuration block to your *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'jira' => array(
        'host'      => '', // URL for your installed JIRA web interface
        'user'      => '', // the username required for JIRA API access
        'password'  => '', // the password required for JIRA API access
        'job_field' => '', // optional, if P4DTG is replicating JIRA issue
IDs
                           // to a job field, list that field here
        'link_to_jobs'    => false, // set to true to enable Perforce job
links in JIRA, P4DTG and job_field required
        'delay_job_links' => 60, // delay in seconds, defaults to 60
seconds
        'relationship'    => '', // subsection name for Perforce job links
                                 // defaults to empty, Perforce job links
added to the "links to" subsection
    ),
```

- **`link_to_jobs`**:
  - **`false`**: disable Perforce job links in JIRA. The default setting is **`false`**.
  - **`true`**: enable Perforce job links in JIRA.

- `delay_job_links`: delay set in seconds. Sets the delay between a new JIRA or job being created or updated, and Swarm adding the job link to the JIRA issue. This configurable is used to avoid a race condition between JIRA and the Helix server. By default `delay_job_links` is set to 60 seconds, if a race condition is experienced increase the delay time.
- `relationship`:
  - `''` empty: Perforce job links are added to the **links to** subsection of **Issue Links** section of JIRA issues. This is the default value.
  - `<JobLinksSubsectionName>`: creates a new subsection in the **Issue Links** section of JIRA issues. Perforce job links are added to this subsection.

> **Note**
> Existing Perforce job links are not moved to the subsection defined by the `relationship` configurable. However, if P4DTG detects a change to an issue, the job link is added to the new subsection. The job link is not deleted from its original location.

JIRA integration and Perforce job link workflow

This section describes the workflow for JIRA integration and job links in JIRA issues when you request a new Swarm review from P4V.

> **Tip**
> The process for adding a Perforce job and creating a review differs for the other clients but the Swarm worker and Perforce Defect Tracking Gateway (P4DTG) workflow is the same.

1. Create a new JIRA issue in your JIRA system.
2. P4DTG detects the new JIRA issue and creates a new Perforce job for that JIRA issue.
3. Add the new Perforce job to a pending changelist in P4V, see Add a job to a pending changelist in the *P4V User Guide*.
4. Request a new Swarm review for the pending changelist in P4V, see Request a review in the *P4V User Guide*.
5. P4V passes the review request to the Swarm worker queue.
6. The review request is processed by the next available Swarm worker, the new review is created.
7. Swarm checks its JIRA configuration. If the `job_field` exists in the Perforce job, Swarm adds a link to the new Perforce job in the **Issue Links** section of the JIRA issue.

To check that these steps have been completed and to view the links:

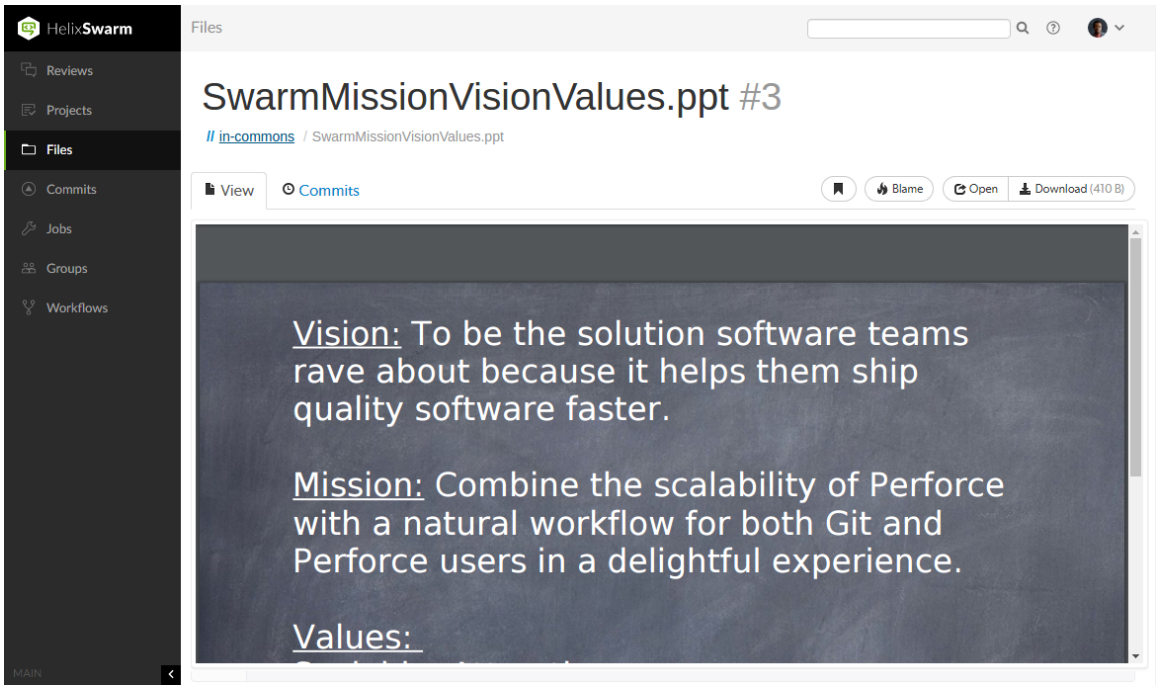
1. Open the new Swarm review, the associated Perforce jobs are displayed on the review page below the review description.
2. Click the Perforce job link to open the job.

3. The Swarm review link is displayed on the job page below the job description.
4. Click the **Details** tab on the Perforce job page to view the **DTG_DTISSUE** field.
5. Click the JIRA link in the **DTG_DTISSUE** field to view the JIRA issue.
6. The Swarm review and Perforce job links are displayed in the **Issue Links** section of the JIRA issue page.

LibreOffice

LibreOffice is a free power-packed open source personal productivity suite. When LibreOffice is installed on the server hosting Swarm, Swarm automatically detects its presence and uses LibreOffice to prepare PDF previews of a variety of file types, including:

- Word documents (`.doc`, `.docx`)
- PowerPoint presentations (`.ppt`, `.pptx`)
- Excel spreadsheets (`.xls`, `.xlsx`)
- Visio diagrams (`.vsd`)
- Rich-text files (`.rtf`).



Depending on your server's platform and distribution, LibreOffice may be provided as multiple packages and not all packages may be installed by default. If certain filetypes do not preview as expected in Swarm, you may need to install these optional packages to include all of the file handling capabilities of LibreOffice.

For more information on LibreOffice, see https://www.libreoffice.org.

Limitations

The LibreOffice integration has several limitations:

- Document previews in Swarm may appear different than on a desktop system if the document's fonts are not installed on the server hosting Swarm and LibreOffice. In addition, LibreOffice has some limitations rendering Microsoft Office file types, so LibreOffice-generated previews may differ from what you see using Microsoft Office.
- Large files may require a notable amount of time to preview. Very large files may exhaust the resources of the server hosting Swarm, causing the preview to fail and temporarily impacting Swarm performance.
- Swarm is currently not able to detect or show differences in LibreOffice-supported file types.
- LibreOffice cannot currently provide previews on a Mac OSX server hosting Swarm.

Installation

We recommend that you install LibreOffice from your OS distribution, via `apt-get`, `yum`, etc.

The minimal packages, and their transitive dependencies required for Swarm are:

- `libreoffice-calc`
- `libreoffice-draw`
- `libreoffice-impress`
- `libreoffice-writer`
- `libreoffice-headless` (CentOS/RHEL only)

Zip archive

When the `zip` command-line tool is installed on the Swarm server, Swarm allows users to download ZIP archives of files and folders. You can download a Zip archive using the **Download .zip** button from the following Swarm pages:

- "Review display" on page 349 page.
- "Changelists" on page 259 page.
- "Files" on page 243 page.

Installation

Install the zip command-line tool from your operating system distribution using `apt-get`, `yum`, etc.

Configuration

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Configure the archiving feature with the following configuration block in the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'archives' => array(
      'max_input_size'    => 512 * 1024 * 1024, // 512M (in bytes)
      'archive_timeout'   => 1800,              // 30 minutes
      'compression_level' => 1,                 // 0-9
      'cache_lifetime'    => 60 * 60 * 24,      // 1 day
    ),
```

The `max_input_size` key specifies the maximum file/folder content size that can be processed into a ZIP archive. The default value permits up to 512 megabytes of content to be compressed. Smaller values limit the amount of file/folder content but provide faster downloads; larger values can allow increased scanning, syncing, compressing, and downloading times.

The `archive_timeout` key specifies the amount of time, in seconds, to allow Swarm to prepare the ZIP archive for downloading. Shorter times can limit the practical size of a ZIP archive, depending on the performance of your network and the filesystem hosting Swarm; even with a generous `max_input_size` setting, if `archive_timeout` seconds have elapsed, the archive operation is terminated.

The `compression_level` key specifies the compression level to use, and must be within the range `0` to `9`. `0` means no compression, `9` means maximum compression. As this value is increased, smaller ZIP archives may result, but may require greater compression time. Swarm uses the default of `1`, which provides a reasonable tradeoff of fast compression times with light compression that can still result in an archive notably smaller than the original file/folder content.

The `cache_lifetime` key specifies the desired maximum age of cached ZIP archives. Increasing the value increases the amount of time that ZIP archives exist in the cache, which can improve the user experience for frequently downloaded files. However, ZIP archives can be quite large (depending on the size of your depot within the Helix server) and can require significant disk storage. Decreasing the value can mitigate the amount of disk space required for the cache; the tradeoff is that frequently accessed ZIP archives may need to be generated more frequently, which can have an impact on CPU and disk resources.

Download files as a Zip archive

To download the zip archive navigate to the review, changelist, file or folder:

> **Note**
> The **Download .zip** button is not displayed if the zip command-line tool is not installed on the Swarm server. For information about installing, and configuring the zip command-line tool, see Zip archive.

When you click the **Download .zip** button, Swarm performs the following steps:

1. Scans the files/folders:
    - Checks that you have permission to access their contents, according to the Helix Core server protections.
    - Checks that the total file size is small enough to be processed by Swarm.
2. Syncs the file contents to the Swarm server from the Helix Core server.
3. Creates the ZIP archive by compressing the file content.
4. Starts a download of the generated ZIP archive.

> **Note**
> - You might not see all of the above steps; Swarm caches the resulting ZIP archives so that repeated requests to the same files/folders can skip the sync and compress steps whenever possible.
> - If an error occurs while scanning, syncing, or compressing, Swarm indicates the error.

11 | Administration

This section covers administration and configuration of Swarm.

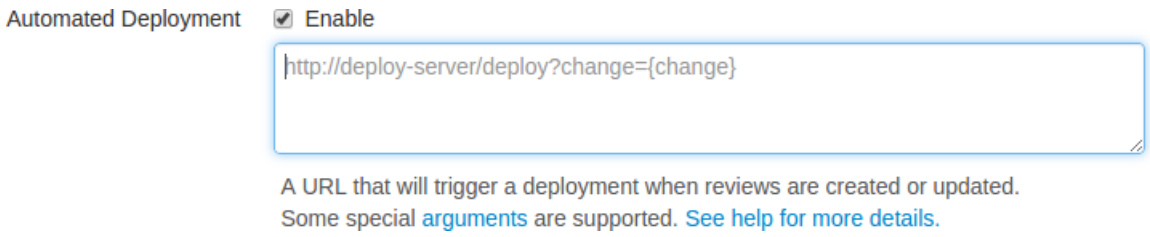
> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Automated deployment for reviews

Deploying code in a code review automatically involves enabling **Automated Deployment** in your project's configuration and providing a *trigger URL*. When the *trigger URL* is requested, Swarm expects a deployment program to be executed.

When the deployment processing ends, Swarm expects either a *success callback URL* or *failure callback URL* to be requested by your deployment program. These callback URLs should include a url parameter (either via GET or POST); when a valid-looking URL is included, clicking the deployment status indicator directs the user to the specified URL. This is intended to facilitate easy viewing of the successfully deployed review, or a report indicating why the deployment failed. The url parameter is mandatory for successful deployments, but is optional for failures.

1. Navigate to the project page.
2. Click the project **Settings** tab to display the **Project Settings** page.
3. **Automated Deployment** checkbox: select **Enable** to display the configuration field:



4. Provide a URL that triggers your deployment execution.

Special arguments are available to inform your deployment program of various details from Swarm:

> **Note**
> **Helix Plugin for Jenkins 1.10.11 and later:** Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Post Body** and select **URL Encoded**.

`{change}`
: The change number

`{status}`
: Status of the change, *shelved* or *submitted*

`{review}`
: The review's identifier

`{project}`
: The project's identifier

`{projectName}`
: The project's name

`{branch}`
: The branch identifier(s), comma-separated

`{branchName}`
: The branch name(s), comma-separated

`{success}`
: Deployment successful callback URL

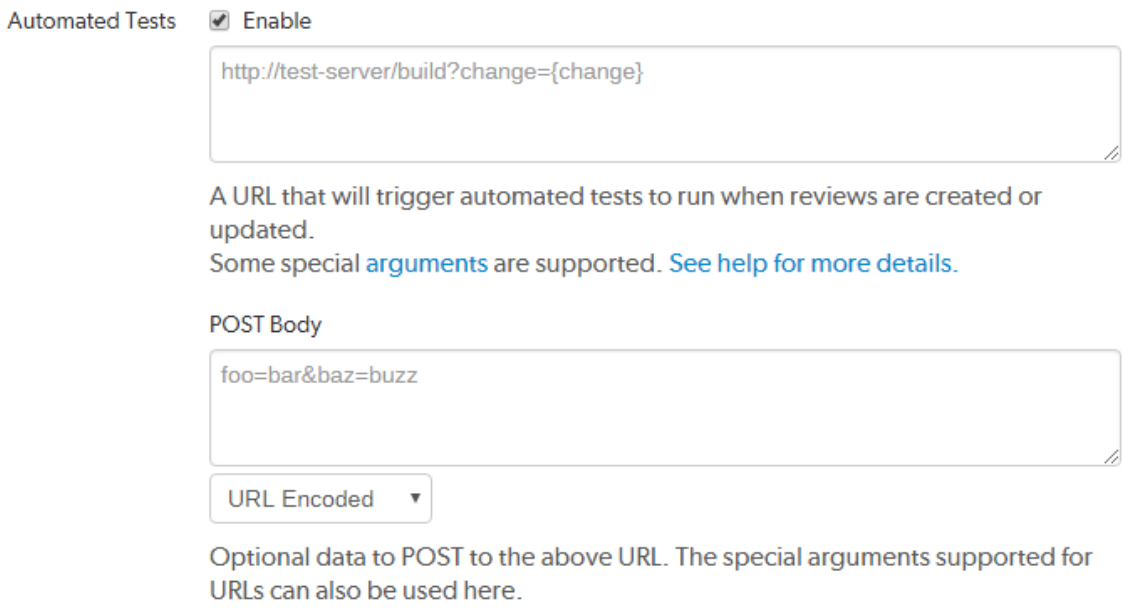
`{fail}`
: Deployment failure callback URL

Automated testing for reviews

> **Tip**
> From Swarm 2020.1, the preferred method for defining tests is on the Swarm "Tests" on page 440 page. This enables you to:
>
> - Associate a test with a workflow to ensure that the test is run when a review associated with that workflow is either created/updated or submitted.
> - Associate a test with the global workflow to ensure that the test is run whenever a Swarm review is either created/updated or submitted. This ensures that the global tests are enforced for all changes even if they are not part of a project.

Integrating Helix Swarm with a test suite involves enabling **Automated Tests** in your project's configuration and providing a *trigger URL*. When the *trigger URL* is requested, Swarm expects your test suite to be executed. When the tests complete, Swarm expects an *update callback URL*, a *pass callback URL*, or a *fail callback URL* to be requested by your test suite.

1. Navigate to the **Project** page.
2. Click the project **Settings** tab to display the **Project Settings** page.
3. Ensure that paths in each named branch configured for the project do not overlap with paths in other named branches.
4. **Automated tests** checkbox: select **Enable** to display the configuration fields:



5. Provide a URL that triggers your test suite execution.

   Special arguments are available to inform your test suite of various details from Swarm. Swarm automatically replaces the arguments with the relevant Swarm information when it calls the test:

> **Note**
> **Helix Plugin for Jenkins 1.10.11 and later:** Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Post Body** and select **URL Encoded**.

`{test}`
: The name of the test

`{testRunId}`
: The test run id

`{change}`
: The change number

`{status}`
: Status of the change, *shelved* or *submitted*

`{review}`

The review's identifier

{version}
The version of the review

{description}
The change description of the change used to generate this update. **{description}** cannot be used in the **URL**, it can only be used in the **POST Body**.

{project}
The project's identifier

{projectName}
The project's name

{branch}
The branch identifier(s) impacted by the review, comma-separated

{branchName}
The branch name(s) impacted by the review, comma-separated

{update}
The update callback URL. You can include any or all of the following when calling the update url to update the test run: status, messages, and a url in the body that links to the CI system for that run. They should be formatted in JSON in the body of the POST request. **Status:** valid status values are **running**, **pass**, and **fail**. **Messages:** you can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated. **{update}** is the preferred option for Swarm 2019.3. For more details, see the note below.

{pass}
Tests pass callback URL. From Swarm 2019.3, **{update}** is preferred. For more details, see the note below.

{fail}
Tests fail callback URL. From Swarm 2019.3, **{update}** is preferred. For more details, see the note below.

> **Note**
> - Swarm 2019.3 and later still supports **{pass}** and **{fail}**, however **{update}** is preferred because you can also include a message with the test status.
> - **{update}**, **{pass}**, and **{fail}** are composed automatically by Swarm. They include Swarm's own per-review authentication tokens.

6. **Optional:** specify any parameters that your automated tests require that must be sent via HTTP POST in the **POST Body** field. The POST parameters can include the special arguments listed above.

   Select the format of the POST parameters, either **URL Encoded** or **JSON Encoded**.

   - **URL Encoded:** POST parameters are parsed into name=value pairs.
   - **JSON Encoded:** parameters are passed raw in the POST body.

Configuring Jenkins for Swarm integration

Important
Your Jenkins host needs to be able to communicate with the Swarm host, and the Swarm host needs to be able to communicate with the Jenkins host. Ensure that the appropriate DNS/host configuration is in place, and that each server can reach the other via HTTP/HTTPS.

1. Install the p4-plugin for Jenkins:

   For instructions on installing the p4-plugin, see in the Installation chapter of the Helix Plugin for Jenkins Guide.

2. Configure a Jenkins project:

   Note
   Helix Plugin for Jenkins 1.10.11 and later: Swarm must send the parameters for the build to Jenkins as a POST request. To do this, enter the parameters in the **Post Body** and select **URL Encoded**.

   a. Specify the job name so that it matches the project identifier used in the trigger URL, as defined below.

      For example, the computed value of `{projectName}_{branchName}`.

      Or, edit the trigger URL to use the Jenkins job name you specify.

   b. Make the build parameterized to accept these parameters (note that these are named to match up with the script that is called):

      `{test}`
      : The name of the test

      `{testRunId}`
      : The test run id

      `{status}`
      : Whether the changelist to be tested is *shelved* or *submitted*

      `{change}`
      : Changelist # to run tests against

      `{review}`
      : The review's identifier

      `{version}`
      : The version of the review

      `{branchName}`
      : The branch name(s) impacted by the review, comma-separated

      `{update}`
      : The URL to POST to update the test run. You can include any or all of the following when calling the update url to update the test run: status, messages, and a url in the body that links to the CI system for that run. They should be formatted in JSON in the body of the POST request. **Status:** valid status values are `running`, `pass`, and

fail. **Messages:** you can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated. **{update}** is the recommended way of reporting test status.

Important
If your test system cannot POST to Swarm, you cannot use **update** and you must use **pass** and **fail** instead.

Note
When using {update}:

- If your build script has access to any messages related to the test execution, pass the messages to Swarm using the **{update}** URL. Swarm uses the provided message(s) to add to the test results.
- If your build script has access to the results of test execution, include a POST parameter called **url** when calling the update URL. Swarm uses the provided url to link reviews to the test results. Valid test status values are **running**, **pass**, and **fail**.
- The **{update}** callback url accepts a JSON body where you can specify any or all of the following: messages, url, and status.

```
{
    "messages" : ["My First Message", "My Second
Message"],
    "url" : "http://jenkins_host:8080/link_to_run",
    "status": "pass"
}
```

{pass}
The URL to wget if the build succeeds. From Swarm 2019.3, **{update}** is preferred. For more details, see the note below.

{fail}
The URL to wget if the build fails. From Swarm 2019.3, **{update}** is preferred. For more details, see the note below.

Note
When using {pass} and {fail}: if your build script has access to the results of test execution, include a GET or POST parameter called **url** when calling the pass or fail URLs. Swarm uses the provided url to link reviews to the test results.

> **Tip**
> Swarm 2019.3 and later still supports `{pass}` and `{fail}`, however `{update}` is preferred because you can also include a message with the test status.

c. Select **Perforce Software** for the **Source Code Management** section.

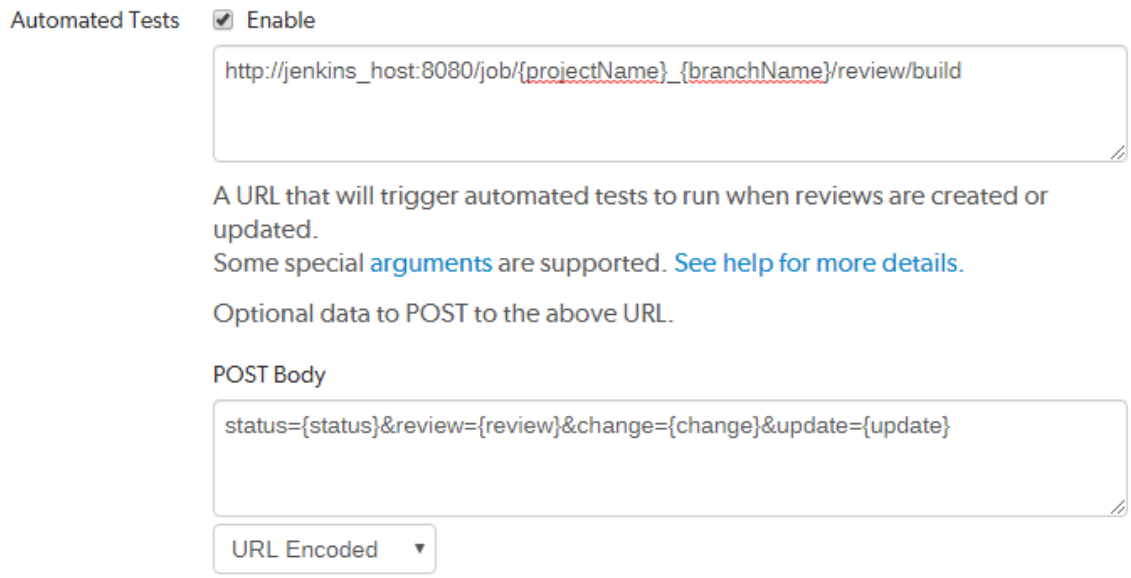
> **Important**
> You might see **Perforce** in the **Source Code Management** section. This represents an earlier community-provided Perforce plugin that does not include support for Swarm.

d. Set up credentials and workspace behavior as needed.

For instructions on configuring credentials and workspaces, see the Credentials and Workspaces sections of the Helix Plugin for Jenkins Guide.

> **Important**
> - If your Helix server is configured for Helix Authentication Service, the service user credentials used for automated testing must not use Helix Authentication Service.
> - The client workspace configured in Jenkins must have a view that includes the paths defined for that branch in Swarm.

3. Configure your Swarm project to run automated tests with a URL and parameters like this:



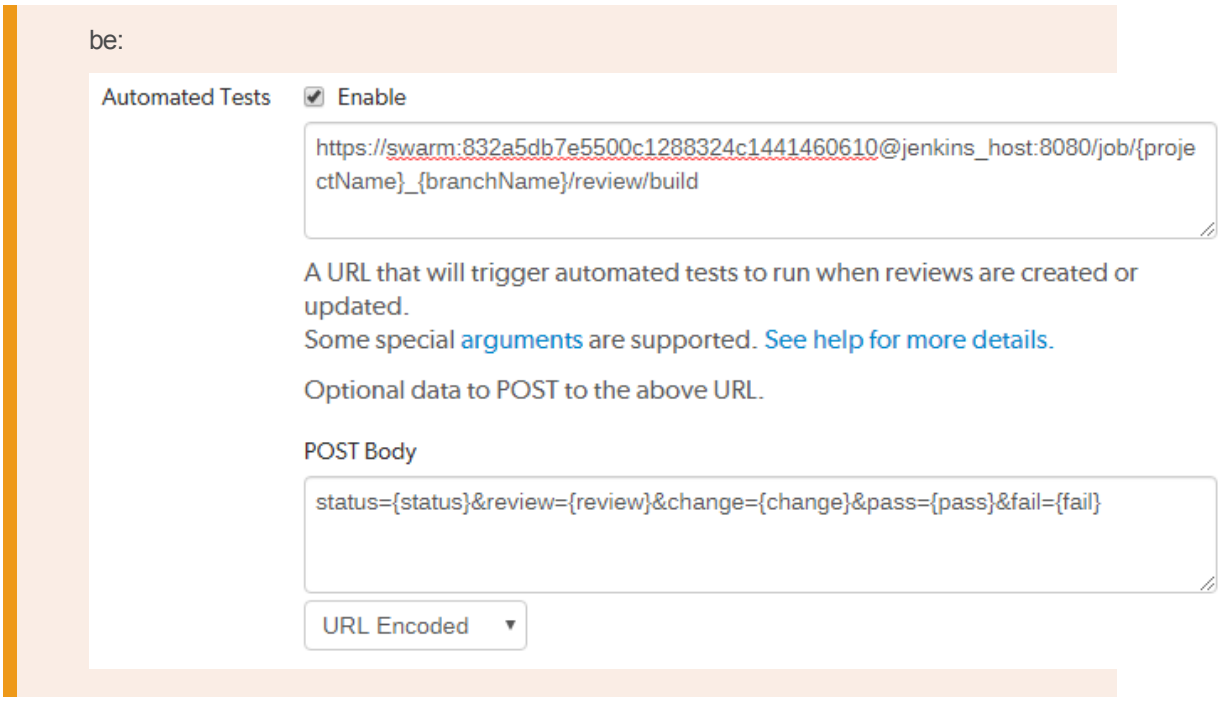
Important
For Jenkins, the job name needs to match the job identifier in the URL. In the example above, this is the computed value of **`{projectName}_{branchName}`**.

If you prefer a different naming scheme in Jenkins, replace **`{projectName}_{branchName}`** in the URL above with the project name actually defined in Jenkins.

Important
If security is enabled in Jenkins, the trigger URL needs to include credentials. Follow these steps:

a. Create a Jenkins user that will trigger Swarm builds. For example swarm.
b. Log into Jenkins as the new user.
c. Click on the user's username in the Jenkins toolbar.
d. Scroll down to **API Token**.
e. Click **Show API Token**.
f. Incorporate the value of the **API Token** into the Swarm trigger URL.

   For example, if the username is swarm and the API Token value is 832a5db7e5500c1288324c1441460610, the Swarm trigger URL and parameters should



Avatars

Swarm uses *avatars*, images that represent users and groups responsible for events in activity streams, projects, reviews, etc.

Avatars are retrieved from an avatar provider; the default provider is gravatar.com. Swarm sends an identifier to the avatar provider (for **`gravatar.com`**, an MD5 hash of the user's or group's email address), and the provider returns the configured image (if one exists). If no avatar is defined with the provider or the requests fails for any reason, Swarm selects an avatar from its internal collection.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Configure the avatar lookups in the avatars configuration block in the *`SWARM_ROOT`***`/data/config.php`** file. Here is an example:

```
<?php
    // this block should be a peer of 'p4'
    'avatars' => array(
        'http_url'  => 'http://www.gravatar.com/avatar/{hash}?s={size}&d={default}',
```

```
        'https_url' => 'https://secure.gravatar.com/avatar/{hash}?s=
{size}&d={default}',
    ),
```

- **`http_url`**: specifies the **`http`** URL Swarm uses to lookup avatars. If the URL is not defined, Swarm uses the default **`gravatar.com`** URL.
- **`https_url`**: specifies the **`https`** URL Swarm uses to lookup avatars. If the URL is not defined, Swarm uses the default **`gravatar.com`** URL.

The **`avatars`** array URL that Swarm uses depends on the settings of the Swarm Apache server protocol and the "external_url" on page 497 protocol, the securest of these protocols is used.

Several replacement values are available for inclusion in the URLs:

- **`{user}`**

  The current Swarm userid, Perforce groupid, or empty string

- **`{email}`**

  The current Swarm user's or group's email address, or empty string

- **`{hash}`**

  The MD5 hash of the Swarm user's or group's email address, or **`00000000000000000000000000000000`** if no email address is configured

- **`{default}`**

  The value blank for a transparent GIF (allowing users or groups without avatars to fallback to Swarm's internal avatars) or the value **`mm`** for a *mystery man* used in circumstances where no user or group identifier is known

- **`{size}`**

  the size Swarm would like in pixels for both the width and height, without units, e.g. 64

The URL you specify must include one of **`{user}`**, **`{email}`**, or **`{hash}`** to properly select a user-specific or group-specific avatar. The URL should include **`{size}`** to assist Swarm's presentation. **`{default}`** is not necessary, but helps provide a consistent avatar experience.

Tip
You can adjust the appearance of avatars by using custom CSS, see "Adjust the appearance of avatars" on page 659.

Note
By default, gravatar.com serves only G-rated avatar images. If your Swarm users and groups wish to use PG-, R-, or X-rated images, you need to configure the avatar lookup URLs with the appropriate rating flag. For example, to allow avatars with G or PG ratings, the configuration would look like:

```
<?php
    // this block should be a peer of 'p4'
    'avatars' => array(
        'http_url'  => 'http://www.gravatar.com/avatar/{hash}?r=pg&s=
{size}&d={default}',
        'https_url' => 'https://secure.gravatar.com/avatar/
{hash}?r=pg&s={size}&d={default}',
     ),
```

For more information on gravatar.com image requests, see:
https://en.gravatar.com/site/implement/images

Disable avatar lookups

If you wish to disable avatar lookups altogether and simply use Swarm's internal avatars, set each URL to `''` (empty string). For example:

```
<?php
    // this block should be a peer of 'p4'
    'avatars' => array(
        'http_url'  => '',
        'https_url' => '',
    ),
```

Backups

Swarm stores all of the information it requires to operate within the Helix server. This includes project definitions, code reviews, comments, followers, and more. Code reviews are largely built on top of Helix server's shelving feature, and most other records are stored in custom counters called *keys*.

Therefore, the standard recommendations for backing up your Helix server also apply when backing up your Swarm data.

For more information about backup and recovery, see Backup and recovery in *Helix Core Server Administrator Guide*.

In addition, your Swarm configuration and any modifications you might make to the provided modules, templates, CSS, JavaScript, etc. also need to be backed up. The *SWARM_ROOT*`/data` directory contains the configuration, as well as temporary working files and browser session storage.

Changelist files limit

Changelists or reviews with many files present a challenge to Swarm; it can take a notable amount of time for the file listing to reach Swarm from the Helix server, and a notable amount of time to provide the file listing HTML to a user's browser. When the file listing is many thousands of files, Swarm may run out of memory. Increasing the amount of memory for Swarm can help, but the Swarm interface may work slowly or not at all depending on the amount of memory available to the user's browser.

With the 2015.1 release, Swarm limits the number of files presented, even if a changelist or review involves more files than the limit.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

You can adjust the limit to suit your needs, by specifying a value for `max_changelist_files` in the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    'p4' => array(
         'max_changelist_files'  => 1000,
),
```

The default value of `1000` should suffice for most Swarm installations. Consider the effects of changing this value:

- Problems in the Swarm UI still exist with arbitrarily large values; there is likely no advantage to using a value over `10000`.

  Performance will be better with Helix server 2014.1 or later, as the file listing will be limited by Helix server.
- Smaller values can potentially interfere with reviews and reading changelists; the file limit may cause interesting files in a changelist or review to no longer be displayed in Swarm.

Client integration

P4V and P4VS can integrate with Swarm. To indicate how these applications should connect with Swarm, Swarm sets the `P4.Swarm.URL` property set in Helix server. P4V and P4VS read this property, and if set, they connect to the specified URL to make Swarm API calls. If the property is unset, Swarm integration features are disabled.

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

When `P4.Swarm.URL` is set, P4V provides the following integration features:

- **Request a review**: requests a review for pending or committed changelists.
- **Update a review**: updates a review from the current state of a pending changelist. This works for changelists that are already associated with a review, or for unassociated changelists.
- **Open review in Swarm**: opens the review associated with the selected changelist in your system's default web browser.
- **Review Id and State columns**: adds **Review Id** and **Review State** columns to both the **Pending** and **Submitted** tabs.

By default, the first Swarm worker auto-detects the URL it is running under and sets `P4.Swarm.URL` accordingly.

For customized Swarm installations, the auto-detected URL may not use the correct hostname or port. In these scenarios, you can disable the URL auto-detection by editing the *SWARM_ROOT*`/data/config.php` file and setting the `auto_register_url` item to `false` in the p4 configuration block. For example:

```
<?php
    'p4' => array(
        'auto_register_url'   => false,
    ),
```

If you choose to disable this feature, you should manually set the `P4.Swarm.URL` property in Helix server to the URL for your Swarm installation:

```
$ p4 property -a -n P4.Swarm.URL -v https://myswarm.url:port/
```

Replace ***https://myswarm.url:port/*** with the URL for your Swarm installation.

Note
P4V uses an integration timeout, specified in the `P4.Swarm.Timeout` property, to limit delays in the P4V user interface. The default timeout is 10 seconds.

To change the integration timeout, run:

```
$ p4 property -a -n P4.Swarm.Timeout -v 10
```

Replace the ***10*** with the desired timeout in seconds. Increasing the timeout could cause notable delays in the P4V user interface, and decreasing the timeout could cause sporadic integration failures if Swarm's API responses take longer than the specified timeout.

Comment attachments

Swarm supports attaching arbitrary files to comments in code reviews and jobs.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Comment attachment storage

To store files attached to comments, Swarm looks for a depot named **`//.swarm`**. As Swarm does not create this depot, you need to create it, or specify another depot that the Swarm *admin* user can write to.

The depot used to store Swarm attachments must be:

- a classic depot (**`stream`**, **`unload`**, **`archive`**, and **`spec`** depots are not supported)
- a **`local`** depot (**`remote`** depots are not supported)
- a depot that the Swarm *admin* user can write to but other users cannot access. For more information about permissions, see Authorizing access in the *Helix Core Server Administrator Guide*.

  > **Tip**
  > It is best practice to use a depot directory that is not used to store any other files, this stops other users having access to the attachments stored in the depot. We recommend that you use **`//.swarm`**

Create a directory for the comment attachments

We recommend that you use **`//.swarm`** because it is best practice to use a depot directory that is not used to store any other files This prevents other users from having direct access to the comment attachments.

1. To create a **`//.swarm`** depot, run the following as a user with *admin*-level privileges:

   ```
   $ p4 depot .swarm
   ```

2. Ensure that the Swarm *admin* user can write to the **`//.swarm`** depot and make sure that other users cannot access it.

   For information on creating depots, see Working with depots in *Helix Core Server Administrator Guide*.

Specify the location for the comment attachments

By default Swarm looks for a depot named **//.swarm**, to specify a different depot path for comment attachments, use the **depot_storage** configuration block in the ***SWARM_ROOT*/data/config.php** file.

Replace **depot_name** with the depot where comment attachments are stored. The Swarm *admin* needs to be able to write to this depot but make sure that other users cannot access it:

```
<?php
    // this block should be a peer of 'p4'
    'depot_storage' => array(
        'base_path'  => '//depot_name',
    ),
```

> **Tip**
> The **base_path** can be more than just the depot name, for example **//depot/perforce/.swarm** would work.

Maximum size of comment attachments

You can limit the size of comment attachments with the **attachments** configuration block in the ***SWARM_ROOT*/data/config.php** file:

```
<?php
    // this block should be a peer of 'p4'
    'attachments' => array(
        'max_file_size'  => 0, // the maximum file size to accept in bytes
    ),
```

Replace the *0* with the maximum file size in bytes that you want Swarm to accept for a comment attachment. If the file size is exceeded, users will see an error.

> **Note**
> Be aware that PHP's **upload_max_filesize** setting in *SWARM_ROOT*/public/.htaccess** overrides **max_file_size** (which overrides the setting in PHP's **php.ini**). You can only use **max_file_size** to be more restrictive than the setting in *SWARM_ROOT*/public/.htaccess**.
>
> **The default for upload_max_filesize is 8 MB (8 megabytes). Increase this limit if your commenters need to upload larger files.**

You may also have to increase **post_max_size**. **post_max_size** should always be set larger or equal to **upload_max_filesize**, and Swarm's **max_file_size** should always be either unset, or set smaller or equal to **upload_max_filesize**, otherwise users will encounter unexpected rejection of their comment attachments.

See Handling file uploads: Common Pitfalls for more details.

Comment mentions

This section describes how to customize how Swarm treats user *@mentions* and group *@@mentions* in auto-complete drop-downs, comments, changelists, and review descriptions.

Note

- *@mentions* for users is only available for Swarm 2017.1 and later.
- *@@mentiions* for groups is only available for Swarm 2017.3 and later.
- If your Helix Core server is configured to be case sensitive this will also apply to the names specified in your exclude lists.

Tip

If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

By default:

- When a user uses an *@mention* or *@@mention* in a comment, Swarm will auto-complete for all Swarm users or groups respectively.
- When a user edits the reviewers on a review, Swarm will auto-complete for all Swarm users and groups.
- When a user edits the author of a review, Swarm will auto-complete for all Swarm users.

This behavior can be fine tuned with the **mentions** configuration block in the *SWARM_ROOT*/**data/config.php** file. For example:

```
<?php
    'mentions' => array(
        'mode' => 'global',
        'user_exclude_list' => array('super', 'swarm-admin'),
        'group_exclude_list' => array('testers', 'writers'),
    ),
```

- **`mode`** controls the scope of the auto-complete dropdown that is shown when the user starts typing an @*mention* or an @@*mention* in a review comment:
  - **`global`** displays all users/groups in Swarm, this is the default value
  - **`projects`** only displays project members (users and groups) in the auto-complete list
  - **`disabled`** disables the auto-complete drop down so that it is not displayed
- **`user_exclude_list`** users in the exclude list:
  - are not suggested when you @*mention* them in a comment, edit reviewers, or edit the author in Swarm.
  - are not added to the review if you manually add them as an @*mention* in a Swarm comment or a review description.
  - are not added to the review if they are @*mentioned* in a changelist description.
  - cannot be added to the review with the Swarm API.

  > **Tip**
  > **When you add a user to the exclude list:**
  > - the user is not removed from any reviews that they are already on.
  > - the user will stop receiving email notifications for those reviews.

- **`group_exclude_list`** groups in the exclude list:
  - are not suggested when you @@*mention* them in a comment or edit reviewers in Swarm.
  - are not added to the review if you manually add them as an @@*mention* in a Swarm comment or review description.
  - are not added to the review if they are @@*mentioned* in a changelist description.
  - cannot be added to the review with the Swarm API.

  > **Tip**
  > **When you add a group to the exclude list:**
  > - the group is not removed from any reviews that it is already on.
  > - the group members will stop receiving email notifications for those reviews.
  > - if a member of an excluded group is also on the review as a user or as a member of another group, that user will continue to receive emails for the review.

Regular expressions in exclude lists

Regular expressions can be used in user and group exclude lists but they should be used with care to avoid accidentally excluding the wrong users or groups. Swarm exclude lists can use any valid PHP regular expression but not all of them make sense for the exclude lists.

Note
Swarm adds a `^` character before your regular expression and a `$` character after it. For example: `^your-reg-ex-pattern$` You must take this into account when constructing your regular expressions.

Tip

- You can test the results of your regular expressions before using them, see https://regex101.com/
- Case sensitivity for regular expressions in the exclude lists is determined by the case sensitivity setting of the Helix server.

Regular expression examples

Useful regular expressions are shown below:

Exact match only:

`'super'` only users or groups named *super* are excluded.

Beginning with:

`'admin.*'` users or groups beginning with *admin* are excluded.

For example: the following would be excluded:

administrator, *admin-group*, and *admin*

Ending with:

`'.*admin'` users or groups ending with *admin* are excluded.

For example: the following would be excluded:

test-admin, *buildadmin*, and *admin*

Ending with digits

`'.*[0-9]+'` users or groups ending with digits are excluded.

For example: the following would be excluded:

test03, *admin-10*, *tester_123456*

Containing:

`'.*admin.*'` users or groups that contain *admin* are excluded.

For example: the following would be excluded:

test-admin, *buildadmin*, *administrator*, *admin-group*, and *admin*

Comments

This section provides information on how to delay comment notifications, and configure comment threading.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Comment notification delay

By default, comment notifications are delayed to allow reviewers to add or edit comments as they progress through a review without sending a notification for each individual comment on the review. Comment notifications are rolled up into a single notification and sent either manually by the reviewer, or automatically after the notification delay time has been exceeded.

The delay countdown is reset each time the reviewer adds or edits a comment on the review, by default the notification delay time is set to 30 minutes.

- If you are commenting on more than one review, each of the reviews that you are commenting on has its own notification delay countdown that only applies to the comments that *you* make on *that* review.
- If another reviewer is making comments on the same review as you, *that* reviewer has their own notification delay timer for *that* review.
- If you manually send a delayed comment notification, the notification will only contain the comments that *you* made on *that* review.

> **Tip**
> The comment notification delay does not delay the posting of the comments, only the comment notification is delayed.

> **Note**
> Comment notifications are only delayed for comments on reviews. Comments on commits or jobs produce notifications immediately.

To change the comment notification delay time, update the *SWARM_ROOT*`/data/config.php` file to include the following configuration item within the comments block:

```
<?php
    'comments' => array(
```

```
        'notification_delay_time' => 1800, //Default to 30 minutes 1800
seconds
    ),
```

notification_delay_time: Specifies the comment notification delay time in seconds.

- Set to **0** to send the comment notification immediately the **Post** button is clicked.
- The default value if **notification_delay_time** is not specified is **1800** seconds (30 minutes).

Comment threading

By default, you can reply to comments, replies are displayed in a thread below the parent comment. Comment thread depth is set to 4 by default, this means you can have up to 4 levels of replies for a parent comment. The **Reply** link is not displayed for replies at or above the maximum thread depth set for Swarm. If the parent comment is archived, replies are archived with the parent, see "Archiving comments" on page 280. Comment threading depth is set by the **max_depth** configurable.

> **Tip**
> If the thread depth is reduced by a Swarm administrator, earlier replies at a deeper level will continue to be displayed but you cannot reply to them.

To configure comment threading, update the *SWARM_ROOT*/**data/config.php** file to include the following configuration item within the comments block:

```
<?php
    'comments' => array(
        'threading' => array(
            'max_depth' => 4, // default depth 4, to disable comment
threading set to 0
        ),
    ),
```

max_depth: Specifies maximum depth of a comment thread.

- Set to **0** to disable comment threading.
- The default value if **max_depth** is not specified is **4**.

Commit credit

When you use Swarm to commit a review, but you are not the review's author, Swarm gives credit to the review author by default. Activity stream entries and email notifications include both the committer and review author's details.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

If you prefer Swarm's original behavior, which was to give credit only to the committer, you can do so by editing the *SWARM_ROOT*`/data/config.php` file and setting the `commit_credit_author` item to `false` in the `reviews` configuration block. For example:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'commit_credit_author'  => false,
    ),
```

Commit-edge deployment

Swarm can connect to a Helix Core server configured to use the *commit-edge architecture*, which is a specific replication configuration that employs a *commit* server and one or more *edge* servers. This configuration distributes the compute, storage, and network requirements for improved performance and geographic distribution.

For more information on Helix server's commit-edge architecture, see the Commit-edge chapter in the *Helix Core Server Administrator Guide*.

> **Tip**
> P4V users may have problems when P4V connects to an edge server and your commit and edge tickets are different. For more information on P4V authentication for commit-edge deployment, see "P4V Authentication" on the facing page.

Swarm commit-edge configuration

This section describes what configuration changes need to be made to Swarm and the Helix servers to run Swarm with a commit-edge deployment. Before you begin making changes ensure that your Helix server commit-edge deployment is working correctly, and that Swarm has been installed and configured.

Configure Swarm for a commit-edge deployment:

1. Configure Swarm to connect to the Helix Core Commit server, see "Swarm configuration" on page 154. This is called the **Commit Server Swarm Instance**.

When Swarm is connected to the **Commit Server Swarm Instance**, the first worker detects this situation and sets a key in the Helix server, **`P4.Swarm.CommitURL`**, to an auto-detected URL.

2. **Configure the Swarm triggers:**
   a. Configure the Swarm triggers on all of the Commit and Edge servers, see "Helix Core server configuration for Swarm" on page 158.

   > **Important**
   > - Save all of the Swarm trigger scripts in exactly the same place on each of the Commit and Edge servers. This is important because they all share the same trigger table.
   > - You must install the trigger dependencies on all of the Commit and Edge servers, see "Trigger dependencies" on page 109.

   b. Configure **`SWARM_HOST`** in the **`swarm-trigger.conf`** file of all of the Commit and Edge servers to point to the **Commit Server Swarm Instance**, see "Helix Core server configuration for Swarm" on page 158.
3. Configure all of the Commit and Edge servers to point to the **Commit Server Swarm Instance** URL. This is done by setting the **`P4.Swarm.URL`** to the **Commit Server Swarm Instance** URL on each of the Commit and Edge servers, see Swarm integration properties in the *Helix Core Server Administrator Guide*.
4. Your Swarm configuration is now complete but you must check that it is working correctly before using it in production, see "Validate your Swarm installation" on page 192.

P4V Authentication

When using P4V's Swarm integration in a commit-edge deployment, users may encounter authentication errors; such errors can result from incorrect configuration of login tickets in distributed environments. Essentially, the problem is that while P4V is connected to an edge server, Swarm is connected to the commit server, and the login tickets do not match.

If P4V users see the **`error Host requires authentication`**, the solution we recommend is to forward login requests to the commit server. This can be achieved by executing the following two commands as a user with *operator* or *super* privileges in the Helix server:

```
$p4 configure set auth.id=authid
$p4 configure set rpl.forward.login=1
```

Replace ***`authid`*** with the authentication identifier for your Helix server.

For more information, see our Knowledge Base article Single Ticket Login in Distributed Environments, and the `p4 serverid` command in the *Helix Core P4 Command Reference*.

Commit timeout

When a code review contains many files, or large files, or both, committing the review within Swarm can take some time. The default configuration, within the *SWARM_ROOT*`/data/config.php` file:

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'commit_timeout'  => 1800, // 30 minutes
    ),
```

The `commit_timeout` key is expressed in seconds. If a commit operation takes longer than this limit, it is terminated. It is likely that a terminated commit requires administrator intervention to complete the commit using another client.

Configuration overview

This section provides an overview of all the possible configuration blocks in the *SWARM_ROOT*`/data/config.php` file. Click on any link in the configuration overview below to see a detailed description of that configurable.

> **Warning**
> While the syntax of this example is correct, it includes configuration values that **cannot work**. Ensure that you adjust the configuration appropriately for your Swarm installation before using this example in testing or production.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

```
<?php
    return array(
        'activity' => array(
            'ignored_users' => array(
                'git-fusion-user',
```

```
                'p4dtguser',
                'system',
            ),
        ),
        'archives' => array(
            'max_input_size'    => 512 * 1024 * 1024, // 512M (in bytes)
            'archive_timeout'   => 1800,              // 30 minutes
            'compression_level' => 1,                 // 0-9
            'cache_lifetime'    => 60 * 60 * 24,      // 1 day
        ),
        'attachments' => array(
            'max_file_size'  => 0, // the maximum file size to accept in
bytes
        ),
        'avatars' => array(
            'http_url'  => 'http://www.gravatar.com/avatar/{hash}?s=
{size}&d={default}',
            'https_url' => 'https://secure.gravatar.com/avatar/{hash}?s=
{size}&d={default}',
        ),
        'comments' => array(
             'notification_delay_time'    => 1800, //Default to 30 minutes
1800 seconds
             'threading'     => array(
                 'max_depth' => 4, // default depth 4, to disable comment
threading set to 0
             ),
        ),
        'depot_storage' => array(
            'base_path'  => '//depot_name',
        ),
        'diffs' => array(
            'max_diffs'                 => 1500,
        ),
        'environment' => array(
            'mode'         => 'production',
```

```
            'hostname'     => 'myswarm.hostname',
            'external_url' => null,
            'base_url'     => null,
            'logout_url'   => null, // defaults to null
            'vendor'       => array(
                'emoji_path' => 'vendor/gemoji/images',
            ),
        ),
        'files' => array(
            'max_size'         => 1048576,
            'download_timeout' => 1800,
            'allow_edits' => true, // default is true
        ),
        'groups' => array(
            'super_only'  => true, // ['true'|'false'] default value is
'false'
        ),
        'http_client_options'   => array(
            'timeout'       => 10, // default value is 10 seconds
            'sslcapath'     => '', // path to the SSL certificate
directory
            'sslcert'       => '', // the path to a PEM-encoded SSL
certificate
            'sslpassphrase' => '', // the passphrase for the SSL
certificate file
            'hosts'         => array(), // optional, per-host overrides.
Host as key, array options as values
        ),
        'jira' => array(
            'host'      => '',
            'user'      => '',
            'password'  => '',
            'job_field' => '',
            'link_to_jobs'    => false, // set to true to enable Perforce
job links in JIRA, P4DTG and job_field required
```

```
            'delay_job_links' => 60, // delay in seconds, defaults to 60
seconds
            'relationship'    => '', // subsection name for Perforce job
links. Defaults to empty, job links added to the "links to" subsection
        ),
        'log' => array(
            'priority'     => 3, // 7 for max, defaults to 3
            'reference_id' => false // defaults to false
        ),
        'mail' => array(
            // 'recipients' => array('user@my.domain'),
            'notify_self'   => false,
            'transport' => array(
                'host' => 'my.mx.host',
            ),
        ),
        'markdown' => array(
            'markdown' => 'safe', // default is 'safe'
'disabled'|'safe'|'unsafe'
        ),
        'mentions' => array(
            'mode'  => 'global',
            'user_exclude_list'  => array('super', 'swarm-admin'),
            'group_exclude_list' => array('testers', 'writers'), //
defaults to empty
        ),
        'menu_helpers' => array(
            'MyMenu01' => array( // A short recognizable name for the menu
item
                'id'        => 'custom01',            // A unique id for
the menu item. If not included in the array, parent array name is used.
                'enabled'   => true,                  // ['true'|'false']
'true' makes the menu item visible. 'true' is the default if not included
in the array.
                'target'    => '/module/MyMenuItem/', // The URL or custom
```

```
module route a menu click takes you to.
                                                        // If not included
in array, id is used. If id not included, parent array name is used.
                'cssClass'  => 'custom_menu',           // The custom CSS
class name added to the menu item, appended to h2.menu- in Swarm CSS
                'title'     => 'MyMenuItem',            // The text that
will be shown on the button.
                                                        // If not included
in array, id is used. If id not included, parent array name is used.
                'class'     => '',                      // If not included
in array or empty, the menu item is added to the main menu.
                                                        // To add the menu
item to the project menu for all of the projects, set to
'\Projects\Menu\Helper\ProjectContextMenuHelper'
                'priority'  => 155,                     // The position the
menu item is displayed at in the menu.
                                                        // If not included
in the array, the menu item is placed at the bottom of the menu.
                'roles'     => null,                    //
['null'|'authenticated'|'admin'|'super'] If not included in the array,
null is the default
                                                        // Specifies the
minimum level of Perforce user that can see the menu item.
                                                        // 'authenticated' =
any authorized user, 'null' = unauthenticated users
            ),
        ),
        'notifications' => array(
            'honor_p4_reviews'      => false,
            'opt_in_review_path'    => '//depot/swarm',
            'disable_change_emails' => false,
        ),
        'p4' => array(
            'port'       => 'my-helix-core-server:1666',
            'user'       => 'admin_userid',
            'password'   => 'admin user ticket or password',
```

```
            'sso_enabled' => false, // defaults to false
            'proxy_mode' => true, // defaults to true
            'slow_command_logging' => array(
                3,
                10 => array('print', 'shelve', 'submit', 'sync',
'unshelve'),
            ),
            'max_changelist_files' => 1000,
            'auto_register_url'    => true,
        ),
        'projects' => array(
            'mainlines' => array(
                'stable', 'release', // 'main', 'mainline', 'master', and
'trunk' are hardcoded, there is no need to add them to the array
            ),
            'add_admin_only'           => false,
            'add_groups_only'          => array(),
            'edit_name_admin_only'     => false,
            'edit_branches_admin_only' => false,
            'readme_mode'              => 'enabled',
            'fetch'                    => array('maximum' => 0), //
defaults to 0 (disabled)
            'allow_view_settings'      => false, // defaults to false
        ),
        'queue'  => array(
            'workers'             => 3,    // defaults to 3
            'worker_lifetime'     => 595,  // defaults to 10 minutes (less
5 seconds)
            'worker_task_timeout' => 1800, // defaults to 30 minutes
            'worker_memory_limit' => '1G', // defaults to 1 gigabyte
        ),
        'redis' => array(
            'options' => array(
                'password' => null, // Defaults to null
                'namespace' => 'Swarm',
```

```
                'server' => array(
                    'host' => 'localhost', // Defaults to 'localhost' or
enter your Redis server hostname
                    'port' => '7379', // Defaults to '7379' or enter your
Redis server port
                ),
            ),
            'items_batch_size' => 100000,
            'check_integrity' => '03:00', // Defaults to '03:00' Use one
of the following options:
                                          //'HH:ii' (24 hour format with
leading zeros), the time the integrity check starts each day
                                          // positive integer, the time
between integrity checks in seconds. '0' = integrity check disabled
            'population_lock_timeout' => 300, // Timeout for initial cache
population. Defaults to 300 seconds.
        ),
        'reviews' => array(
            'patterns' => array(
                'octothorpe' => array(     // #review or #review-1234 with
surrounding whitespace/eol
                    'regex'  => '/(?P<pre>(?:\s|^)\(?)\#
(?P<keyword>review|append|replace)(?:-(?P<id>[0-9]+))?(?P<post>[.,!?:;)]*
(?=\s|$))/i',
                    'spec'   => '%pre%#%keyword%-%id%%post%',
                    'insert' => "%description%\n\n#review-%id%",
                    'strip'  => '/^\s*\#(review|append|replace)(-[0-9]+)?
(\s+|$)|(\s+|^)\#(review|append|replace)(-[0-9]+)?\s*$/i',
                ),
                'leading-square' => array(     // [review] or [review-
1234] at start
                    'regex'  => '/^(?P<pre>\s*)\
[(?P<keyword>review|append|replace)(?:-(?P<id>[0-9]+))?\](?P<post>\s*)/i',
                    'spec'  => '%pre%[%keyword%-%id%]%post%',
                ),
```

```
                'trailing-square' => array(     // [review] or [review-
1234] at end
                    'regex'  => '/(?P<pre>\s*)\
[(?P<keyword>review|append|replace)(?:-(?P<id>[0-9]+))?\]
(?P<post>\s*)?$/i',
                    'spec'   => '%pre%[%keyword%-%id%]%post%',
                ),
            ),
            'filters' => array(
                'filter-max' => 15,
                'result_sorting' => true,
                'date_field' => 'updated', // 'created' displays and sorts by
created date, 'updated' displays and sorts by last updated
            ),
            'cleanup' => array(
                'mode'        => 'user', // auto - follow default, user -
present checkbox(with default)
                'default'     => false,  // clean up pending changelists
on commit
                'reopenFiles' => false,   // re-open any opened files into
the default changelist
            ),
            'statistics' => array(
                'complexity' => array(
                    'calculation' => 'default', // 'default|disabled'
                    'high' => 300,
                    'low' => 30
                ),
            ),
            'commit_timeout'                 => 1800, // 30 minutes
            'disable_commit'                 => true,
            'moderator_approval'              => 'any', // 'any|each'
            'disable_self_approve'            => false,
            'end_states'                     => array('archived',
'rejected', 'approved:commit'),
            'disable_approve_when_tasks_open' => false,
```

```
            'process_shelf_delete_when'       => array(),
            'commit_credit_author'            => true,
            'ignored_users'                   => array(),
            'unapprove_modified'              => true,
            'allow_author_change'             => true,
            'allow_author_obliterate'         => false,
            'sync_descriptions'               => true,
            'expand_all_file_limit'           => 10,
            'expand_group_reviewers'          => false,
            'more_context_lines'              => 10, // defaults to 10
lines
        ),
        'search' => array(
            'maxlocktime'     => 5000, // 5 seconds, in milliseconds
            'p4_search_host'  => '',   // optional URL to Helix Search
Tool
        ),
        'security'  => array(
            'disable_system_info'      => false,
            'email_restricted_changes' => false,
            'emulate_ip_protections'   => true,
            'https_port'               => null,
            'https_strict'             => false,
            'https_strict_redirect'    => true,
            'require_login'            => true,
            'prevent_login'            => array(
                'service_user1',
                'service_user2',
            ),
        ),
        'session'  => array(
            'cookie_lifetime'            => 0, // 0=expire when browser
closed
            'remembered_cookie_lifetime' => 60*60*24*30, // 30 days
            'gc_maxlifetime'             => 60*60*24*30, // 30 days
```

```
            'gc_divisor'                => 100, // 100 user requests
        ),
        'short_links' => array(
            'hostname'     => 'myho.st',
            'external_url' => 'https://myho.st:port/sub-folder',
        ),
        'test_definitions' => array(
            'project_and_branch_separator' => ':',
        ),
        'translator' => array(
            'detect_locale'             => true,
            'locale'                    => array("en_GB", "en_US"),
            'translation_file_patterns' => array(),
            'non_utf8_encodings'        => array('sjis', 'euc-jp',
'windows-1252'),
            'utf8_convert' => true,
        ),
        'upgrade' => array(
            'status_refresh_interval' => 10,    //Refresh page every 10
seconds
            'batch_size' => 1000,       //Fetch 1000 reviews to lower memory
usage
        ),
        'users' => array(
          'dashboard_refresh_interval' => 300000, //Default 300000
milliseconds
           'display_fullname'           => false,
           'settings' => array(
              'review_preferences' => array(
                  'show_comments_in_files'             => true,
                  'view_diffs_side_by_side'            => true,
                  'show_space_and_new_line_characters' => false,
                  'ignore_whitespace'                  => false,
              ),
              'time_preferences' => array(
```

```
                'display'  => 'Timeago', // Default to 'Timeago' but can
be set to 'Timestamp'
            ),
        ),
    ),
    'workflow' => array(
        'enabled' => true, // Switches the workflow feature on.
Default is true
    ),
    'xhprof' => array(
        'slow_time'      => 3,
        'ignored_routes' => array('download', 'imagick',
'libreoffice', 'worker'),
    ),
    'saml' => array(...),
);
```

Note
The Swarm configuration file does not include PHP's standard closing tag (?>). This is intentional as it prevents unintentional whitespace from being introduced into Swarm's output, which would interfere with Swarm's ability to control HTTP headers. Debugging problems that result from unintentional whitespace can be challenging, since the resulting behavior and error messages often appear to be misleading.

Diff configuration

Swarm's Diffs feature can be customized via the following config item.

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

max_diffs

If the number of diffs is greater than the configured maximum, then the list of diffs is truncated by default. An option will be given to show all the diffs for that file.

```
    'diffs' => array(
        'max_diffs' => 1500,
    ),
```

Email configuration

Swarm's email delivery is controlled by the mail configuration block in the *SWARM_ROOT*`/data/config.php` file.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Example:

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        // 'sender' => 'swarm@my.domain',   // defaults to
'notifications@hostname'
        'transport' => array(
            'name' => 'localhost',          // name of SMTP host
            'host' => '127.0.0.1',          // host/IP of SMTP host
            'port' => 587,                  // SMTP host listening port
            'connection_class'  => 'plain', // 'smtp', 'plain', 'login',
'crammd5'
            'connection_config' => array(   // include when auth required
to send
                'username'  => 'user',      // user on SMTP host
                'password'  => 'pass',      // password for user on SMTP
host
                'ssl'       => 'tls',       // empty, 'tls', or 'ssl'
            ),
            // override email deliveries and store all messages in this
path
            // 'path' => '/var/spool/swarm',
        ),
```

```
        // override regular recipients; send email only to these addresses
        // 'recipients' => array(
        //     'user1@my.domain',
        //     'user2@my.domain',
        // ),

        // send notifications of comments to comment authors?
        'notify_self' => false,

        // blind carbon-copy recipients
        // 'use_bcc' => true,

        // suppress reply-to header
        // 'use_replyto' => false,

        // change the email subject prefix, the default prefix is '
[Swarm]'
        // 'subject_prefix' => '[Swarm]',

        // Control email thread indexing
        // 'index-conversations' => true, // ['true'|'false'] default is
true, email thread indexing is turned on
    ),
```

Note
Without any mail configuration in the *SWARM_ROOT*`/data/config.php` file, Swarm attempts to send email according to PHP's configuration, found in the `php.ini` file. By default, the configuration in `php.ini` relies on SendMail being installed.

Important
Email delivery for events related to restricted changes is disabled by default. See "Restricted Changes" on page 565 for details on how to enable restricted change notifications.

Sender

The sender item within the **mail** block specifies the sender email address that should be used for all notification email messages. The default value is:

```
notifications@hostname
```

hostname is the name of the host running Swarm, or when specified with the "Environment" on page 495 configuration.

Transport

The **transport** block within the **mail** block defines which mail server Swarm should use to send email notifications. Most of the items in this block can be omitted, or included as needed. See the Laminas Framework's Mail Transport Configuration Options for a description of most fields and their default values.

> **Note**
> Swarm supports the Laminas component versions in the **LICENSE.txt** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **LICENSE.txt** file is in the **readme** folder of your Swarm installation.

Swarm uses the custom path item to direct all email messages to a directory instead of attempting delivery via SMTP. For details, see "Save all messages to disk" on page 493.

Recipients

The **recipients** item within the **mail** block allows you to specify a list of recipients that should receive email notifications, overriding the normal recipients. This is useful if you need to debug mail deliveries.

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        'recipients' => array(
            'user1@my.domain',
            'user2@my.domain',
        ),
    ),
```

Any number of recipients can be defined. If the array is empty, email notifications are delivered to the original recipients.

notify_self

The `notify_self` item within the `mail` block specifies whether comment authors should receive an email for their comments. The default value is `false`. When set to `true`, comment authors receive an email notification for their own comments.

Use BCC

The `use_bcc` item within the `mail` block allows you to address recipients using the Bcc email field. Setting the value to `true` causes Swarm to use the `Bcc:` field in notifications instead of the `To:` field, concealing the email addresses of all recipients.

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        'use_bcc' => true,
    ),
```

Use Reply-To

The `use_replyto` item within the `mail` block allows you to suppress populating the Reply-To email field. Setting the value to `false` causes Swarm to omit the `Reply-To:` field in notifications; by default, it is populated with the author's name and email address. When this field is `true`, users receiving an email notification can simply reply to the email and their response will be addressed to the author.

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        'use_replyto' => false,
),
```

Email subject prefix

The `subject_prefix` item withing the `mail` block sets the prefix for the subject line of emails sent by Swarm. By default this is set to **[Swarm]**.

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        'subject_prefix' => '[Swarm]',
),
```

Save all messages to disk

For testing purposes, you may want to send all email to disk without attempting to send it to recipients. Use the following configuration block to accomplish this:

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        'transport'  => array('path' => MAIL_PATH),
    ),
```

<MAIL_PATH> should be replaced with the absolute path where email messages should be written, the mail transport will create a new file for each email message and write it to the path directory. This path must already exist and be writable by the web server user.

> **Note**
> Use of the path item causes Swarm to ignore **all** other configuration within the transport block. This is why path is commented out in the main example.

Email headers

Swarm sends out notification emails that contain custom headers which email programs can use for automatic filtering. Emails also contain headers to ensure they are correctly threaded in email clients which support doing so.

All Swarm emails contain the following headers, which can be used to identify which Swarm server they came from:

```
X-Swarm-Host: swarm.perforce.com
X-Swarm-Version: SWARM/2019.1/1798019 (2019/05/09)
```

The exact values may differ according to the version of Swarm you are running, and its configuration.

If a notification is applicable to one or more projects, then each project will be listed in the **X-Swarm-Project** header, which contains a list of one or more project names. Reviews may span multiple projects, so in this case a single email is sent out with each project listed.

```
X-Swarm-Project: gemini, apollo
X-Swarm-Host: swarm.perforce.com
X-Swarm-Version: SWARM/2019.1/1798019 (2019/05/09)
```

If one or more of the applicable projects is private, then two or more emails may be sent. In order for the existance of the private project to be hidden from non-members, any email sent to them will not contain references to the private project. Members of each private project will receive an email tailored to them which contains references to that private project. The email to the non-private projects will not contain references to any of the private projects in the **X-Swarm-Project** header.

For example, if a review spans three projects, called **Gemini**, **Apollo**, and **Ultra**, where **Ultra** is a private project, then members of projects **Gemini** and **Apollo** will receive an email with the following headers:

```
X-Swarm-Project: gemini, apollo
X-Swarm-Host: swarm.perforce.com
X-Swarm-Version: SWARM/2019.1/1798019 (2019/05/09)
```

Members of the **Ultra** project will receive an email with the following header:

```
X-Swarm-Project: ultra
X-Swarm-Host: swarm.perforce.com
X-Swarm-Version: SWARM/2019.1/1798019 (2019/05/09)
```

This can result in users receiving two notification emails (if they are members of **Ultra** and one of the other two projects), but privacy for the private project is preserved.

Email thread indexing

By default, Swarm indexes emails so that email conversations are threaded correctly in your email clients. If you find that your email clients are not displaying Swarm email threads correctly, disable thread indexing. To disable indexing, use the following configuration block and set to **false**:

```
<?php
    // this block should be a peer of 'p4'
    'mail' => array(
        'index-conversations' => false, // ['true'|'false'] default is
'true'
    ),
```

Emoji

By default, Swarm uses a font to provide Emoji images. Swarm can make use of Emoji images from the Gemoji project. Gemoji provides support for more Emojis and works on more browsers and platforms than the font Swarm normally uses.

> **Note**
> By default, Gemoji is only supplied with a small number of custom emojis. For instructions on adding more emojis to Gemoji, see the documentation for the Gemoji project.

Due to licensing issues, Gemoji cannot be distributed with Swarm. You can use the Gemoji images after following these steps:

1. Download the latest release (currently 3.0.0) of the Gemoji project from their releases page.
2. Unpack the release archive into *SWARM_ROOT*`/public/vendor`.

   After unpacking, you should see a new folder: ***SWARM_ROOT*`/public/vendor/gemoji-3.0.0`**.

   `3.0.0` represents the version of Gemoji, which may differ if you downloaded a different or newer release.

   > **Tip**
   > By default, Gemoji is stored in the ***SWARM_ROOT*`/public/vendor/gemoji`** directory.
   >
   > If the "emoji_path" on page 498 configurable has been changed: unpack the Gemoji release archive into the directory specified in the **`emoji_path`** configurable. Modify the commands in the following steps to match that directory.

3. Rename the new folder, default location shown in the example below:

   ```
   $ cd SWARM_ROOT/public/vendor
   $ mv gemoji-3.0.0 gemoji
   ```

   Replace *`3.0.0`* in the above command if you have downloaded a different or newer release of Gemoji.

4. Ensure that the new images are readable.

   ```
   $ cd SWARM_ROOT/public/vendor
   $ chmod -R +r gemoji
   ```

Swarm detects and uses Gemoji images automatically.

Environment

This section describes the *environment* configuration items available for Swarm:

- **mode**: whether Swarm operates in *production* or *development* mode.
- **hostname**: specifies the canonical hostname Swarm should use, such as in links to Swarm in email notifications.
- **external_url**: specifies the canonical URL Swarm should use, such as in links to Swarm in email notifications. Swarm can often auto-detect the correct URL, but use of **`external_url`** might be necessary in complex web hosting environments.
- **base_url**: specifies the folder name Swarm is installed within when Swarm is not installed in the web server's document root.
- **logout_url**: specifies the URL that users are redirected to when they logout of Swarm.
- **emoji_path**: specifies the location the Gemoji images are stored in.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Add the following configuration block to the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'environment' => array(
        'mode'         => 'development',      // defaults to 'production'
        'hostname'     => 'myswarm.hostname', // defaults to requested
hostname
        'external_url' => null,               // defaults to null
        'base_url'     => null,               // defaults to null
        'logout_url'   => null,               // defaults to null
        'vendor'       => array(
            'emoji_path' => 'vendor/gemoji/images',
        ),
    ),
```

mode

By default, Swarm operates in *production* mode. When `mode` is set to `development`, Swarm displays greater error detail in the browser. Also, Swarm switches from including aggregated and minified JavaScript and CSS to requesting each JavaScript and CSS resource for all active modules. The default value is production. Any value other than `development` is assumed to mean production.

`development` mode makes it easier to discover problems and to identify their source, but also incurs additional browser overhead due to many more JavaScript and CSS requests for larger files. We recommend that you do not use development mode in production environments, unless directed to do so by Perforce technical support.

hostname

The hostname item allows you to specify Swarm's hostname. This could be useful if you have multiple virtual hosts deployed for a single Swarm instance; Swarm uses the hostname you configure when generating its web pages and email notifications.

Note
The value specified for the hostname item should be just the hostname. It should not include a scheme (e.g. `"http://"`), nor should it include a port (e.g. `":80"`).

external_url

The `external_url` item allows you to specify Swarm's canonical URL. This is useful if your Swarm instance is proxied behind another web service, such as a load balancer, caching proxy, etc., because Swarm's auto-detection of the current hostname or port could otherwise result in incorrect self-referencing URLs.

When specified, Swarm uses the `external_url` item as the prefix for any URLs it creates that link to itself in its web pages and email notifications.

Note
Any path components included in external_url are ignored. If you specify `https://myswarm.url:8080/a/b/c`, Swarm only uses `https://myswarm.url:8080/` when composing URLs.

Important
Multiple Swarm instances connected to a single Helix server instance:

If you specify base_url along with `external_url`, ensure that all Swarm instances specify the same `base_url`. Varying `base_url` amongst cooperating Swarm instances is not supported.

base_url

The `base_url` item allows you to specify Swarm's folder within the web server's document root. This is useful if you cannot configure Swarm to operate within its own virtual host, such as when you have an existing web service and Swarm must exist alongside other applications or content. For information on configuring Swarm to run within a sub-folder of an existing website, see "Run Swarm in a sub-folder of an existing web site" on page 183.

Tip
The `swarm-trigger.conf` file must be updated to include the `base_url` in the `SWARM_HOST` path, see "Swarm trigger configuration" on page 186

By default, `base_url` is null, which is equivalent to specifying `/`. If you specify a folder, include the leading `/`. For example, `/swarm`.

Warning
Multiple-Helix server instances on a single Swarm instance:

Issue: Swarm will lose connection to all of the Helix servers if you edit the base_url configurable value in the **environment** block of **<swarm_root>/data/config.php**. This will stop your system working.

Fix: Remove the **base_url** configurable from the **environment** block of **<swarm_root>/data/config.php**.

Important
Multiple Swarm instances connected to a single Helix server instance:

If you specify external_url along with **base_url**, ensure that all Swarm instances specify the same **base_url**. Varying **base_url** amongst cooperating Swarm instances is not supported.

logout_url

The **logout_url** configurable is used to automatically redirect users to a custom URL after they have logged out of Swarm.

When the **logout_url** item is set and you log out of Swarm:

- Using **Log out** from Swarm: you are logged out by Swarm and then redirected to the custom URL.
- Using **Log out** from a single Swarm instance on the global dashboard: you are logged out of the Helix server instance by Swarm. You are only redirected to the custom URL if you are not logged in to any other Helix server instances on the global dashboard.
- Using **Log out from all instances** on the global dashboard: you are logged out of all of the Helix server instances by Swarm and then redirected to the custom URL.

To redirect users to a custom URL after they have logged out of Swarm, edit the *SWARM_ROOT*/**data/config.php** file and set the **logout_url** configurable to the URL you want to redirect users to:

```
<?php
    // this block should be a peer of 'p4'
    'environment' => array(
        'logout_url'   => <http[s]://your/redirect/logout/url>, //
defaults to null
    ),
```

The default value is **null**, users are not redirected when they log out of Swarm.

emoji_path

The **emoji_path** configurable defines the location of the Gemoji images in *SWARM_ROOT*/**public/**.

To change the location of the Gemoji images, edit the *SWARM_ROOT*`/data/config.php` file and set the `emoji_path` configurable to the location the Gemoji images are stored in:

```
<?php
    // this block should be a peer of 'p4'
    'environment' => array(
        'vendor'   => array(
            'emoji_path' => 'vendor/gemoji/images',
        ),
    ),
```

The default location is `vendor/gemoji/images`, this means that by default images are stored in *SWARM_ROOT*`/public/vendor/gemoji/images`.

Further information

For instructions on downloading Gemoji and configuring Swarm to use the images, see Emoji.

Excluding Users from Activity Streams

Larger Helix server installations often have one or more *service* users that perform automated tasks, such as build systems, continuous integration test servers, integrations with 3rd-party databases via P4DTG, or with Git via Perforce Git Fusion.

As Swarm reports the activity of users, and these service users can generate significant volumes of activity entries, Swarm provides a mechanism to ignore activity from specified users.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Update the *SWARM_ROOT*`/data/config.php` file to include the following configuration block:

```
<?php
    // this block should be a peer of 'p4'
    'activity' => array(
        'ignored_users' => array(
            'git-fusion-user',
            'p4dtguser',
            'system',
        ),
    ),
```

After ***SWARM_ROOT*/data/config.php** is updated, Swarm no longer records activity for any of the listed userids. Any previously recorded activity is included in activity streams.

Files configuration

Swarm can be customized using the following config items:

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

max_size

Swarm limits the size of files that are displayed in a review or standard file view. This defaults to 1MB, but can be configured to be larger or smaller. Files larger than this will be truncated.

If the value is set to zero, then the file size is unlimited.

```
<?php
    // this block should be a peer of 'p4'
    'files' => array(
        'max_size' => 1 * 1024 * 1024, // 1MB (in bytes)
    ),
```

download_timeout

When downloading files, there is a timeout which defaults to 30 minutes. This can be changed by setting the **download_timeout** configurable to a value in seconds. Alternatively, setting it to zero removes the timeout.

```
<?php
    // this block should be a peer of 'p4'
    'files' => array(
      'download_timeout' => 1800, // seconds (30 minutes)
    ),
```

allow_edits

By default, you can edit a file from the files page and shelve or commit it.

When set to false, files cannot be edited from the Swarm files page.

```
<?php
    // this block should be a peer of 'p4'
    'files' => array(
        'allow_edits' => false, // default is true
    ),
```

Groups configuration

By default, Swarm allows all users to see the **Groups** menu item and use it to view the list of user groups. Users can view the members of a group, as well as their activities.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

You configure the visibility of this menu item with the **`groups`** configuration block in the *SWARM_ROOT*`/data/config.php` file, as in the following example:

```
<?php
    // this block should be a peer of 'p4'
    'groups' => array(
        'super_only' => true, // ['true'|'false'] default value is 'false'
    ),
```

Setting the **`super_only`** option to **`true`** hides the groups tab from non-admin users.

> **Tip**
> If you need more fine grained control over the visibility of the **Groups** menu item, create a **`groups`** block in the `menu_helpers` configuration block. This enables you to restrict the visibility to only authenticated users, only users with admin privileges, or only users with super privileges. It is important to note that visibility of the **Groups** menu is controlled by a combination of the **`super_only`** setting and the **`role`** setting for groups in the **`menu_helper`** configuration block. Swarm uses the most restrictive of these settings to control the visibility of the **Groups** menu item.

Ignored users for reviews

Automated test environments may inadvertently participate in code reviews if they copy user-generated change descriptions. For example, if an automated system copied a change description containing #review and subsequently shelved or committed files, a new review would be started. Similarly copying a description with **`#review-123`** could inadvertently update an existing review. As test environments may involve thousands or millions of tests, such interactions can potentially generate far too many Swarm notifications.

To mitigate this problem, Swarm can be configured to ignore specified users for the purposes of starting or updating a review. Edit the *`SWARM_ROOT`***`/data/config.php`** file, and provide the list of users to ignore in the **`ignored_users`** item in the reviews configuration block.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Edit the *`SWARM_ROOT`***`/data/config.php`** file, and provide the list of users to ignore in the **`ignored_users`** item in the reviews configuration block. For example:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'ignored_users'  => array('build_user1', 'build_user2'),
    ),
```

License

Helix Swarm has always been free to use with an unlicensed Helix server. With the 2014.4 release, Swarm is free to use with any Helix server. You no longer need to purchase a Swarm-specific license, and any existing Swarm-specific licenses are no longer evaluated.

An unlicensed Helix server provides unlimited use for up to 5 users and 20 workspaces. When the user or workspace limit is crossed, Helix server imposes a maximum of 1,000 files. Swarm honors the restrictions of Helix server.

Localization & UTF8 character display

Swarm supports localization and the display of some non-UTF8 characters.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Localization

Swarm is fully localized; with an appropriate language pack installation, Swarm can support users in multiple languages.

A language pack consists of **`gettext-style default.po`** and **`default.mo`** files, placed in a folder named for the locale they represent, within the language folder in the Swarm root directory. In addition, language packs contain two JavaScript files to provide translation strings for the in-browser UI, **`locale.js`** and **`locale.jsgz`**, which both appear in the *SWARM_ROOT*`/public/build/language` folder.

The following example illustrates the directory layout of a language pack:

```
SWARM_ROOT/
    language/
        locale/
            default.mo
            default.po
    public/
        build/
            language/
                locale.js
                locale.jsgz
```

You can configure certain localization behaviors with the `translator` configuration block in the *SWARM_ROOT*`/data/config.php` file. Here is an example:

```
<?php
    // this block should be a peer of 'p4'
    'translator' => array(
        'detect_locale'              => true,
        'locale'                     => "",
        'translation_file_patterns'  => array(),
    ),
```

The `detect_locale` key determines whether Swarm attempts to detect the browser's locale. The default value is `true`. Set the value to `false` to disable browser locale detection.

The **locale** key is a string specifying the default locale for Swarm. Alternately, an array of 2 strings can be used to specify the default locale, and a fallback locale. For example:

```
<?php
    // this block should be a peer of 'p4'
    'translator' => array(
        'locale'                    => array("en_GB", "en_US"),
    ),
```

The **translation_file_patterns** key allows you to customize the Laminas translation infrastructure, which you might do if you are developing your own language pack. For details, see Laminas\I18n.

> **Note**
> Swarm supports the Laminas component versions in the **LICENSE.txt** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **LICENSE.txt** file is in the **readme** folder of your Swarm installation.

Non-UTF8 character display

Swarm supports the display of non-UTF8 characters in file content when all of the following are true:

- The Helix Core server is Unicode enabled.
- The character encoding is supported by Swarm.
- The character encoding is listed in the **translator** array.

Multiple character encodings can be specified in the array, if Swarm fails to find a matching encoding in the array it will fall back to **windows-1252**. Unsupported and invalid character encodings in the array are ignored.

> **Tip**
> For information on character encodings supported by Swarm, see http://php.net/manual/en/mbstring.supported-encodings.php

Configure character encodings by adding the following block to the *SWARM_ROOT*/**data/config.php** file. For example:

```
<?php
    // this block should be a peer of 'p4'
    'translator' => array(
        'non_utf8_encodings' => array('sjis', 'euc-jp', 'windows-1252'),
    ),
```

Important
windows-1252 is the default character encoding, it must always be the last entry in the array.

UTF8 conversion

If Swarm is connected to a non-unicode enabled Helix server, Swarm converts special characters to UTF8 by default. For ANSI and ASCII files, this can result in some special characters being displayed in the diff view incorrectly or not at all. To stop Swarm converting special characters to UTF8 and to display special characters as unknown ( ), disable UTF8 conversion by setting **utf8_convert** to **false**.

Disable UTF8 character conversion by adding the following block to the *SWARM_ROOT*/**data/config.php** file. For example:

```
<?php
    // this block should be a peer of 'p4'
    'translator' => array(
        'utf8_convert' => false,
    ),
```

By default, the value of this configurable is **true**.

Logging

Helix Swarm is a web application, so there are several types of logging involved during the course of Swarm's normal operations.

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Web server logging

All accesses to Swarm may be logged by the web server hosting Swarm. As web server log configuration is web server specific, refer to your web server's documentation. Since we recommend the use of Apache, more information regarding log configuration in Apache can be found here:
http://httpd.apache.org/docs/2.4/logs.html

Helix Core server logs

Swarm communicates with the associated Helix server for every page request. Review the Swarm-generated requests on your Helix server by enabling logging.

For more information, see Configuring logging and Auditing user file access in the *Helix Core Server Administrator Guide*.

Swarm logs

Depending on the log configuration you provide to Swarm, Swarm can log its own operations. Swarm's logs are much more helpful if you encounter problems.

Swarm stores its log data in the *SWARM_ROOT*`/data/log` file. The volume of entries recorded in the log depends on the configuration stored in the ***SWARM_ROOT***`/data/config.php` file. Here is an example:

```
<?php
    // this block should be a peer of 'p4'
    'log' => array(
        'priority'  => 3, // 7 for max, defaults to 3
),
```

The log `priority` specifies the importance of the messages to be logged. When `priority` is set to `0` (the lowest value), only the most important messages are logged. When `priority` is set to `7` (the highest value), all messages, including the least important messages, are logged. The default priority is `3`.

The different priority levels are:

Priority	Description
0	Emergency: a message about a system instability
1	Alert: a message about a required immediate action
2	Critical: a message about a critical condition
3	Error: a message about an error
4	Warning: a message about a warning condition
5	Notice: a message about a normal but significant condition
6	Info: an informational message
7	Debug: a debugging message

> **Note**
> Setting **priority** to a value lower than **0** does not result in reduced logging.

Reference ID

When **reference_id** is set to **true**, every log message is appended with **referenceId:<id>** where **<id>** is a hash value based on the web request id that generated the log message. The **<id>** is appended to all of the log messages related to that web request for the life of the web request allowing you to follow the request in the log. This can help to diagnose which request was being processed when the log message was created. It is particularly useful when you have lots of requests updating the log because it helps you to see which request each log message relates to.

Example: when a Swarm web request starts a worker the **<id>** that is generated is attached to all of the log entries related to that worker for the lifetime of the worker. This enables you to see all of the events the worker processes in its lifetime. For this example the web request **reference Id** is **"066ae44103ced2ab05c28578a36b6854"**:

```
2019-02-27T09:07:21+00:00 INFO (6): Worker 1 startup.
{"referenceId":"066ae44103ced2ab05c28578a36b6854"}
2019-02-27T09:07:21+00:00 INFO (6): Worker 1 event: task.commit(3276)
{"referenceId":"066ae44103ced2ab05c28578a36b6854"}
2019-02-27T09:07:21+00:00 DEBUG (7): P4 (00000000557948260000000002e2ec483)
start command: help  {"referenceId":"066ae44103ced2ab05c28578a36b6854"}
 .
 .
 .
2019-02-27T09:17:16+00:00 INFO (6): Worker 1 shutdown.
{"referenceId":"066ae44103ced2ab05c28578a36b6854"}
```

To append the **referenceId:<id>** to Swarm log messages, edit the *SWARM_ROOT*/**data/config.php** file to include the **reference_id** configurable and set it to **true**:

```
<?php
    // this block should be a peer of 'p4'
    'log' => array(
        'reference_id'  => true, // defaults to false
),
```

The default value is **false**.

Trigger Token Errors

If the trigger tokens are missing or invalid, the web server error log contains a suitable error:

```
queue/add attempted with invalid/missing token: token used
```

A token is *invalid* when it is not formed from the characters A through Z (in upper or lowercase), 0 through 9, or a dash (-).

A token is *missing* when a file using the token as its name does not exist in the *SWARM_ROOT*`/data/queue/tokens` directory.

Performance logging

Swarm logs warnings whenever commands issued to the Helix server take longer than expected to complete. These warnings can be used to diagnose performance problems in the Helix server.

> **Note**
> By default, warnings are not captured in the Swarm log. To capture warnings, the log priority must be set to 4 or higher.

The default configuration is:

```
<?php
    // this block should be placed within the 'p4' block
    'slow_command_logging' => array(
        3, // commands without a specific rule have a 3-second limit
        10 => array('print', 'shelve', 'submit', 'sync', 'unshelve'),
    ),
```

In this configuration block, the numeric key specifies the time threshold in seconds, and the value (if present) is an array of Helix server commands that should use the threshold. Should a command be associated with multiple thresholds, the largest is used for that command.

In addition, Swarm automatically detects when the PHP extension **xhprof** is installed and collects profiling data for requests that take longer than expected. The profiling data could be helpful in diagnosing performance issues within Swarm itself, particularly when analyzed in combination with the slow command logging described above. When collected, profiling data is stored in the *SWARM_ROOT*`/data/xhprof` folder.

The default configuration is:

```
<?php
    // this block should be a peer of 'p4'
    'xhprof' => array(
        'slow_time'      => 3, // the threshold in seconds
        'ignored_routes' => array('download', 'imagick', 'libreoffice',
'worker'),
    ),
```

slow_time specifies the threshold, in seconds, that should be used to determine when a Swarm request is slow. **ignored_routes** is an array that specifies a list of Laminas Framework **route** names that should not be profiled. For example, Swarm's **Files** module specifies that the download route should be ignored from profiling as downloads could take significantly longer than the default threshold.

> **Note**
> Depending on the performance of the server hosting Swarm, and particularly the performance of the associated Helix server, you may want to monitor the *SWARM_ROOT*/**data/xhprof** folder for disk usage. Each request that exceeds the time threshold causes 200-600KB of data to be written.

Mainline branch identification

When viewing a project's files, the initial view is the list of the project's branches. The branches appear in alphabetical order, but the branch identified as the main branch appears first and in bold. By default, a project overview page is displayed when there is a README Markdown file in the project mainline, this can be disabled by your Swarm administrator, see "Project readme" on page 538.

Swarm uses a list of names to identify which of a project branches should be considered as the main branch and which Markdown file is used for the project overview page. The **mainlines** array supports regular expressions, see "Regular expressions" on the next page. The default names are *main*, *mainline*, *master*, and *trunk*. For the steps Swarm goes through to find the mainline branch and the Markdown file for a project, see "Project mainline and README check" on page 511 below.

> **Tip**
> The default *main*, *mainline*, *master*, and *trunk* names are hardcoded in Swarm, there is no need to add them to your array. Swarm will always try to find exact matches to them from your branch names.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

You can adjust the configuration to match your local branch naming convention . Here is an example of the configuration block:

```
<?php
    // this block should be a peer of 'p4'
    'projects' => array(
        'mainlines' => array(
            'stable', 'release', // 'main', 'mainline', 'master', and
'trunk' are hardcoded, there is no need to add them to the array
```

```
        ),
    ),
```

Regular expressions

The `mainlines` array supports any valid PHP regular expression, this can be useful if you have a large number of similar branch names across different projects that you are using as your main branches. Regular expressions must be used with care to avoid accidentally including a branch that is not a mainline branch.

> **Note**
> Swarm adds a `^` character before your regular expression and a `$` character after it. For example: `^your-reg-ex-pattern$` You must take this into account when constructing your regular expressions.

> **Tip**
> - You can test the results of your regular expressions before using them, see https://regex101.com/
> - Regular expressions are case insensitive.

Regular expression examples

Useful regular expressions are shown below:

Exact match only:

`'mymainbranch'` branches named `mymainbranch` are treated as mainline branches.

Only match default `mainlines` array branch names:

`''` (an empty `mainlines` array) only branches that exactly match the default names of `'main'`, `'mainline'`, `'master'`, or `'trunk'` are treated as mainline branches.

Beginning with:

`'stable.*'` branch names beginning with `stable` are treated as mainline branches.

For example: the following are treated as mainline branches.

stablecode-01, *stable2019_2*, and *stable_branchxyz*

Ending with:

`'.*mastprj'` branch names ending with `mastprj` are treated as mainline branches.

For example: the following are treated as mainline branches.

productxy-mastprj, *assets_mastprj*, and *project07mastprj*

Ending with digits

`'.*[0-9]+'` branch names ending with digits are treated as mainline branches.

For example: the following are treated as mainline branches.

branch03, *product-10*, *stable22*, *main_123456*

Containing:

`'.*main.*'` branch names that contain **`main`** are treated as mainline branches.

For example: the following are treated as mainline branches.

swarm-main-branch, *branchmain_22*, *main05*, and *assetmain-brnch*

Project mainline and README check

> **Tip**
> Swarm checks the branches of a project in alphabetical order.

Swarm uses the following process to find the mainline branch and README Markdown file for a project:

1. Swarm searches the project branches in alphabetical order for a branch that matches a name in the **`mainlines`** array. The search includes the default names.
2. When Swarm finds a matching branch, that branch is used as the mainline when displaying the branches on the project page.
    - If the branch contains a README Markdown file, that file is used for the project overview page.
    - If the branch does not contain a README Markdown file, see step 3.
3. Swarm checks the remaining branches for a branch that matches a name in the **`mainlines`** array.
    - If Swarm finds a README file, that file is used for the project overview page and swarm stops checking the branches.
    - If a README file is not found, this step is repeated until a README file is found in a in a branch that matches a name in the **`mainlines`** array or all of the project branches have been checked.
    - If no README Markdown page is found, the project overview page is not displayed.

Markdown

The **markdown** configurable defines what can be rendered in Markdown by Swarm for project overview pages and files stored in the Helix Core server. By default, **markdown** is set to **safe**, Markdown text is displayed, but Markdown support is limited to prevent execution of raw HTML and JavaScript content.

Tip

- Valid Markdown file extensions are: **md**, **markdown**, **mdown**, **mkdn**, **mkd**, **mdwn**, **mdtxt**, **mdtext**.
- By default, project overview pages are displayed when there is a **README** Markdown file in the project mainline. This can be disabled by your Swarm administrator, see "Project readme" on page 538.

Tip

If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Add or update the following configuration block to the *SWARM_ROOT*/**data/config.php** file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'markdown' => array(
        'markdown' => 'safe', // default is 'safe'
    ),
```

- **safe**: Markdown content is displayed, but Markdown support is limited to prevent execution of raw HTML and JavaScript content. This is the default.
- **unsafe**: Markdown support is unrestricted, allowing full HTML and JavaScript to be used. This is insecure as any person with access to Swarm can add script to the Markdown which would execute as the currently logged in user.
- **disabled**: Markdown text is not rendered and is only displayed as plain text. This is the most secure setting.

Note

Markdown content is displayed in comments, but Markdown support is limited to prevent execution of raw HTML and JavaScript content. This is the equivalent of **safe** mode and cannot be changed.

Menu helpers

The **menu_helpers** configuration block is used to control the visibility of Swarm menu items and to create new menu items for URLs and Swarm modules.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Add a menu item to a menu

Add Swarm menu items in the **menu_helpers** configuration block in the *SWARM_ROOT*/**data/config.php** file, as in the following example:

```
<?php
    // this block should be a peer of 'p4'
    'menu_helpers' => array(
        'MyMenu01' => array( // A short recognizable name for the menu
item
            'id'        => 'custom01',            // A unique id for the
menu item. If not included in the array, parent array name is used.
            'enabled'   => true,                  // ['true'|'false']
'true' makes the menu item visible. 'true' is the default if not included
in the array.
            'target'    => '<https://MyWebite>',  // The URL or custom
module route a menu click takes you to.
                                                  // If not included in
array, id is used. If id not included, parent array name is used.
            'cssClass'  => 'custom_menu',         // The custom CSS class
name added to the menu item, appended to h2.menu- in Swarm CSS
            'title'     => 'MyMenuItem',          // The text that will be
shown on the button.
                                                  // If not included in
array, id is used. If id not included, parent array name is used.
            'class'     => '',                    // If not included in
array or empty, the menu item is added to the main menu.
                                                  // To add the menu item
```

```
to the project menu for all of the projects, set to
'\Projects\Menu\Helper\ProjectContextMenuHelper'
            'priority'  => 155,                  // The position the menu
item is displayed at in the menu.
                                                 // If not included in
the array, the menu item is placed at the bottom of the menu.
            'roles'     => null,                 //
['null'|'authenticated'|'admin'|'super'] If not included in the array,
null is the default
                                                 // Specifies the minimum
level of Perforce user that can see the menu item.
                                                 // 'authenticated' = any
authorized user, 'null' = unauthenticated users
        ),
    ),
```

Configurables

> **Tip**
> You do not need to enter values for all of the configurables in a menu item array, configurables not included in the array will be set to their default values. For an example of this, see "Create a menu item using only the custom module name" on page 516.

- **`id`** a unique id used by Swarm for the menu item, this can be anything. If **`id`** is not included in the array, Swarm uses the name of the parent array.
- **`enabled`**
  - **`true`** the menu item is visible subject to restrictions imposed by the **`roles`** value. This is the default value if **`enabled`** is not included in the array.
  - **`false`** the menu item is not in use
- **`target`** the URL or Swarm module route used when the menu item is clicked. If **`target`** is not included in the array, Swarm uses the **`id`** for the route. If **`id`** is not included in the array, Swarm uses the parent array name for the route.
- **`cssClass`** specifies the CSS class for the menu item, used to add a custom icon to the menu item, see "Add a custom icon to a menu item" on the facing page. The **`cssClass`** value is appended to **`h2.menu-`** to create the CSS class for the menu item. In the example above, it create a CSS class of **`h2.menu-custom_menu`**. By default, if a **`cssClass`** is not specified Swarm adds a chevron **>** icon to the menu item.

- **`title`** specifies the menu item name displayed in the menu. If **`title`** is not included in the array, Swarm uses the **`id`** for the menu title. If **`id`** is not included in the array, Swarm uses the parent array name for the menu title.
- **`class`** specifies whether the menu item is displayed in the main menu or the project menu.
  - **`''`** or not included in the array, adds the menu item to the main menu.
  - **`\Projects\Menu\Helper\ProjectContextMenuHelper`** adds the menu item to the project menu of all of your projects.

  > **Tip**
  > You can restrict the menu item to a specific project by extending the **`ProjectAwareMenuHelper`** module, see "Add a menu item to the project menu of a specific project" on page 517.

- **`priority`** specifies the menu item position in the menu, lower numbers are displayed higher up the menu structure. If priority is not included in the array, the menu item is placed at the bottom of the Swarm menu. If you use the same priority value as an existing menu item (not recommended), the menu items with the same priority are displayed in alphabetical order.

  > **Tip**
  > Enter **`<yourSwarmURL>/api/v10/menus`** in your browser address bar to return a list of the priority values for the Swarm menu items.

- **`roles`** restrict the display of the menu item based on the permissions of the logged in user:
  - **`null`** menu item visible to any user even if they are not authenticated. if roles is empty or not included in the array, null is used as the default.
  - **`authenticated`** menu item is visible to any authenticated user
  - **`admin`** menu item is visible to any user with Perforce *admin* permissions or higher
  - **`super`** menu item is visible to any user with Perforce *super user* permissions

Add a custom icon to a menu item

The **`cssClass`** specified for a menu item is appended to **`h2.menu-`** in the Swarm CSS.

This enables you to add some custom CSS to Swarm to change the default icon for the menu item. For example, the CSS below adds the **`double-speech-bubbles.svg`** icon to the menu item with the **`cssClass`** of **`custom-menu`**:

```
.swarm-menu-item.menu-custom-menu .svgIcon{
    display:none;
}
.swarm-menu-item.menu-googleMain a::before{
    content: '';
```

```
    background-image: url('/swarm/img/icons/line/double-speech-
bubbles.svg');
    background-size: 16px;
    background-repeat: no-repeat;
    padding-left: 18px;
}
```

Make Groups menu item visible to admin-user and above only

> **Note**
> The visibility of the Swarm **Groups** menu is controlled by a combination of the **`role`** setting here and the **`super_only`** setting in the **`groups`** configuration block, see "Groups configuration" on page 501. Swarm uses the most restrictive of these two settings to control the visibility of the **Groups** menu item.

If you need fine grained control over the visibility of the **Groups** menu item, create a **`groups`** block in the **`menu_helpers`** configuration block and set the **`roles`** as required. In this example **`roles`** is set to **`'admin'`** so that only users with *admin-user* access and higher will be able to see the **Groups** menu item:

```
<?php
    // this block should be a peer of 'p4'
    'menu_helpers' => array(
        'groups' => array(
            'roles'     => 'admin',
        ),
    ),
```

> **Tip**
> Enter **`<yourSwarmURL>/api/v10/menus`** in your browser address bar to return a list of **`roles`** for the Swarm menu items.

Create a menu item using only the custom module name

If you have created a Swarm custom module you can create a menu item for it that uses the custom module route and you do not need to add any other configurables for it in the **`menu_helpers`** configuration block.

For example, if you have a custom module with a route of **`'codeAnalytics'`**, the configuration block only needs to be:

```
<?php
    // this block should be a peer of 'p4'
    'menu_helpers' => array(
        'codeAnalytics' // links the menu item to a module called
'codeAnalytics'
    );
```

All of the other configurable settings for the **codeAnalysis** menu item will be set to their default values because the **codeAnalysis** array is empty:

- **id**: parent array name used (**codeAnalysis**)
- **enabled = true**, menu item is enabled
- **target**: parent array name used (**codeAnalysis**), this must match the custom module name
- **cssClass**: no class is set and Swarm uses the default chevron **>** icon for the menu item
- **class**: no class is set, the menu item is displayed in the main menu
- **priority**: no priority is set, the menu item is displayed at the bottom of the Swarm menu
- **roles**: **null**, visible to all users even if they are not authenticated

Add a menu item to the project menu of a specific project

To add a menu item in the project menu of a specific project, extend the **ProjectAwareMenuHelper** and set the menu item **class** to use the extended helper.

The example in this section only displays the **Jenkins Build** menu item in the **Jam** project menu.

Extend the ProjectAwareMenuHelper

Extend the **ProjectAwareMenuHelper** so that the **Jenkins Build** menu item is only displayed in the **Jam** project menu. This helper can be called anything and saved anywhere but it should have a descriptive name and be in a logical place, in this example it is **/modules/Project/src/Menu/Helper/JamOnlyMenuItem.php**:

```
<?php
  /**

  * Perforce Swarm
  * @copyright 2020 Perforce Software. All rights reserved.
  * @license Please see LICENSE.txt in top-level folder of this
distribution.
  *@version <release>/<patch>
```

```
  */

namespace Projects\Menu\Helper;

  /**
  * Only show the menu item using this class for the specified Project.
  * In this example it is for the project with the id of Jam.
  * Class JamOnlyMenuItem
  * @package Projects\Menu\Helper
  */
class JamOnlyMenuItem extends ProjectAwareMenuHelper
{
    /**
    * Modifies an item's target if the context is for a project, the
project supports the item and
    * the item already has a target. Otherwise, nullify the item
    * @return array|null
    */
  public function getMenu()
  {
    if ($this->project && $this->project->getId() === 'jam') {
      $item = parent::buildMenu();
      // Allow project characteristics to determine menu item availability
      return !empty($this->project) && $this->isDisabled() === false ?
$item : null;
    }
    return null;
  }
}
```

Define your menu item

The following block defines the **Jenkins Build** menu item you are adding to the **Jam** project menu. The important part is that **`class`** is set to **`\Projects\Menu\Helper\JamOnlyMenuItem`** so that it is only displayed in the **Jam** project.

```
<?php
    // this block should be a peer of 'p4'
    'menu_helpers' => array(
```

```
        'jenkins_jam' => array( // A short recognizable name for the menu
item
            'id'        => 'jenkins_jam',
            'enabled'   => true,
            'target'    => 'http://my-jenkins.instance.jam.com',
            'cssClass'  => 'custom_menu',
            'title'     => 'Jenkins Build',
            'class'     => '\Projects\Menu\Helper\JamOnlyMenuItem',
            'priority'  => 160,
            'roles'     => ['authenticated'],
        ),
    ),
```

Delete Swarm config cache to enable the menu item

Your changes will not be used by Swarm until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Multiple-Helix server instances

A single Swarm server can be connected one or more Helix Core server (P4D) instances. This section describes how to configure Swarm to connect to multiple Helix servers.

Warning
Issue: Swarm will lose connection to all of the Helix servers if you edit the base_url configurable value in the **`environment`** block of **`<swarm_root>/data/config.php`**. This will stop your system working.

Fix: Remove the **`base_url`** configurable from the **`environment`** block of **`<swarm_root>/data/config.php`**.

Note
Known limitations, only applies if you want to use " Global Dashboard" on page 232**:**

- **Issue:** If Helix Authentication Service is enabled for Swarm and the **Try to login to all available servers with these credentials** checkbox or the **All available servers** option is selected in a login dialog, Swarm will not try to log in to any of the other Helix server instances that are configured for Helix Authentication Service.

Workaround: Log in to them individually using the instance **Log in** button  in the sidebar or by including the server instance name in the URL, for example: **`https://swarm.company.com/serverA`**.

- Swarm must be installed in its own virtual host.

Complete the following steps to connect Swarm to multiple-Helix server instances:

- "Set up the Swarm configuration file for the Helix servers" below
- "Set the Swarm trigger token and Swarm host variable for each Helix server" on page 522
- "Configure a cron job for each Helix server instance" on page 523

Set up the Swarm configuration file for the Helix servers

The Swarm Helix server connection details are configured using the **`p4`** item in the **`<swarm_root>/data/config.php`** configuration file.

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

When Swarm is configured for a single Helix server instance the **`p4`** item looks like this:

```
<?php
    return array(
        'p4' => array(
            'port'      => 'my-helix-core-server:1666',
            'user'      => 'admin_userid',
            'password'  => 'admin user ticket or password',
            'sso_enabled' => false, // defaults to false
         ),
    );
```

Where:

- **`port`** is the P4PORT for your Helix server.
- **`user`** is a *userid* that has *admin* privileges for the Helix server.
- **`password`** is the ticket or password for the Helix server user.
- **`sso_enabled`** set to true if the Helix server is configured for Helix Authentication Service.

Multiple Helix server instances can be added to the **p4** item in **config.php** but each Helix server instance must have a label to identify the server instance. This enables Swarm to connect to the Helix servers in the **p4** item.

> **Tip**
> The Helix server instances do not have to all be at the same version but they must all be versions that Swarm supports. For a list of Helix server versions supported by Swarm, see Helix Core server requirements.

The first Helix server in the **p4** item is the primary Helix server, this instance renders the global dashboard. If Swarm fails to connect to the primary Helix server, Swarm cannot display any of the Helix server instances in the global dashboard. You can still connect directly to the other P4D instances by including the Helix server label in the URL for each instance, for example: **https://swarm.company.com/serverA**.

Configure the Swarm configuration file for multiple Helix server instances:

Enter a label for each Helix server instance and enter the server connection details under the server label.

For example:

```
<?php
    return array(
        'p4' => array(
            'serverA' => array(
                'port'      => 'p4d-A:1666',
                'user'      => 'admin_userid_serverA',
                'password'  => 'admin user ticket or password for
serverA',
                'sso_enabled' => false, // defaults to false
            ),
            'serverB' => array(
                'port'      => 'p4d-B:1666',
                'user'      => 'admin_userid_serverB',
                'password'  => 'admin user ticket or password for
serverB',
                'sso_enabled' => true, // defaults to false
            ),
        );
```

Where:

- *serverA* and *serverB* are labels that identify the individual Helix servers.
  - The labels must be URL-friendly labels.
  - The labels must not contain any spaces or any characters that require URL encoding, and
  - The total length of the server label and Redis **`namespace`** combined is limited to ≤ 127 characters. The default **`namespace`** is **`Swarm`**.
- The **`port`**, **`user`**, **`password`**, and **`sso_enabled`** items are specific to the Helix server instance they are nested under.

Set the Swarm trigger token and Swarm host variable for each Helix server

Helix server uses a Swarm trigger token to confirm that trigger requests from Swarm are valid. Each Helix server instance must have a valid Swarm trigger token in its **`swarm-trigger.conf`** file. The **`swarm-trigger.conf`** file also contains the Swarm host URL for the Helix server instance.

Set the Swarm trigger token and Swarm host variable for each Helix server instance:

1. Navigate to the Swarm URL for the instance.

   For example: **`https://swarm.company.com/serverA`**
2. Log in to Swarm as a *super* user.
3. Click your *userid*, in the main toolbar and select **About Swarm**.

   The **About Swarm** dialog is displayed with a **Trigger Token**. Swarm generates the trigger token if it doesn't already exist.
4. Copy the trigger token value from the dialog.
5. Open the **`swarm-trigger.conf`** file for the Helix server instance.

   > **Tip**
   > For more information about the location of the **`swarm-trigger.conf`** file, see "Set up Swarm triggers with a Windows or Linux-hosted Helix server" on page 159.
6. In the **`swarm-trigger.conf`** file:
   a. Set the **`SWARM_HOST`** URL.
   b. Set the **`SWARM_TOKEN`** by pasting the Swarm trigger token you copied in the steps above.

   For example:

```
# SWARM_HOST (required)
# Hostname of your Swarm instance, with leading "http://" or
"https://".
SWARM_HOST="https://swarm.company.com/serverA"

# SWARM_TOKEN (required)
# The token used when talking to Swarm to offer some security. To
obtain the
# value, log in to Swarm as a super user and select 'About Swarm' to
see the
# token value.
SWARM_TOKEN="TRIGGER-TOKEN-FOR-SERVERA"
```

7. Save the **swarm-trigger.conf** file.
8. Repeat these steps for each Helix server.

> **Tip**
> Alternatively, you can simply touch a file in each **<Swarm root>/data/servers/<serverid>/tokens** folder. The file content is ignored, the filename itself is the token.
>
> This allows you to specify a common token that is used by all of the Helix server instances.
>
> However, we recommend that you use a separate token for each Helix server instance. This makes it easier to invalidate a token for a specific Helix server by deleting the file in the **tokens** folder for that server if you need to.

Configure a cron job for each Helix server instance

To automatically spawn workers for each of the servers connected to Swarm, you must manually configure a cron job for each of the servers.

Swarm package and OVA installations only

The **swarm-cron-hosts.conf** file specifies the connection type (HTTP or HTTPS), hostname, port number and, server label for Swarm cron jobs.

If you have installed Swarm via packages or you are running the Swarm OVA, you must specify all of the Helix server URLs within **/opt/perforce/etc/swarm-cron-hosts.conf**.

1. Edit the **swarm-cron-hosts.conf** file so that it contains the actual Swarm hostname, ports, and server labels you have configured for Swarm.

   The following format is used with one Helix server on each line:

```
[http[s]://]<swarm-host>[:<port>][/<base-url>]
```

Default if value not specified:

- **[http[s]://]** http
- **<swarm-host>** must be specified
- **[:<port>]** 80
- **[/<base-url>]** server label, must be specified

For example, for **serverA** and **serverB** configured earlier on a Swarm host of *https://swarm.company.com* with a default port value of 80. The entries in the **swarm-cron-hosts.conf** file would be:

```
https://swarm.company.com/serverA
https://swarm.company.com/serverB
```

2. Save the **swarm-cron-hosts.conf** file.
3. Swarm is now configured to connect to multiple-Helix server instances.

> **Tip**
> To check or modify Swarm worker configuration, see "Worker configuration" on page 603.

Swarm Tarball installation only

If you have installed Swarm from a tarball or configured cron manually, you need create a file called **helix-swarm** and add all of the Helix server instances connected to Swarm.

1. Create a file named **helix-swarm** in **/etc/cron.d** if it does not already exist.
2. Edit the **helix-swarm** file so that it has the following content:

```
#
# Cron job to start Swarm workers every minute
#
* * * * * nobody [ -x /opt/perforce/swarm/p4-bin/scripts/swarm-
cron.sh ] && /opt/perforce/swarm/p4-bin/scripts/swarm-cron.sh
```

3. Save the **helix-swarm** file.
4. Edit the **swarm-cron-hosts.conf** file so that it contains the actual Swarm hostname, ports, and server labels you have configured for Swarm.

   The following format is used with one Helix server on each line:

   ```
   [http[s]://]<swarm-host>[:<port>][/<base-url>]
   ```

   Default if value not specified:

- **[http[s]://]** http
- **<swarm-host>** must be specified
- **:<port>** 80
- **[/<base-url>]** server label, must be specified

For example, for **serverA** and **serverB** configured earlier on a Swarm host of *https://swarm.company.com* with a default port value of 80. The entries in the **swarm-cron-hosts.conf** file would be:

```
https://swarm.company.com/serverA
https://swarm.company.com/serverB
```

5. Save the **swarm-cron-hosts.conf** file.
6. Swarm is now configured to connect to multiple-Helix server instances.

> **Tip**
> To check or modify Swarm worker configuration, see "Worker configuration" on page 603.

Further information

Users

- To visit the Global Dashboard, enter the basic Swarm URL without a Helix server instance name, for example: **https://swarm.company.com**.
- To visit a specific Helix server instance in Swarm without going via the global dashboard, include the server name in the URL, for example: **https://swarm.company.com/serverA**.

Once you are viewing the Helix server in Swarm, Swarm works as a standard single Helix serverSwarm system.

> **Tip**
> - To browse jobs on **serverA**, navigate to: **https://swarm.company.com/serverA/jobs**
> - To browse reviews on **serverB**, navigate to: **https://swarm.company.com/serverB/reviews**
> - To view the dashboard for **serverB**, navigate to **https://swarm.company.com/serverB/#actionable-reviews**
> - If you don't include the server label in the URL you will be taken to the specified page for the primary Helix server.

> For example: navigating to **`https://swarm.company.com/reviews`** will redirect you to **`https://swarm.company.com/serverA/reviews`** because **serverA** is the Primary Helix server in the **`p4`** configuration item.

Administrators

On the web server hosting Swarm, Swarm automatically creates a data folder for each Helix server instance in **`<Swarm_root>/data/servers`**. This is because each Helix server request is a field.

For example:

```
# ls -la /opt/perforce/swarm/data/servers/serverA
total 362296
drwx------ 6 apache apache    4096 Sep  8 17:15 .
drwx------ 6 apache apache    4096 Sep  6 15:41 ..
drwx------ 2 apache apache    4096 Sep 20 18:08 cache
drwx------ 3 apache apache    4096 Sep  7 13:25 clients
-rw-r--r-- 1 apache apache 3709543 Sep 26 14:19 log
-r-------- 1 apache apache      84 Sep  6 15:41 p4trust
drwx------ 4 apache apache    4096 Sep 20 14:27 queue
drwx------ 2 apache apache    4096 Sep 20 11:18 sessions
```

> **Tip**
> It is important to understand that there will be a Swarm log file for each Helix server instance.

Developers

To get a list of projects via the Swarm API for *serverA*, run bash:

- If you are using **`wget`**:

  ```
  $ wget -u "apiuser:password"
  https://swarm.company.com/serverA/api/v10/projects
  ```

- If you are using **`curl`**:

  ```
  $ curl -u "apiuser:password"
  https://swarm.company.com/serverA/api/v10/projects
  ```

Tip
If you don't include the server label in the URL you will be taken to the projects page for the primary Helix server.

For example: navigating to **`https://swarm.company.com/api/v10/projects`** will redirect you to **`https://swarm.company.com/api/v10/serverA/projects`** because **serverA** is the Primary Helix server in the **`p4`** configuration item.

Notifications

This section describes the configurables that can be set for notifications.

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Swarm can be configured to provide generic notifications of committed changes in Helix server, taking the role of a review daemon.

Notifications configuration is expressed with a **`notifications`** block in the *`SWARM_ROOT`***`/data/config.php`** file, similar to the following example:

```
<?php
    // this block should be a peer of 'p4'
    'notifications' => array(
        'honor_p4_reviews'      => false,                  // defaults to
false
        'opt_in_review_path'    => '//depot/swarmReviews', // required if
honor_p4_reviews is true; defaults to ''
        'disable_change_emails' => false,                  // optional;
defaults to false
    ),
```

If **`honor_p4_reviews`** is set to **`true`**, then **`opt_in_review_path`** must be set to a path somewhere in the depot. This path does not need to point to an actual file that exists, but it must be accessible by all users who want to make use of this functionality. For example:

```
'notifications' => array(
        'honor_p4_reviews'   => true,
        'opt_in_review_path' => '//depot/swarmReviews',
),
```

If these two values are set, then users can make use of the Perforce review functionality by subscribing to the **opt_in_review_path** in their user spec. Any user subscribed to that file, will receive notifications for all the other paths they are subscribed to.

We recommend that **opt_in_review_path** point to a file that does not exist. Ideally, it points to a file that no user is likely to want to create. It must however be in a valid depot.

For example, if a user has the following review paths set in their user spec:

```
$ p4 user -o asmith
User:   asmith

Email:  asmith@example.com

FullName:       Alice Smith

Reviews:
        //depot/swarmReviews
        //depot/main/acme/...
        //depot/main/orion/...
        //depot/dev/asmith/...
```

The **//depot/swarmReviews** means that this user is subscribed to the path set in **opt_in_review_path**, and therefore will receive notifications. The rest of the subscription lines define which paths in the depot this user is interested in. Therefore this user will receive a notification for a change made to **//depot/main/acme/foo.txt**, but not a change made to **//depot/dev/acme/foo.txt**.

For example, to see which users are subscribed to receive notifications you can run **p4 reviews <path>**against the **opt_in_review_path** value:

```
$ p4 reviews //depot/swarmReviews
asmith <asmith@example.com> (Alice Smith)
bbrown <bbrown@example.com> (Bob Brown)
erogers <erogers@example.com> (Eve Rogers)
```

To see which users are subscribed to files in a particular changelist, you can run **p4 reviews -c <changelist>**. Swarm will notify the users who subscribe to both the review of this changelist and the review path, **opt_in_review_path**.

- **honor_p4_reviews**: When set to **true**, Swarm sends notification emails for every committed change to all users where the change matches one of their **Reviews:** paths.

- **opt_in_review_path**: Optional item, required only if `honor_p4_reviews` is set. This item specifies a special depot path, which typically would not exist in the Helix server machine. When a path is specified, users must include this path (or a path that contains it) in the `Reviews:` field of their user spec to cause Swarm to send the user a notification for every committed change that matches one of their `Reviews:` paths.

  For example, if the `opt_in_review_path` is set `to //depot/swarmReviews`, users can opt-in to Swarm review notifications by adding that path, or a path such as `//depot/...`, to the `Reviews:` field in their user spec.
- **disable_change_emails**: Optional item. When set to `true`, notifications for committed changes, based on the `Reviews:` field and the users and projects you follow, are disabled. Notifications for reviews and comments will still be sent.

Note
If your Helix server machine already has a review daemon in operation, users receive two notifications for `Reviews:` paths. You may want to deprecate the review daemon in favor of Swarm's change notifications.

Note
"Groups" on page 291 have per-group notification settings. See "Add a group" on page 393 for details.

Global settings

There are many situations that can result in email notifications being sent out to users and groups. Whilst it is possible for a user and group owner to configure their own settings, it is also possible for the system owner to configure the defaults for all users and groups by modifying the settings in the `config.php`.

Note
- **If a group owner is not specified for the group:** only users with *super* privileges can configure group notification settings.
- **If one or more group owners are specified for the group:** only group owners and users with *super* privileges can configure group notification settings.

Each notification consists of an `Event` and a `Role`. The `Event` is what happened (for example, a new review was created, a file was submitted) and the `Role` is the role of the user or group who could receive a notification. A user or group can belong to multiple roles, in which case if any of them are set to send a notification, then the user or group will receive a notification.

For example, when a review is voted on (`review_vote`) , there are a number of roles of users, and groups that could be notified:

- **Users:** the user that voted on the review (`is_self`), the author of the review (`is_author`), a user who is a moderator of the project branch the review is in (`is_moderator`), and a user who

is a reviewer of the review (`is_reviewer`).

- **Groups:** members of a moderator group for the project branch the review is in (`is_moderator`), and members of a reviewer group for the review (`is_reviewer`).

Configuration options

By default, all notifications are enabled for all roles. The system-wide defaults can be changed by adding the following options into the notifications block of the *SWARM_ROOT*`/data/config.php`. These options are in addition to those described above.

```
<?php
    // this block should be a peer of 'p4'
    'notifications' => array(
        'review_new' => array(
            'is_author' => 'Enabled',
            'is_member' => 'Enabled',
        ),
        'review_files' => array(
            'is_self'      => 'Enabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_moderator' => 'Enabled',
        ),
        'review_vote' => array(
            'is_self'      => 'Enabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_moderator' => 'Enabled',
      ),
      'review_required_vote' => array(
            'is_self'      => 'Enabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_moderator' => 'Enabled',
      ),
      'review_optional_vote' => array(
            'is_self'      => 'Enabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
```

```
            'is_moderator' => 'Enabled',
        ),
        'review_state' => array(
            'is_self'      => 'Disabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_moderator' => 'Enabled',
        ),
        'review_tests' => array(
             'is_author'    => 'Enabled',
             'is_reviewer'  => 'Enabled',
             'is_moderator' => 'Enabled',
        ),
        'review_changelist_commit' => array(
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_member'    => 'Enabled',
            'is_moderator' => 'Enabled',
        ),
        'review_comment_new' => array(
            'is_author'   => 'Enabled',
            'is_reviewer' => 'Enabled',
        ),
        'review_comment_update' => array(
            'is_author'   => 'Enabled',
            'is_reviewer' => 'Enabled',
        ),
        'review_comment_liked' => array(
            'is_commenter' => 'Enabled',
        ),
        'review_opened_issue' => array(
            'is_self'      => 'Enabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_moderator' => 'Enabled',
        ),
```

```
        'review_join_leave' =>   array (
            'is_self'      => 'Enabled',
            'is_author'    => 'Enabled',
            'is_reviewer'  => 'Enabled',
            'is_moderator' => 'Enabled',
        ),
    )
```

Each setting can have one of four possible values to either enable or disable notifications of that type. If multiple settings apply to a given event, then a user will receive a notification if *any* of them are enabled.

- **Enabled**: Notifications for this event and role are enabled, and a user will receive emails by default.
- **Disabled**: Notifications for this event and role are disabled, and a user will not receive emails by default.
- **ForcedEnabled**: Same as for Enabled, but this is forced enabled for all users. An individual user is not able to disable this notification type on their settings page.
- **ForcedDisabled**: As for Disabled, but this is forced disabled for all users. An individual user will not be able to enable this notification type on their settings page.

Unless one of the forced options is used, system wide options can be overridden by individual users and group owners, who can configure which notifications they receive.

Notification Roles

The various roles are as follows:

- **is_self**: This role is the user who changed the state of the review.
- **is_author**: This role is the user who was the author of the review.
- **is_reviewer**: This role includes all users and groups who are listed as being a reviewer on the review.
- **is_member**: This role includes all users who are a member of the project in which the event happened.
- **is_moderator**: This role includes all users and groups who are listed as being a moderator of the project branch in which the event happened.
- **is_follower**: This role includes all users who are followers of the project in which the event happened.
- **is_commenter**: This role is the user who was the author of a comment.

Notification events

An event is the action that causes the notification:

- **review_new**: A new review has been created.
- **review_files**: Files have been added to a review.
- **review_vote**: A user has voted on a review.
- **review_state**: The status of a review has been changed.
- **review_tests**: Automated tests have failed for a review. The first time automated tests pass for a review after a test failure for that review.
- **review_changelist_commit**: Files on a review have been committed.
- **review_comment_new**: A comment has been added to a review.
- **review_comment_update**: A comment on a review has been updated.
- **review_comment_liked**: A user has liked a comment.
- **review_join_leave**: A user or group has joined or left a review.

Obliterate Review

Note

- By default, you must be a user with *admin* or *super* user rights to obliterate a review.
- **Optional:** Swarm can be configured to allow users to obliterate reviews that they have authored. Configured by your Swarm administrator, see "Allow author obliterate review" on page 556.

Obliterate is used to permanently delete reviews that have been created by mistake. For instance, if a review is associated with the wrong changelist, or a review contains sensitive information that should not be openly available.

When you obliterate a review

- **All reviews:**
  - Metadata associated with the review is permanently deleted.
  - Events associated with the review are permanently deleted from the activity feed.
  - The Swarm 404 page is displayed if a user navigates to the review from a Swarm notification email link, or by using the URL.
- **Review with a shelved changelist:**
  - The Swarm review changelist, and any associated shelved files are permanently deleted.
  - Files shelved in the user's associated changelist are left intact.

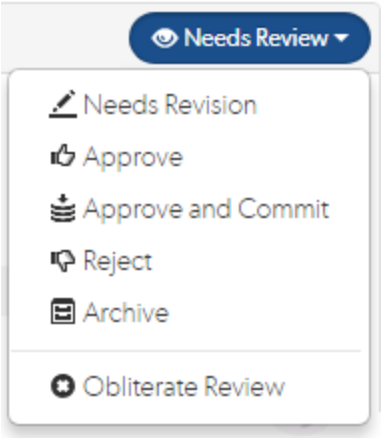
- **Review with a committed changelist:**
  - The Swarm review changelist is permanently deleted.
  - The committed changelist is left intact.

Obliterate a review

> **Important**
> Obliterate must be used with care, the review and all of its associated metadata are permanently deleted. An obliterated review cannot be reinstated, not even by Perforce Support.

To obliterate a review:

1. Navigate to the review.
2. Click the **Review state** button and select **Obliterate Review**.



3. Click **Yes** on the confirmation dialog to complete the obliterate action.
4. The review is obliterated.

OVA management

The Helix Swarm OVA is installed with Ubuntu 16.04 LTS. Ubuntu's "Long-Term Support" releases receive security updates periodically over their 5-year support window. We recommend that use of the OVA involve package updates from time to time, which can be accomplished as follows:

1. Log in to the OVA as ***root***. The password for the root account was established during setup.
2. Update the catalog of available packages:

```
# apt-get update
```

3. Download and install any updated packages:

```
# apt-get upgrade
```

See "Package management with APT" for more information:
https://help.ubuntu.com/community/AptGet/Howto

Dependency conflicts

Ubuntu software packages are often dependent on other packages that provide, for example, common libraries, utilities, and configuration. Occasionally, an upgraded package may have differing dependencies than its previous version, which can lead to dependency conflicts that can prevent package updates from completing successfully.

Should this situation occur with the Swarm OVA, use the following command to use "smart" conflict resolution, which attempts to upgrade the most important packages at the expense of less important packages if necessary:

```
# apt-get dist-upgrade
```

> **Warning**
> Upgrading packages could potentially make the OVA-hosted Swarm no longer functional. If you use the Swarm OVA in a production environment, perform the package updates on a copy of the OVA and test that Swarm continues to function properly.

P4TRUST

When Swarm is configured to connect to a Helix server (P4D) using an SSL connection, Swarm automatically executes the `p4 trust` command, which accepts the SSL fingerprint and creates a `p4trust` file containing a list of trusted servers and their fingerprints.

The location the `p4trust` file is saved to depends on whether Swarm is connected to a single Helix server or to multiple Helix servers.

- **Single Helix server:** saved as ***SWARM_ROOT*`/data/p4trust`**
- **Multiple Helix servers:** saved as a separate file for each server. For example, for **`serverA`**, **`serverB`**, and **`serverC`** they are saved as:
  - ***SWARM_ROOT*`/data/serverA/p4trust`**
  - ***SWARM_ROOT*`/data/serverB/p4trust`**
  - ***SWARM_ROOT*`/data/serverC/p4trust`**

If a certificate changes

If a certificate for a Helix server is changed for any reason then Swarm connections to that server will fail after that server is restarted.

The solution is to delete the **p4trust** file for that Helix server from the location described above. Swarm will automatically run **p4 trust** on the next request if the **p4trust** file is not found.

Projects

This section describes the configurables that can be set for projects.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Restrict project name and branch definition editing to administrators

By default, once a project has been created, any member of the project can edit or delete the project's settings. Projects can also be set to **Only Owners and Administrators can edit the project**, only owners and administrators can make changes to the project.

Instead of allowing any changes, or preventing all changes, you may want to prevent project members from making specific changes, such as to the project's name (and associated identifier), or adjusting the branch definition(s). This is useful when build infrastructure or other tooling treats these details as operational configuration, but you still want members to be able to adjust other aspects of the project configuration.

To do so, edit the *SWARM_ROOT*/**data/config.php** file, and set the following two items, similar to the following example:

```
    'projects' => array(
        'edit_name_admin_only'     => true,
        'edit_branches_admin_only' => true,
    ),
```

- **edit_name_admin_only**: when set to true, only users with *admin* privileges in the Helix server can modify a project's name.
- **edit_branches_admin_only**: when set to true, only users with *admin* privileges in the Helix server can modify a project's branch definition(s).

Both items default to **false**.

Limit adding projects to administrators

By default, any authenticated user can add new projects. Swarm can restrict project creation to users with *admin*-level privileges or higher. Once restricted, Swarm prevents non-administrators from adding projects, and does not display the **+** icon to add a project to non-administrators.

Add or update the following configuration block to the *SWARM_ROOT*`/data/config.php` file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'projects' => array(
        'add_admin_only' => true,
    ),
```

Important
If **`add_admin_only`** is enabled and `add_groups_only` has one or more groups configured, project creation is only available to users with administrator privileges and who are members of the specified groups.

Limit adding projects to members of specific groups

Swarm can restrict project creation to members of specific groups. The groups and membership need to be defined in the Helix server.

Add or update the following configuration block to the *SWARM_ROOT*`/data/config.php` file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'projects' => array(
        'add_groups_only' => array('wizards', 'slayers', 'phbs'),
    ),
```

Important
If `add_admin_only` is also enabled, project creation is only available to users with administrator privileges **and** who are members of the specified groups.

Project readme

By default, projects can have a **README** markdown file associated with them that is automatically displayed on the project overview page. This file is read from the root of the project's mainline if it is available, and displayed on the project page. See "Mainline branch identification" on page 509 for details on configuring the project mainline.

Tip

- Valid Markdown file extensions are: **md**, **markdown**, **mdown**, **mkdn**, **mkd**, **mdwn**, **mdtxt**, **mdtext**.

  If more than one **README** file is found, Swarm displays the first one it finds based on the order above.
- By default, Markdown support is limited to prevent execution of raw HTML and JavaScript content. The level of Markdown support can be configured by your Swarm administrator to **disabled**, **safe**, and **unsafe**, see "Markdown" on page 512.

Add or update the following configuration block to the *SWARM_ROOT*/**data/config.php** file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'projects' => array(
        'readme_mode'    => 'enabled',
    ),
```

- **enabled**: the use of **README** markdown files is enabled for project overview pages, text content is displayed for project overview pages with a README markdown file. This is the default.
- **disabled**: the use of **README** markdown files is disabled for project overview pages, project overview pages are not displayed.

Note

In Swarm 2018.3 and earlier, **readme_mode** could be set to **disabled**, **restricted**, or **unrestricted**. If your **readme_mode** was originally set to **restricted** or **unrestricted** and you have upgraded to Swarm 2019.1 or later, Swarm treats **readme_mode** as if it is set to **enabled**. Swarm uses the **markdown** configurable to determine the level of Markdown support (default is **safe**), see "Markdown" on page 512.

Projects tab initial fetch

By default, all of the projects are fetched for the **Projects** tab with a single call. If your Swarm system has a large number of projects, it can take some time to populate the **Projects** tab. The `fetch` configurable is used to tell Swarm to fetch and display the first `x` projects and then load the rest from cache in the background. This speeds up the initial display of projects in the **Projects** tab.

Add or update the following configuration block to the *SWARM_ROOT*`/data/config.php` file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'projects' => array(
        'fetch' => array('maximum' => 50), // defaults to 0 (disabled)
    ),
```

The default value is `0`, all projects are fetched in a single call.

Allow project members to view project settings

By default, if the **Only Owners and Administrators can edit the project** checkbox is selected for a project, only the project owners and administrators can view the project **Settings** tab.

When `allow_view_settings` is set to `true` and the **Only Owners and Administrators can edit the project** checkbox is selected for a project, project members that are not owners or administrators can view a read-only version of the project **Settings** tab. The project **Automated Tests** and **Automated Deployment** details are hidden from the project members unless they are an owner or an administrator. This enables project members to check the project settings but not change them.

Add or update the following configuration block to the *SWARM_ROOT*`/data/config.php` file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'projects' => array(
        'allow_view_settings' => true, // defaults to false
    ),
```

The default value is `false`, project members cannot see the project settings page if the project is set to **Only Owners and Administrators can edit the project**.

Redis server

Swarm requires Redis to manage its caches and by default Swarm uses its own Redis server on the Swarm machine. Swarm caches data from the Helix server to improve the performance of common searches in Swarm and to reduce the load on the Helix server.

Redis configuration:

- "Swarm connection to Redis " below: *SWARM_ROOT*`/data/config.php`.
- "Redis server configuration file" on page 542: `/opt/perforce/etc/redis-server.conf`.

Tip

- If required, you can use your own Redis server instead of the Swarm Redis server. For instructions on how to configure Swarm to use your Redis server, see "Use your own Redis server" on page 189.
- If users and groups are added, edited, or deleted in the Helix server while the Swarm server is shutdown, the Swarm user and group Redis caches will be out of sync. After the Swarm server has restarted, run a curl request to verify that the Redis caches are in sync with the Helix server. For the Redis cache verification steps, see "Manually verify the Redis caches" on page 543.
- When Swarm starts it verifies the Redis cache, during this time you cannot log in to Swarm. The time taken to verify the Redis cache depends on the number of users, groups, and projects Swarm has. Start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves in the `redis-server.conf` file, see "Redis server configuration file" on page 542.

Tip

If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Swarm connection to Redis

The `redis` block of the *SWARM_ROOT*`/data/config.php` file contains the `password`, `namespace`, `host` and `port` details of the Swarm Redis server:

```
<?php
    // this block should be a peer of 'p4'
    'redis' => array(
        'options' => array(
            'password' => null, // Defaults to null
            'namespace' => 'Swarm',
            'server' => array(
                'host' => 'localhost', // Defaults to 'localhost' or enter
your Redis server hostname
```

```
                'port' => '7379', // Defaults to '7379 or enter your Redis
server port
            ),
        ),
        'items_batch_size' => 100000,
        'check_integrity' => '03:00', // Defaults to '03:00' Use one of
the following two formats:
                                      // 1) The time of day that the
integrity check starts each day. Set in 24 hour format with leading zeros
and a : separator
                                      // 2) The number of seconds between
each integrity check. Set as a positive integer. Specify '0' to disable
the integrity check.
        'population_lock_timeout' => 300, // Timeout for initial cache
population. Defaults to 300 seconds.
    ),
```

Configurables:

- **`password`**: Redis server password. Defaults to **`null`** and should be left at default if using the Swarm Redis server.
- **`namespace`**: the prefix used for key values in the Redis cache. Defaults to **`Swarm`** and should be left at default if using the Swarm Redis server.

  > **Note**
  > If you have multiple-Swarm instances running against a single Redis server, each Swarm server must use a different Redis **`namespace`**. This enables the cache data for the individual Swarm instances to be identified. The **`namespace`** is limited to ≤ 128 characters.
  >
  > If one or more of your Swarm instances is connected to multiple-Helix servers, the Redis **`namespace`** includes the **`server label`** and the character limit is reduced to ≤ 127 characters, see "Multiple-Helix server instances" on page 519.

- **`host`**: Redis server hostname. Defaults to **`localhost`** and should be left at default if using the Swarm Redis server.
- **`port`**: Redis server port number. Defaults to **`7379`**. Swarm uses port **`7379`** as its default to avoid clashing with other Redis servers that might be on your network. It should be left at default if using the Swarm Redis server. The default port for a non-Swarm Redis server is **`6379`**.
- **`items_batch_size`**: Maximum number of key/value pairs allowed in an **`mset`** call to Redis. Sets exceeding this will be batched according to this maximum for efficiency. Defaults to **`100000`**.

> **Note**
> The default value of `100000` was chosen to strike a balance between efficiency and project data complexity. This value should not normally need to be changed, contact support before making a change to this value.

- `check_integrity`: In some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server. Swarm can run a regular integrity check to make sure that the Redis caches and Helix server are in sync. If an integrity check finds an out of sync cache file, Swarm automatically updates the data in that cache.

  The `check_integrity` configurable specifies when the Redis cache integrity check is run. Set as a specific time of day (24 hour format with leading zeros) or a number of seconds (positive integer) between checks. Disable the integrity check with `'0'`. Defaults to `'03:00'`.

- `population_lock_timeout`: specifies the timeout, in seconds, for initial cache population. If you have a large Swarm system, increase this time if the initial cache population times out. Defaults to `300` seconds.

Redis server configuration file

> **Tip**
> To fine-tune your Redis server settings, make your changes in the Laminas Redis adapter. For information about the Laminas Redis adapter, see the Laminas Redis Adapter documentation.

The Redis server configuration for Swarm is defined in **`/opt/perforce/etc/redis-server.conf`** and defaults to:

```
bind 127.0.0.1
port 7379
supervised auto
save ""
dir /opt/perforce/swarm/redis
```

Default values:

> **Tip**
> - The default settings are shown below, the **`redis-server.conf`** file contains more detailed information about the Redis configuration for Swarm.
> - On Swarm systems with a large number of users, groups, and projects, start-up time can be improved by persisting the memory cache. You can persist the memory cache by disabling background saves and enabling append saves, see the **`redis-server.conf`** file comments for detailed information.

- `bind 127.0.0.1` - Redis server IP address (loopback interface)
- `port 7379` - Redis server port number
- `supervised auto` - detects the use of `upstart` or `systemd` automatically to signal that the process is ready to use the supervisors
- `save ""` - background saves disabled, recommended.
- `dir /opt/perforce/swarm/redis` - the directory the Redis cache database is stored in.

Manually verify the Redis caches

Important

- Only *admin* and *super* users can verify the Redis cache.
- On Swarm systems with a large number of users, groups, and projects verification of the caches can take some time and can impact performance.

  In normal operation, the Redis cache and Helix Core server data should stay in sync and there is no need to verify the caches.

  If you have a large number of users, groups, and projects, we recommend that you verify the individual user and group caches rather than using **Verify All**.

In normal operation the Redis cache and Helix Core server data should stay in sync and there is no need to verify the caches. However, in some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server.

If you suspect that you have a cache issue, you can verify the cache manually from the System Information page.

For example, to verify the User cache:

1. Navigate to the **User id** dropdown menu.
2. Select **System Information**.
3. Click the **Cache Info** tab.
4. Click the **Verify User** button.

   Swarm responds with a message to say that it has successfully verified the user cache.

   If verification finds the cache is out of sync with the Helix server, Swarm automatically updates the data in that cache.

   For more information, see "Verify buttons" on page 587.

Check progress of the verification request

To check the progress of the verification request:

1. From the **Cache Info** tab.
2. Click the **Refresh Status** button.

Review cleanup

This section describes the configurables that can be set for review cleanup.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

When a review is created in Swarm, it creates its own version of the changelist, leaving the user's own changelist untouched so as not to interfere with the user's ongoing work. Each time new work is submitted to the review, Swarm creates a new changelist so that there is a versioned history of the review.

When the review is finally committed it is Swarm's own changelist that is committed. Swarm's changelists are generally hidden from the users, but the user's changelist will remain open. By default it is necessary for the user to tidy up and remove this changelist themselves after the review has been completed.

There is an option to do this automatically when the review is committed. Configuration for this is expressed with a reviews block in the *SWARM_ROOT*`/data/config.php` file, similar to the following example:

```
<?php
    'reviews' => array(
        'cleanup'              => array(
            'mode'        => 'user', // auto - follow default, user -
present checkbox(with default)
            'default'     => false,  // clean up pending changelists on
commit
            'reopenFiles' => false,   // re-open any opened files into the
default changelist
        ),
    ),
```

By default, this option is enabled but defaults to no clean up (so a user can select the option if they want to when they commit a review).

If the Helix server user that Swarm is configured to use is a super user, then the user can clean up all user changelists associated with a review. If this is not the case, then the user who commits a review can only clean up changelists that are in their name.

By default, Swarm only cleans up changelists that are owned by the committing user. In the case where a user is committing a review that has been contributed to by other users, their changelists will not be cleaned up.

If you want to configure Swarm to clean up all changelists contributing to a review, regardless of whether they are owned by the committing user, you can do this by granting the Helix server user that Swarm is configured as 'super' permissions/privileges (rather than just the 'admin' permissions/privileges that Swarm requires for its other operations).

Note
There is an API option that allows full cleanup to be executed with *super user* permissions using an external script. This removes the need for the Swarm Helix server user to have super privileges. See Pending review cleanup example for details.

- **mode**

  If the mode is `user` then a checkbox is displayed when a review is committed, and the user has the option as to whether to clean up changelists or not.

  If the mode is `auto`, then no checkbox is displayed, and all committed reviews will either always be tidied up, or never will be, depending on the value of `default`.

- **default**

  If mode is set to `user`, then this determines whether the checkbox shown on commit is ticked by default or not.

  If mode is set to `auto`, then it determines the action to be taken automatically. If set to `true` then changelists will always be cleaned up, otherwise they will never be cleaned up.

- **reopenFiles**

  If a changelist has files checked out (not shelved), then it cannot be deleted. Setting `reopenFiles` to `true` will mean that when a changelist is cleaned up, any opened files will first be moved to the default changelist, allowing the changelist to be removed.

  If set to `false`, then a changelist with checked out files will not be cleaned up.

  If users normally revert their files after shelving them, then this option may not be needed. If set to `true` then it may result in files appearing in the user's default changelist unexpectedly.

The review cleanup feature is designed to help users keep their workspaces clean, and prevent the proliferation of unwanted changelists after reviews have been approved and committed.

However, it cannot guarantee to be perfect, and because it is taking actions automatically on the part of the user, it may do things that the user doesn't expect. The following caveats should be kept in mind when using this feature.

- The changelists created by Swarm itself will not be touched by this process. There will always be a record of the review history that is kept.
- If the user who commits a review is normally different to the user that did the work, then unless Swarm runs as a super user then many changelists will not be cleaned up.
- If the committer created some of the changelists, then those changelists will be removed. However, changelists created by other users will not be removed.
- If the user has shelved files into changelists without reverting them, then they will still remain in the user's local workspace, and will need to be cleaned up manually.

Review keyword

This section describes how to configure keywords, and how to use keywords in a changelist description to create a review, and add a changelist to a review.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Review keyword configuration

The keyword can be configured with a regular expression so that most any keyword syntax can be used. If you choose to customize the review keyword, take care to choose syntax and terminology that is unlikely to occur in a changelist description, to avoid unexpected Swarm activity.

To configure the review keyword, add the following block to the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'patterns' => array(
            'octothorpe' => array(     // #review or #review-1234 with
surrounding whitespace/eol
                'regex'  => '/(?P<pre>(?:\s|^)\(?)\#
(?P<keyword>review|append|replace)(?:-(?P<id>[0-9]+))?(?P<post>[.,!?:;)]*
(?=\s|$))/i',
                'spec'   => '%pre%#%keyword%-%id%%post%',
                'insert' => "%description%\n\n#review-%id%",
                'strip'  => '/^\s*\#(review|append|replace)(-[0-9]+)?
```

```
(\s+|$)|(\s+|^)\#(review|append|replace)(-[0-9]+)?\s*$/i',
            ),

            'leading-square' => array(     // [review] or [review-1234] at
start
                'regex'  => '/^(?P<pre>\s*)\
[(?P<keyword>review|append|replace)(?:-(?P<id>[0-9]+))?\](?P<post>\s*)/i',
                'spec'  => '%pre%[%keyword%-%id%]%post%',
            ),

            'trailing-square' => array(     // [review] or [review-1234]
at end
                'regex'  => '/(?P<pre>\s*)\
[(?P<keyword>review|append|replace)(?:-(?P<id>[0-9]+))?\]
(?P<post>\s*)?$/i',
                'spec'   => '%pre%[%keyword%-%id%]%post%',
            ),
        ),
    ),
```

Multiple patterns can be specified; the first successful match is used and none of the other patterns are evaluated.

The keyword types are grouped under their identifiers. In each group, the **`regex`** item specifies the regular expression to be used to identify the review keyword in the changelist description. The spec item is used when the review keyword needs to be updated.

> **Note**
> The use of named capture groups in the **`regex`**, for example **`(?<pre>\s*)`**. The values captured during regex matching are used to replace any identically named placeholder values in the spec item that are surrounded by percent % characters. In the example configuration above, the pre and post capture groups and placeholders maintain any whitespace surrounding the review keyword.

For **`octothorpe`** (or "hashtag") review keywords, these can appear anywhere in the changelist description. The **`strip`** item is used to ensure that the keyword is removed from the review description if it appears at the start or end of the changelist description. The **`insert`** item is currently not used; it is included here to prevent future upgrade issues. The intended use case is when a review is started and the changelist does not already contain a review keyword, the insert item would be used to add the review keyword to the changelist description.

For more information on named capture groups in PHP, see: http://www.regular-expressions.info/named.html

Create a review

By default, including the keyword `#review` within a changelist description (separated from other text with whitespace, or on a separate line) tells Swarm that a review should begin when the changelist is shelved or committed.

Once a review has begun, Swarm adds the review identifier to the `#review` keyword, for example `#review-1234`. This tells Swarm which review should be updated whenever the original changelist is re-shelved or committed.

Note

- If you create a pre-commit review and a user appends a changelist to your review, the default add mode of your review is changed from replace to append, see "Add a changelist to a review" below for details.
- Swarm acts on the first valid keyword it finds in the changelist description, Swarm then ignores any further valid keywords it finds in the description.
- Swarm can also accept `[review]` at the start or end of the changelist description, but this form of review keyword is now deprecated and is likely to be removed in a future version of Swarm.

Add a changelist to a review

Once a review has been started you can add a changelist to the review. It can be useful to add changelists to an existing review. For example, if follow up changes are made to files in a review or if you need to group a number of changelists under a single review.

Note
The changelist must not be part of another review, if it is Swarm will reject it.

Note
Swarm acts on the first valid keyword it finds in the changelist description, Swarm then ignores any further valid keywords it finds in the description.

By default, the `#append-`, `#replace-`, and `#review-` keywords along with the *review identifier* (separated from other text with whitespace, or on a separate line) can be used in a changelist description to add changelists to an existing review. The options available depend on whether the review is pre-commit or post-commit:

- **Pre-commit reviews:**
  - `#append-` + *review identifier*. When you add a changelist to a review with the append option, the files in the changelist are appended to the existing files in the review. This changes the default add mode of the review to append.

- `#replace-` + *review identifier*. When you add a changelist to a review with the replace option, all of the files in the review are replaced with the files in the changelist you are adding to the review. This changes the default add mode of the review to replace.

  > **Note**
  > If you replace a pre-commit review with a committed changelist, the new version of the review will be a post-commit review.

- `#review-` + *review identifier*. When you add a changelist to a review with this option, the review's default add method is used. The default add method can be either replace or append and it is set by the most recent add method used on the review:
  - When a review is first created the default add mode for the review is replace.
  - If a changelist is added to the review using append, the default add mode for the review is set to append.
  - If a changelist is added to the review using replace, the default add mode for the review is set to replace.
  - If you do not know what the current default add mode of the review is, delete the `#review-` keyword and specify the add mode you want by using either `#append-` + *review identifier*, or `#replace-` + *review identifier*.

- **Post-commit reviews:**
  - `#review-` or `#replace-` + *review identifier*. When you add a changelist to a review with the replace changelist option, all of the files in the review are replaced with the files in the changelist you are adding to the review.

    > **Note**
    > If you replace a post-commit review with a pending changelist, the new version of the review will be a pre-commit review.

> **Tip**
> When the content of a review is changed, Swarm checks to see which branches are in the new revision of the review:
>
> - **If a new branch was added to the review:**
>   - Default reviewers on the new branch are added to the review.
>   - Moderators from the added branch become moderators for the review alongside the existing moderators.
>   - **Only if workflow is enabled:** if the new branch is associated with a workflow, the workflow is merged with the existing workflow. The most restrictive workflow is used for the review.

- **If a branch is no longer part of the review:**
  - Reviewers for the review are not changed.
  - Moderators from the removed branch no longer moderate the review.
  - **Only if workflow is enabled:** if the branch was associated with a workflow, the branch workflow is removed from the review.

Example: Appending a changelist to review 1234:

Add `#append-1234` to the changelist description.

- Original review contains the following files: $A^{\#1}$, $B^{\#2}$, $C^{\#1}$, $D^{\#1}$, and $E^{\#4}$
- Changelist contains the following files: $A^{\#2}$, $C^{\#3}$, $E^{\#4}$ (marked for delete), and $F^{\#1}$
- Appending the changelist to the original review generates a new revision of the review that contains the following files: $A^{\#2}$, $B^{\#2}$, $C^{\#3}$, $D^{\#1}$, $E^{\#4}$ (marked for delete), and $F^{\#1}$

Important

Committing a review with Swarm (recommended): Swarm automatically commits the files in the approved revision of the review.

Committing a review outside of Swarm:

Before you commit the review:

1. Unshelve the review into the pending changelist associated with the review.
2. Reshelve the files in the pending changelist.
3. Commit the pending changelist.
4. This process ensures that all of the files in the approved revision of the review are committed.

   Workflow feature enabled **(default):**

   Swarm can be configured to automatically check that the files being committed match the files in the approved revision of the review by using the **On commit with a review** workflow rule. For information about the **On commit with a review** rule, see "Workflow rules" on page 423.

   Workflow feature disabled**:**

   Swarm can be configured to automatically check that the files being committed match the files in the approved revision of the review by using the strict trigger option.

Example: Replacing the files in a review 1234 with the files in a changelist:

Add `#replace-1234` to the changelist description.

- Original review contains the following files: $A^{\#1}$, $B^{\#2}$, $C^{\#1}$, $D^{\#1}$, and $E^{\#4}$
- Changelist contains the following files: $A^{\#2}$, $C^{\#3}$, $E^{\#4}$ (marked for delete), and $F^{\#1}$

- Replacing the original review with the changelist generates a new revision of the review that contains the following files: A#2, C#3, E#4 (marked for delete), and F#1

> **Tip**
> When you replace a review with a changelist, the base versions of the files in the new revision of the review are the base versions of the files in the replacement changelist.

Keyword workflow

For workflow examples using keywords, see "Review creation, and modification outside of Swarm" on page 385.

Reviews filter

This section describes the configurables that can be set for the reviews filter.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Swarm review sorting on the "Reviews list" on page 342 page is customized by using the following configuration block in the `SWARM_ROOT/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'filters' => array(
           'filter-max' => 15,
           'result_sorting' => true,
           'date_field' => 'updated', // 'created' displays and sorts by created
date,'updated' displays and sorts by last updated
        ),
    ),
```

filter-max

When you visit the "Reviews list" on page 342 page, reviews are displayed incrementally **x** reviews at a time until the white space on the page is filled with reviews or there are no more reviews to display. This process continues as you scroll down the page. The **`filter-max`** configurable sets this number.

Default value is `15`.

result_sorting

Controls whether the **Result order** button is displayed on the "Reviews list" on page 342 page.

- Set `result_sorting` to `true` to display the **Result order** button on the "Reviews list" on page 342 page, this is the default setting.
- Set `result_sorting` to `false` to remove the **Result order** button from the "Reviews list" on page 342 page, reviews are sorted by when they were created.

date_field

> **Note**
> The `date_field` setting is only used if `result_sorting` is set to `true`.

`date_field` sets the initial setting for the **Result order** button on the "Reviews list" on page 342 page.

- `created`: reviews are sorted by when they were created. This is the default setting.
- `updated`: reviews are sorted by when they were last updated.

When a user starts a new session the **Result order** button is set to the initial sort order set in `date_field`. The review sort order can be changed by the user from the **Result order** button. This setting is remembered even if the user navigates away from the "Reviews list" on page 342 page. When the user starts a new session, the **Result order** button is reset back to the setting in `date_field`.

Reviews

This section describes the configurables that can be set for reviews.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Disable self-approval of reviews by authors

The Swarm 2015.2 release provides the ability to disable review approval by authors, even if they are moderators or administrators. This is useful for development workflows where review by others is of paramount importance.

To disable review approval by authors, update the *SWARM_ROOT*/**data/config.php** file to include the following configuration item within the reviews block:

```
'reviews' => array(
    'disable_self_approve' => true,
),
```

The default value is **false**.

Moderator behavior when a review spans multiple branches

By default, when a review spans multiple project branches that have different moderators, only one moderator from any one of the branches needs to approve the review. Set the configurable to **each** to require one moderator from each branch approve the review. If a moderator belongs to more than one of the branches spanned by the review, their approval will count for each of the branches they belong to.

To require one moderator from each branch to approve a review, edit the *SWARM_ROOT*/**data/config.php** file and set the **moderator_approval** configurable to **each** in the **reviews** section of the codeblock:

```
'reviews' => array(
    'moderator_approval' => 'each', // 'any|each'
),
```

The default value is **any**.

Prevent Approve for reviews with open tasks

When enabled, Swarm will not allow you to **Approve**, or **Approve and Commit** a review that has open tasks. To approve, or approve and commit a review with open tasks, you must address the tasks first and then set them to **Task Addressed**, or **Not a Task**, see "Set a task to Task Addressed or Not a Task" on page 270 for details.

> **Important**
> If the open tasks on the review are archived, the review can then be approved, or approved and committed.

To prevent reviews being approved, or approved and committed when the review has open tasks, update the *SWARM_ROOT*/**data/config.php** file to include the following configuration item within the reviews block:

```
'reviews' => array(
    'disable_approve_when_tasks_open' => true,
),
```

The default value is `false`, in this case a warning is displayed for the open tasks but the action is allowed.

Protected end states

By default, the content of a review can be updated no matter what state the review is in. Reviews can be protected from automatic content change if they are in a specified state by using the `end_states` configurable.

> **Tip**
> When a review is in a protected end state, it can still be updated manually by a user from the Swarm UI.

Workflow:

- **If workflow is enabled** (default): the `end_states` configurable sets the protected review states. A combination of the **On update of a review in an end state** rule and workflows determines which projects and branches have protected reviews.
- **If workflow is disabled:**: when the content of a review is updated, the `end_states` array is checked to see if the review can be updated. If the review state matches a state in the `end_states` array the submit is rejected.

To set the protected review states, update the *SWARM_ROOT*`/data/config.php` file to include the following configuration item within the reviews block. In this example the **Archived**, **Rejected**, and **Approved : Committed** states are specified:

```
            'reviews' => array(
                'end_states' => array('archived','rejected',
'approved:commit'),
            ),
```

Valid review states for the array:

- `archived`: the review state is **Archived**.
- `rejected`: the review state is **Rejected**.
- `approved:commit`: the review has been approved and committed, the review state is **Approved : Commit**.
- `approved`: the review state is **Approved**.

By default the array is empty, review content can be updated no matter what state the review is in.

Process shelf file delete when

By default, when you delete files from a shelved changelist, the files are not removed from the associated review.

When you have a review state set in the **process_shelf_delete_when** array, Swarm will automatically carry out the following steps when a file is deleted from a shelf:

1. Check to see if the shelf is associated with a review.
2. The deleted file is only removed from the review if the review state matches the review state in the **process_shelf_delete_when** array.

Tip

- More than one review state can be specified in the **process_shelf_delete_when** array.
- When files are removed from a review, any votes on the review become stale. The vote counts are reset, and the vote indicators are muted.
- If **process_shelf_delete_when** is configured and the review is in a state that is specified in the array: When the review contains a file that is a common to multiple changelists that are associated with the review, if the owner of any of the associated pending changelists deletes the common file from their shelf, that file will be removed from the review, and a new version of the review will be created. This is true even if the file is not the most recent version in the review.

Important
Required for **process_shelf_delete_when**:

- A supported version of Helix server (standard maintenance), see "Helix Core server requirements" on page 107.
- You must have the *swarm.shelvedel shelve-delete* trigger line in the Helix server trigger table, see "Helix Core server configuration for Swarm" on page 158.
- You must satisfy the Swarm trigger dependencies, see "Trigger dependencies" on page 109.

Important
Do not use the Swarm user that is configured in the Swarm configuration file when deleting shelves, or deleting files from shelves. The Swarm logic processes the *shelve-delete* trigger event, if the event is invoked by the Swarm user it is rejected. The delete operations will fail.

To remove files from a review when they have been deleted from an associated shelved changelist and the review is in a specified state, update the *SWARM_ROOT*/**data/config.php** file to include the following configuration item within the reviews block. In this example the **Needs Review** and **Needs Revision** states are specified:

```
            'reviews' => array(
                'process_shelf_delete_when' => array('needsReview',
```

```
'needsRevision'),
            ),
```

Valid review states for the array:

- **`needsReview`**: the review state is **Needs Review**.
- **`needsRevision`**: the review state is **Needs Revision**.
- **`rejected`**: the review state is **Rejected**.
- **`approved`**: the review state is **Approved**.
- Leave the array empty to disable this feature, see below. This is the default setting for the array.

```
            'reviews' => array(
                'process_shelf_delete_when' => array(),
            ),
```

> **Note**
> If a review is **Approved and Committed** or **Archived**, files will not be removed from the review when they are deleted in an associated shelf. This is true even if the **`approved:commit`** or **`archived`** states are in the **`process_shelf_delete_when`** array.

Allow author change

When enabled, the author of a review can be changed. This is useful in the case where the original author is no longer available, and someone else needs to take over ownership.

To allow the author of a review to be changed, update the *`SWARM_ROOT`***`/data/config.php`** file to include the following configuration item within the reviews block:

```
            'reviews' => array(
                'allow_author_change' => true,
            ),
```

The default value is **`false`**.

Allow author obliterate review

When enabled, users can obliterate reviews that they have authored. For details about how to obliterate a review, see "Obliterate Review" on page 533.

Important
By default, only users with *admin* and *super* user rights can obliterate a review. Perforce recommends that the **`allow_author_obliterate`** configurable is kept at its default value of **`false`**. An obliterated review cannot be reinstated, not even by Perforce Support.

To allow users to obliterate reviews that they have authored, update the *SWARM_ROOT*`/data/config.php` file to include the following configuration item within the reviews block:

```
            'reviews' => array(
                'allow_author_obliterate' => true,
            ),
```

The default value is **`false`**.

Synchronize review description

By enabling the synchronization of review descriptions, it becomes possible to update the description of a review by updating the description of a changelist associated with the review. Whenever an associated changelist is saved, the text of the review will be updated to match.

Note
The update is not triggered if:

- another changelist is attached to a review.
- files are updated in a changelist attached to a review
- a user selects **Update pending changelist** when editing a review description.

To enable synchronize review descriptions, update the *SWARM_ROOT*`/data/config.php` file to include the following configuration item within the reviews block:

```
            'reviews' => array(
                'sync_descriptions' => true,
            ),
```

The default value is **`false`**.

Expand all file limit

The review page has an **Expand All** button that opens all the files within that review. If the number of files is large, clicking the button might affect performance.

The **`expand_all_file_limit`** disables the button if the number of files in the review exceeds the given value. If the value is set to zero, the button is always enabled and can therefore open all the files.

To change the expand all file limit, update the *SWARM_ROOT*/**data/config.php** file to include the following configuration item within the reviews block:

```
'reviews' => array(
    'expand_all_file_limit' => 10,
),
```

The default value if the option is not specified is **10**.

Expand group reviewers

By default, reviewer group members are not displayed in the *Individuals* area of the reviews page when they interact with a review (vote, comment, update, commit, archive, etc.). This avoids overloading the *Individuals* area with individual avatars if you have large reviewer groups.

> **Note**
> An exception to this behavior is when a member of a reviewer group is also an individual required reviewer, in this case their avatar will be displayed in the *Individuals* area.

When **expand_group_reviewers** is set to **true**, reviewer group members are added to the *Individuals* area of the review page when they interact with the review (vote, comment, update, commit, archive, etc.). If you have large reviewer groups, this might affect performance.

To expand group reviewers, update the *SWARM_ROOT*/**data/config.php** file to include the following configuration item within the reviews block:

```
'reviews' => array(
    'expand_group_reviewers' => true,
),
```

The default value is **false**.

Disable tests on approve and commit

> **Important**
> - From Swarm 2020.1, the **disable_tests_on_approve_commit** configurable has been removed from Swarm. This functionality has been moved to workflows and is set on a per workflow per test basis. A test can be set to run **On Update** or **On Submit** for a specific workflow or globally by setting the test on the global workflow.
> - If the **disable_tests_on_approve_commit** configurable is in your "swarm_root" on page 580/**data/config.php** file, it is ignored.

More context lines

The Review display and Changelist display pages have **Show More Lines above** and **Show More Lines Below** buttons on the file diff view when a file is expanded. The buttons are used to display more lines of context for the file being viewed. By default 10 extra lines are displayed.

To change the number of lines displayed when the buttons are clicked, update the *SWARM_ROOT*`/data/config.php` file to include the following configuration item within the reviews block:

```
            'reviews' => array(
                'more_context_lines' =>15, // defaults to 10 lines
            ),
```

The default value if the option is not specified is `10`.

Review complexity

Review complexity is displayed for each review in the **Complexity** column on the "Reviews list" on page 342 page to help users to quickly see how complicated each review in the list is. By default, review complexity is based on the number of lines changed in the files in a review and simply displayed as high, medium, or low for each review. The values for high and low complexity are set using the `high` and `low` configurables.

> **Tip**
> - Review complexity is only calculated for a review when the review is updated and the file content has changed.
> - Review complexity is only stored for the current revision of a review.
> - If you change the `high` or `low` complexity value, complexity is only recalculated for a review when the review is updated and the file content has changed.

To change the complexity values, update the *SWARM_ROOT*`/data/config.php` file with item within the reviews block:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'statistics' => array(
            'complexity' => array(
                'calculation' => 'default', // 'default|disabled'
                'high' => 300,
                'low' => 30
            ),
```

```
        ),
    ),
```

- **`calculation`**
  - **`default`** the complexity is based on the number of lines changed in the files in the review. For the purposes of complexity, a new file is counted as the number of lines in the file. Default value is **`default`**.
  - **`disabled`** set to **`disabled`** to disable the complexity calculation.
- **`high`** set the number (integer) of line changes that are considered high for your organization. Default value is **`300`**.
- **`low`** set the number (integer) of line changes that are considered low for your organization. Default value is **`30`**.

By default, complexity is calculated as:

- **High:** number of lines changed in files in the review ≥ **`high`**.
- **Medium:** number of lines changed in files in the review < **`high`** and > **`low`**.
- **Low:** number of lines changed in files in the review ≤ **`low`**.

Search

Swarm's search feature combines user, group, project, and file path searching. The standard Swarm searches can be extended to search file content and changelist description by using the optional Helix Core Search API.

You can download the P4Search installer from the **Helix Core APIs** section of the Perforce download page. For more information, see the Helix Core Search Developer Guide and the P4Search release notes.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Configure Swarm search with the following configuration block in the *`SWARM_ROOT`***`/data/config.php`** file:

```
<?php
    // this block should be a peer of 'p4'
    'search' => array(
        'maxlocktime'     => 5000, // 5 seconds, in milliseconds
        'p4_search_host'  => 'http://myp4search.mydomain.com:4567', //
```

```
optional URL to the P4Search host
    ),
```

maxlocktime

The **maxlocktime** key specifies the maximum amount of time, in milliseconds, that any table within the Helix server should be locked while performing **fstat** command searching. Increasing this value might allow better search results at the expense of potentially blocking other queries on the Helix server. Decreasing this value impacts the Helix server less, but may be insufficient for returning the desired search results.

p4_search_host

The **p4_search_host** configurable specifies the URL of your P4Search server. When configured, Swarm issues API calls to P4Search to take advantage of its file content and description indexing.

Security

There are many strategies for securing a Swarm installation. This section provides guidance on security features Swarm controls, and recommendations for several areas for the system hosting Swarm.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Require login

> **Important**
> Prior to Swarm's 2016.1 release, **require_login** defaulted to **false**. For 2016.1 and later releases, the default is **true**.

By default, Swarm prevents anonymous users from viewing any Helix server resources; users must login to see commits, reviews, etc.

Swarm can be configured to allow anonymous users to access any readable resources (creating or editing resources by anonymous users is not permitted). Add the following configuration block to the *SWARM_ROOT*/**data/config.php** file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
```

```
        'require_login' => false, // defaults to true
    ),
```

There is one exception: the **`/queue/worker`** endpoint is available to any user.

> **Note**
> Service and operator users are not permitted to login. For more information on these user types, see User types in *Helix Core Server Administrator Guide*.

Prevent login

When your Helix server has users that should not be able to log in to Swarm, for example *service* users involved with Helix server replicas, the **`prevent_login`** configuration item can be used to prevent successful authentication.

Add or update the following configuration block to the *`SWARM_ROOT`***`/data/config.php`** file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
        'prevent_login' => array(
            'service_user1',
            'service_user2',
        ),
    ),
```

`prevent_login` defaults to **`array()`**, which means no users in your Helix server are prevented from logging into Swarm.

For more information, see Service users in *Helix Core Server Administrator Guide*.

Auto-create new users

Swarm supports the automatic-creation of new users in Perforce on login if the Helix server is configured to allow it and the user exists in LDAP. This function is configured in the Helix server and no Swarm configuration is required. For instructions on how to configure Helix server to automatically create new users in Perforce on login, see **`auth.ldap.userautocreate`** and **`auth.default.method`** in the Defining LDAP-related configurables section of the *Helix Core Server Administrator Guide*.

Sessions

Swarm manages logged-in sessions using cookies in the browser, and PHP session storage on the server. Swarm uses reasonable defaults for the cookie and session lifetimes (measured in seconds); when the lifetime is exceeded users need to login again. To specify session lifetimes and garbage collection frequency, add the following configuration block to the *SWARM_ROOT*`/data/config.php` file, at the same level as the `p4` entry:

```
<?php
    // this block should be a peer of 'p4'
    'session' => array(
        'cookie_lifetime'             => 0, // 0=expire when browser
closed
        'remembered_cookie_lifetime'  => 60*60*24*30,  // 30 days
        'gc_maxlifetime'              => 60*60*24*30, // 30 days
        'gc_divisor'                  => 100, // 100 user requests
    ),
```

- `cookie_lifetime`

  Optional: Limits the lifetime of session cookies. The default is `0`, which causes the session cookie to expire when the user's browser is closed.

  > **Note**
  > When the **Remember Me** checkbox on the login dialog is checked, the `remembered_cookie_lifetime` value will be used for `cookie_lifetime`.

- `remembered_cookie_lifetime`

  Optional: Limits the lifetime of session cookies when the **Remember Me** checkbox on the login dialog is checked. The default is `60*60*24*30` seconds (30 days).

- `gc_maxlifetime`

  Optional: If a session is inactive for the specified number of seconds, the user is logged out. The default is `60*60*24*30` seconds (30 days). User sessions are stored in *SWARM_ROOT*`/data/sessions/`.

  > **Note**
  > By default, the user's Perforce ticket expires after 12 hours, which also causes them to be logged out.

- `gc_divisor`

Optional: Sets how often garbage collection is run based on the number of user requests that are made. The setting range is 1 to 100. Garbage collection deletes user session files that are older than the `gc_maxlifetime` setting from *SWARM_ROOT*`/data/sessions/`.

- `gc_divisor` **= 100:** Garbage collection runs after every 100th user request. 100 is the default setting.
- `gc_divisor` **= 1:** Garbage collection runs after every user request.

> **Important**
> If your Swarm system has a large number of users, setting `gc_divisor` to a low number can result in performance issues.

X-Frame-Options header

By default, Swarm emits a `X-Frame-Options` HTTP header set to `SAMEORIGIN`. This prevents embedding of the Swarm interface into other web pages, which avoids *click-jacking* attacks.

If your deployment of Swarm needs to be integrated into another web interface, you can adjust the `X-Frame-Options` header by adjusting the `x_frame_options` item within the security configuration block, found in the *SWARM_ROOT*`/data/config.php` file. For example:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
        'x_frame_options' => value,
        ),
    ),
```

Where value can be one of:

- `'SAMEORIGIN'` - Swarm can only be displayed in a frame hosted on the same domain.
- `'DENY'` - Swarm cannot be displayed in a frame.
- `'ALLOW-FROM URI'` - Swarm can only be displayed in a frame hosted on the specified URI.
- `false` - The `X-Frame-Options` header is not emitted, so Swarm can be embedded without restriction.

For more information on the `X-Frame-Options` header, see this Mozilla Developer Network article.

For more information on click-jacking attacks, see this Wikipedia article.

Disable commit

Swarm provides the ability to commit reviews within the Swarm interface. You may want to disable this capability to prevent reviews from being committed by someone other than the review's author. When disabled, the **Approve and Commit** (and **Commit** if the review is already approved) option is removed from the list of states available to a code review.

To disable commits, set **`disable_commit`** to **`true`** within the reviews item in the *SWARM_ROOT*`/data/config.php` file. For example:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'disable_commit' => true,
        ),
    ),
```

Restricted Changes

The Helix server provides two changelist types: **`public`** (the default), and **`restricted`**. Swarm honors restricted changelists by preventing access to the changelist, and any associated comments or activity related to the changelist.

If a user has *list*-level privileges to at least one file in the changelist, Swarm allows the user to see the changelist and any of the files they have permission to see.

To prevent unintended disclosures, email notifications for restricted changes are disabled by default. To enable email notifications for restricted changes, set **`email_restricted_changes`** to **`true`** within the security item in the *SWARM_ROOT*`/data/config.php` file. For example:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
        'email_restricted_changes' => true,
        ),
    ),
```

Note
When **`email_restricted_changes`** is set to **`true`**, email notifications for restricted changes are sent to all interested parties with no permissions screening. These notifications might disclose sensitive information.

Swarm can only report on changes that the configured *admin*-level user has access to. When using restricted changes, we advise that you grant the Swarm*admin*-level user access to the restricted files and set `require_login = true` to avoid leaking information to unauthenticated users.

Limit adding projects to administrators

Important
For Swarm 2016.1, the configuration item **add_project_admin_only** was moved from the **security** block to the **projects** block, and the item was renamed to **add_admin_only**. The functionality of this configuration item remains unchanged.

If you do not update your *SWARM_ROOT*/**data/config.php** configuration file, the old configuration for restricting project creation to administrators continues to work.

If you add the new configuration item **add_admin_only** to the **projects** block, it takes precedence over any remaining **add_project_admin_only** setting in the **security** block.

Limit adding projects to members of specific groups

Important
For Swarm 2016.1, the configuration item **add_project_groups** was moved from the **security** block to the **projects** block, and the item was renamed to **add_groups_only**. The functionality of this configuration item remains unchanged.

If you do not update your *SWARM_ROOT*/**data/config.php** configuration file, the old configuration for restricting project creation to specific groups continues to work.

If you add the new configuration item **add_groups_only** to the **projects** block, it takes precedence over any remaining **add_project_groups** setting in the **security** block.

IP address-based protections emulation

A Helix server can be configured via *protections* to restrict access to a depot in a variety of ways, including by IP address. As Swarm is a web application acting as a client to the Helix server, often with *admin*-level privileges, Swarm needs to emulate IP address-based restrictions. It does so by checking the user's IP address and applying any necessary restrictions during operations such as browsing files, viewing file content, viewing and adding comments on files.

Swarm also emulates proxy-based protections, in addition to regular IP-based protections emulation. However, Swarm does not detect whether it is connecting to a Helix Proxy or not; it merely attempts to emulate protections table entries that use proxy syntax.

IP address-based protections emulation is enabled by default. Swarm performs somewhat faster without this emulation; if you do not require them for your Swarm installation these can be disabled by setting the configuration:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
        'emulate_ip_protections' => false,
    ),
```

Tip
If Swarm is connected to multiple-Helix server instances, you have the following options for **`emulate_ip_protections`**:

- To apply a global **`emulate_ip_protections`** setting to all Helix server instances that Swarm is connected to, set **`emulate_ip_protections`** in the **`security`** block as shown above.
- To apply an individual **`emulate_ip_protections`** setting to a Helix server instance, set **`emulate_ip_protections`** in the **`serverid`** block for that server. This value overrides the global setting in the **`security`** block for the Helix server instance.

For more information about configuring Swarm for multiple-Helix server instances, see "Set up the Swarm configuration file for the Helix servers" on page 520.

Known limitations

- Notification e-mails for reviews or commits include the list of affected files. Swarm cannot reliably know the IP address used to retrieve that e-mail, and makes no attempt to filter the files and their depot paths nor any details included in the description. However, when a user follows a link from the notification e-mail to a restricted resource, that access is denied.
- Swarm filters comments from activity streams, but any comments created prior to upgrading to the 2013.3 release cannot be filtered and may leak sensitive information.
- Swarm displays a comment count in code the review list, code reviews, jobs, and activity streams, but the count does not account for any comments that may be hidden from the user due to association with files the user is restricted from viewing.
- Should Swarm users connect to Swarm via a proxy or VPN, the protections will generally use the IP address of the proxy/VPN.
- When the user's IP address and Swarm's IP address both have restrictions applied, the user experiences the most constraining of the two IP address-based restrictions; Swarm cannot bypass restrictions applied to itself.
- Swarm performs a variety of operations with *admin*-level privileges, on behalf of a user. Even if the Helix server has IP-based, or userid-based protections, installed to prevent access to some or

most of its versioned data, Swarm typically does have access to this data. Therefore, *Swarm cannot guarantee that no information leakage will occur*, particularly when custom modules are in use, or Swarm source has been customized.

For more information, see Authorizing Access in *Helix Core Server Administrator Guide*.

Disable system info

Swarm provides a **System Information** page, available to users with *admin* or *super* privileges, which displays information about Helix server that Swarm is configured to use, as well as PHP information and the Swarm log file.

While this information can be invaluable when communicating with Perforce support engineers, you may wish to prevent disclosure of any system information. The **System Information** page can be disabled for all users by adding the following configuration block to the *SWARM_ROOT*`/data/config.php` file, at the same level as the p4 entry:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
        'disable_system_info' => true, // defaults to false
    ),
```

Once disabled, the **System Information** link disappears from the About Swarm dialog, and 403 errors are generated for any attempts to browse to the **System Information** page.

HTTP client options

Swarm permits configuration of options that are passed through to the underlying Laminas Framework HTTP client. These options can be used to specify SSL certificate locations, request timeouts, and more, and can be specified globally or per host.

Here is an example configuration:

```
<?php
    // this block should be a peer of 'p4'
    'http_client_options' => array(
        'timeout'       => 10, // default value is 10 seconds

        // path to the SSL certificate directory
        'sslcapath'     => '',

        // the path to a PEM-encoded SSL certificate
        'sslcert'       => '',
```

```
        // the path to Certificate Authority file
        'sslcafile'       => '',

        // the passphrase for the SSL certificate file
        'sslpassphrase' => '',

        // optional, per-host overrides;
        // host as key, array of options as value
        'hosts'     => array(
            'jira.example.com'  => array(
                'sslcapath'     => '/path/to/certs',
                'sslcert'       => 'jira.pem',
                'sslpassphrase' => 'keep my JIRA secure',
                'timeout'       => 15,
            ),
            'jenkins.example.com' => array(
                'sslcafile'       => '/path/to/certs/jenkins.pem',
                'sslpassphrase'   => 'keep my jenkins very secure',
                'timeout'         => 15,
            ),
        ),
    ),
```

See the Laminas Framework Socket Adapter documentation for more information.

Note
Swarm supports the Laminas component versions in the **`LICENSE.txt`** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **`LICENSE.txt`** file is in the **`readme`** folder of your Swarm installation.

Warning
While it is possible to use a self-signed SSL certificate, adding the configuration to do so disables certificate validity checks, making connections to the configured host less secure. **We strongly recommend against using this configuration option.**

However, if you need to configure continuous integration, deployment, or JIRA connections and those connections must use a self-signed SSL certificate, set the **`sslallowselfsigned`** item to **`true`** for the specific host that needs it, as in the following example:

```
<?php
    // this block should be a peer of 'p4'
    'http_client_options' => array(
        'hosts'     => array(
            'jira.example.com'  => array(
                'sslallowselfsigned'  => true,
            ),
        ),
    ),
```

Strict HTTPS

To improve the security when users work with Swarm, particularly if they need to do so outside of your network, Swarm provides a mechanism that tries to force web browsers to use HTTPS. When enabled, Swarm's behavior changes in the following ways:

- HTTP requests to Swarm include a meta-refresh to the HTTPS version. If a load balancer handles encryption before requests reach Swarm, the meta-refresh should be disabled. See below.
- A strict transport security header is included for all requests, which pins the browser to using HTTPS for your Swarm installation for 30 days.
- All qualified URLs that Swarm produces use HTTPS for the scheme.
- Cookies are flagged as HTTPS-only.

Here is an example of how to enable strict HTTPS:

```
<?php
    // this block should be a peer of 'p4'
    'security' => array(
        'https_strict'          => true,
        'https_strict_redirect' => true, // optional; set false to avoid
meta-refresh
        'https_port'            => null, // optional; specify if HTTPS is
                                         // configured on a non-standard
port
    ),
```

When the **`https_strict_redirect`** item is set to **`false`**, Swarm does not add a meta-refresh for HTTP clients. This prevents an endless redirect when a load balancer in front of Swarm applies HTTPS to the client-to-load balancer connection, but not the load balancer-to-Swarm connection.

Apache security

There are several Apache configuration changes that can improve security for Swarm:

- **Disable identification**

  By default, each Apache response to a web request includes a list of tokens identifying Apache and its version, along with any installed modules and their versions. Also, Apache can add a signature line to each response it generates that includes similar information. By itself, this identification information is not a security risk, but it helps would-be attackers select attacks that could be successful.

  To disable Apache identification, add the following two lines to your Apache configuration:

  ```
  ServerSignature Off
  ServerTokens ProductOnly
  ```

- **Disable TRACE requests**

  `TRACE` requests cause Apache to respond with all of the information it has received, which is useful in a debugging environment. `TRACE` can be tricked into divulging cookie information, which could compromise the credentials being used to login to Swarm.

  To disable `TRACE` requests, add the following line to your Apache configuration:

  ```
  TraceEnable off
  ```

- **Update SSL configuration**

  Swarm works correctly with an SSL-enabled Apache. Several attacks on common SSL configurations have been published recently. We recommend that you update your Apache configuration with the following lines:

  ```
  <IfModule mod_ssl.c>
      SSLHonorCipherOrder On
      SSLCipherSuite ECDHE-RSA-AES128-SHA256:AES128-GCM-
  SHA256:RC4:HIGH:!MD5:!aNULL:!EDH
      SSLCompression Off
  </IfModule>
  ```

PHP security

There are several PHP configuration changes that can improve security for Swarm:

- **Disable identification**

  By default, PHP provides information to Apache that identifies that it is participating in a web request, including its version.

  To disable PHP identification, edit your system's php.ini file and change the line setting **`expose_php`** to:

  ```
  expose_php = Off
  ```

- **Remove scripts containing phpinfo()**

  During module development or other debugging, you may need to call **`phpinfo()`**, which displays PHP's active configuration, compilation details, included modules and their configuration. Typically, you would add a script to Swarm's public directory containing:

  ```
  <?php phpinfo() ?>
  ```

  Any such scripts should be removed from a production instance of Swarm.

proxy_mode

By default, Swarm works in proxy mode, passing the end-user's browser IP address to the Helix server. The Helix server checks the IP address against its IP protection table to work out what permissions the end-user has.

Set the Swarm **`proxy_mode`** in the **`p4`** configuration block of the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    'p4' => array(
        'proxy_mode' => true, // defaults to true
    ),
```

- **`true`**: Swarm passes the end-user's browser IP address to the Helix server allowing the Helix server to work out the what permissions the end-user has. This is the default setting.
- **`false`**: the Swarm server IP address is passed to the Helix server. The Helix server checks the Swarm IP address against its protections table to work out the permissions the end-user has. The result is that all of the Swarm users will have the same permissions.

> **Tip**
> If Swarm is connected to multiple-Helix server instances, you have the following options for **`proxy_mode`**:
>
> - To apply a global **`proxy_mode`** setting to all Helix server instances that Swarm is connected to, set **`proxy_mode`** in the **`p4`** block as shown above.

- To apply an individual **proxy_mode** setting to a Helix server instance, set **proxy_mode** in the **serverid** block for that server. This value overrides the global setting in the **p4** block for the Helix server instance.

For more information about configuring Swarm for multiple-Helix server instances, see "Set up the Swarm configuration file for the Helix servers" on page 520.

Short links

Short links work with your Swarm installation's current hostname, but you have the option of registering/configuring an even shorter hostname to make shareable file/directory links as short as possible.

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

1. Register a short domain name, or if you control your own DNS server, a short domain name for your network.
2. Point the short domain name at your Swarm host.
3. Edit the *SWARM_ROOT*`/data/config.php` file and add the following configuration block:

```
<?php
    // this block should be a peer of 'p4'
    'short_links' => array(
        'hostname'  => 'myho.st',
    ),
```

Replace `myho.st` with the short domain name you registered/configured.

If your Swarm is configured to use HTTPS, a custom port, a sub-folder, or any combination of these custom installation options, the short links configuration block should look like:

```
<?php
    // this block should be a peer of 'p4'
    'short_links' => array(
        'external_url' => 'https://myho.st:port/sub-folder',
    ),
```

Replace *myho.st* with the short domain name you have registered/configured.

If you have not configured Swarm to use HTTPS, replace `https://` with `http://`.

If you have configured Swarm to run on a custom port, replace `:port` with the correct custom port. Otherwise, remove `:port`.

If you have configured Swarm to run in a sub-folder, replace `/sub-folder` with the correct sub-folder name. Otherwise, remove `/sub-folder`.

Important
The **external_url** configuration item is only honored if you have also configured the "external_url" on page 497 item within the "Environment" on page 495 configuration item as well. Otherwise, Swarm could generate short links that cannot correctly link to their corresponding full URLs.

When **external_url** is configured, the **hostname** configuration item is ignored.

Single Sign-On PHP configuration

Tip
If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Helix Authentication Service

The Helix Authentication Service acts as the SAML Identity Provider (IdP) for Swarm's Service Provider (SP) and enables authentication with your external IdP using the protocol the administrator has configured in the Helix Authentication Service.

This section describes how to configure SAML 2.0 in the Swarm **config.php** file to enable Swarm to authenticate with a Helix server that is configured for the Helix Authentication Service. The SAML PHP configuration block in the *SWARM_ROOT*/**data/config.php** file configures the Helix Authentication Service and Service Provider (SP) connection details for Swarm. This enables Swarm to connect to your Helix Authentication Service and your Helix Authentication Service to connect to Swarm so that you can log in to Swarm using the IdP log in process.

Configuring Swarm for Helix Authentication Service:

- For an overview of the Helix Authentication Service, see Overview of Helix Authentication Service chapter in the *Helix Authentication Service Administrator Guide*.
- For instructions on configuring the Helix Authentication Service, see the Configuring Helix Authentication Service chapter in the *Helix Authentication Service Administrator Guide*.
- For instructions on configuring Swarm to use the Helix Authentication Service, see "Swarm SAML 2.0 settings" on the facing page.

- For a more in-depth explanation of Swarm configuration for the Helix Authentication Service, see the Example Helix Swarm configuration chapter in the *Helix Authentication Service Administrator Guide*.

Swarm SAML 2.0 settings

This section describes the minimum settings that you must enter to enable Swarm to connect to a Helix server that is enabled for Helix Authentication Service. The **saml** configuration block must be added to the end of the *SWARM_ROOT*/**data/config.php** file as shown in the following example.

> **Tip**
> The exact content of your **saml** configuration block depends on your SAML configuration, you can add other configurables to the SAML PHP block if they are required by your Helix server SAML configuration. For example the **x509cert** and **privateKey** for your SP (Swarm) might be in the certs folder so you would not need to have them specified in the **sp** part of your **saml** block. It is important to note however that the idp **x509cert** must always be in the **idp** part of your **saml** block.
>
> For an overview of SAML 2.0 , see https://github.com/onelogin/php-saml.

Example SAML PHP configuration, follow the underlined links for more information about the configurables:

> **Warning**
> While the syntax of this example is correct, it includes configuration values that **cannot work**. Ensure that you adjust the configuration appropriately for your SAML configuration before using the **saml** block in testing or production.

```
<?php
    // the saml block should be a peer of 'p4' located at the end of
    // the Swarm configuration block in the config.php file
    'saml' => array(
        // If your Helix Server trigger expects a message header so that
it can
        // easily recognize SAML response messages, add the header text
        'header' => 'saml-response: ', // leave empty for no message
header ''
        // Service Provider Data that we are deploying
        'sp' => array(
            // Identifier of the SP entity  (must be a URI)
            'entityId' => '<urn:my_swarm:sp>',
            // Specifies info about where and how the AuthnResponse
```

```
message MUST be
            // returned to the requester, in this case our SP.
            'assertionConsumerService' => array(
                // URL Location where the Response from the IdP will be
returned, this is the Swarm URL and port
                'url' => '<[http[s]://]<swarm-host>[:<port>]>',
            ),

            // Usually x509cert and privateKey of the SP (Swarm) are
provided by files placed in
            // the certs folder. These files must be named sp.crt and
sp.key.
            // Optional: you can also provide them with the following
parameters
            'x509cert' => '<my_sp_swarm_full_cert_string_including_the_
BEGIN_CERTIFICATE_and_END_CERTIFICATE_parts>',
            'privateKey' => '<my_sp_swarm_private_key>',
        ),
        // Identity Provider Data that we want to connect to with our SP
(Swarm)
        'idp' => array(
            // Identifier of the IdP entity  (must be a URI)
            'entityId' => '<my_entityid_provided_by_the_idp>',
            // SSO endpoint info of the IdP. (Authentication Request
protocol)
            'singleSignOnService' => array(
                // URL Target of the IdP where the SP (Swarm) will send
the Authentication Request Message
                'url' => '<full_idp_URL_path_to_send_authentication_
request_message_to>',
            ),

            // The x509cert of the idp is provided by the following
x509cert parameter.
            // Do not add the privateKey parameter.
```

```
            // You must use the x509cert parameter, you must not add the
cert file the certs folder.
            'x509cert' => '<my_idp_full_cert_string_including_the_BEGIN_
CERTIFICATE_and_END_CERTIFICATE_parts>',
        ),
    ),
```

Note
The Swarm configuration file does not include PHP's standard closing tag (?>). This is intentional as it prevents unintentional whitespace from being introduced into Swarm's output, which would interfere with Swarm's ability to control HTTP headers. Debugging problems that result from unintentional whitespace can be challenging, since the resulting behavior and error messages often appear to be misleading.

header

Some Helix server triggers need a header (prefix) added to SAML response messages so that the Helix server can easily identify the messages. The header is set in the **`header`** value.

If a header is not required, set **`header`** to empty `''`.

```
<?php
    'saml' => array(
        // If your Helix Server trigger expects a message header so that
it can
        // easily recognize SAML response messages, add the header text
        'header' => 'saml-response: ', // leave empty for no message
header ''
    ),
```

The default value is **`'saml-response: '`**.

sp

The **`sp`** (service provider) section defines the callback destination and identifier information for your IdP (Identity Provider). This tells your IdP how to connect to Swarm.

- **`entityId`**: this is the identifier your IdP knows your Swarm as. Set your Entity ID here and then use the same value in your IdP's configuration tool. When Swarm connects to your IdP the Entity ID is used to verify your connection. This must be a URI.

- **`assertionConsumerService`** **`url`**: enter your Swarm URL and port number. This sets your Swarm instance as the URL your IdP sends responses to. If you don't enter a port number, port 80 is used.

  > **Important**
  > Do not use **`localhost`** for the **`url`**.

- **`x509cert`** and **`privateKey`**: enter your Swarm instance security connection details.

```
<?php
    'saml' => array(
        // Service Provider Data that we are deploying
        'sp' => array(
            // Identifier of the SP entity  (must be a URI)
            'entityId' => '<urn:my_swarm:sp>',
            // Specifies info about where and how the AuthnResponse
message MUST be
            // returned to the requester, in this case our SP.
            'assertionConsumerService' => array(
                // URL Location where the Response from the IdP will be
returned, this is the Swarm URL and port
                'url' => '<[http[s]://]<swarm-host>[:<port>]>',
            ),

            // Usually x509cert and privateKey of the SP (Swarm) are
provided by files placed in
            // the certs folder. These files must be named sp.crt and
sp.key.
            // Optional: you can also provide them with the following
parameters
            'x509cert' => '<my_sp_swarm_full_cert_string_including_the_
BEGIN_CERTIFICATE_and_END_CERTIFICATE_parts>',
            'privateKey' => '<my_sp_swarm_private_key>',
        ),
    ),
```

idp

The Helix Authentication Service acts as the SAML Identity Provider (IdP) for Swarm's Service Provider (SP) and enables authentication with your external IdP using the protocol the administrator has chosen for the Helix Authentication Service. The **`idp`** (identity provider) section defines the IdP connection and security information, this tells Swarm how to connect to your Helix Authentication Service.

- **`entityId`**: enter your Entity ID, this is configured in your Helix Authentication Service. This enables Swarm to connect to the Helix Authentication Service. This must be a URI.
- **`singleSignOnService`** **`url`**: enter the URL Swarm sends authentication requests to. This is configured in your Helix Authentication Service.
- **`x509cert`**: enter your Helix Authentication Service security connection cert.

```
<?php
    'saml' => array(
        //  Identity Provider Data that we want to connect to with our SP
(Swarm)
        'idp' => array(
            // Identifier of the IdP entity  (must be a URI)
            'entityId' => '<my_entityid_provided_by_the_idp>',
            // SSO endpoint info of the IdP. (Authentication Request
protocol)
            'singleSignOnService' => array(
                // URL Target of the IdP where the SP (Swarm) will send
the Authentication Request Message
                'url' => '<full_idp_URL_path_to_send_authentication_
request_message_to>',
            ),

            // The x509cert of the idp is provided by the following
x509cert parameter.
            // Do not add the privateKey parameter.
            // You must use the x509cert parameter, you must not add the
cert file the certs folder.
            'x509cert' => '<my_idp_full_cert_string_including_the_BEGIN_
CERTIFICATE_and_END_CERTIFICATE_parts>',
        ),
    ),
```

swarm_root

`swarm_root` refers to the directory where Swarm lives in the filesystem. Depending on how you installed Swarm, the default location could be:

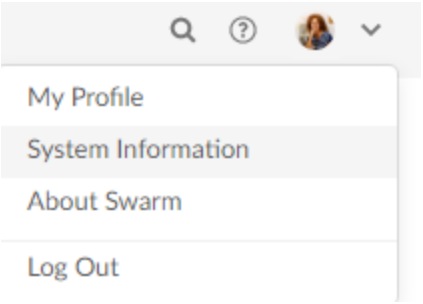
- For a package or OVA installation: `/opt/perforce/swarm`.
- For a tarball installation: wherever you unpacked Swarm. If you are unsure, you could check your web server's configuration to discover where Swarm exists.

System Information

> **Note**
> The system information page is available to users with *admin* or *super* privileges.

The system information page provides details that can be useful to Perforce support engineers when you ask for assistance.

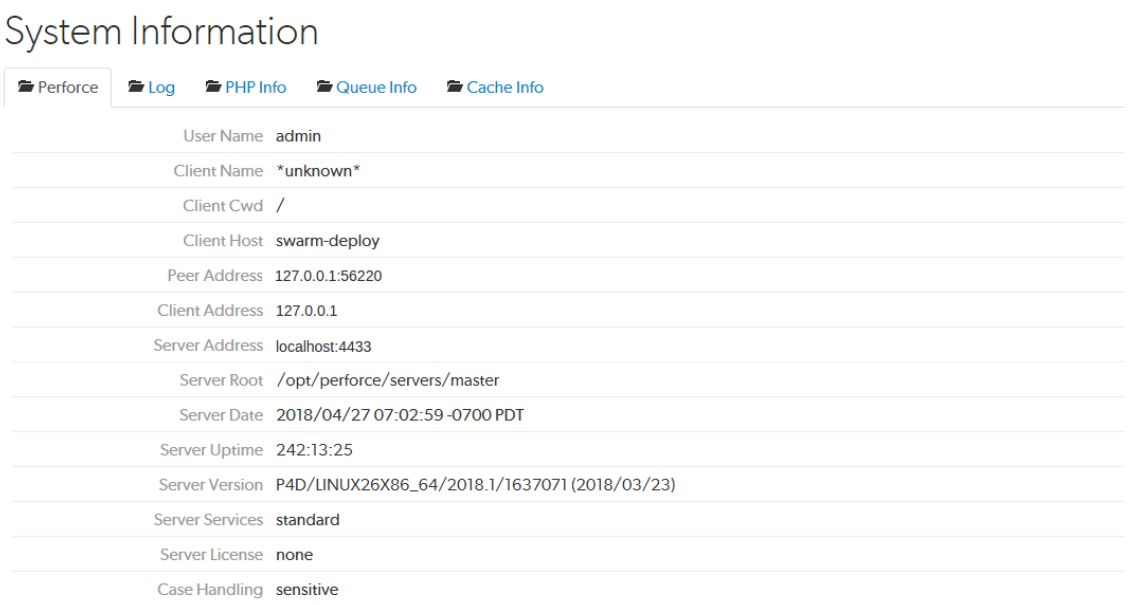
To display the system information page, click **System Information** from the **User id** dropdown menu:



In this section:

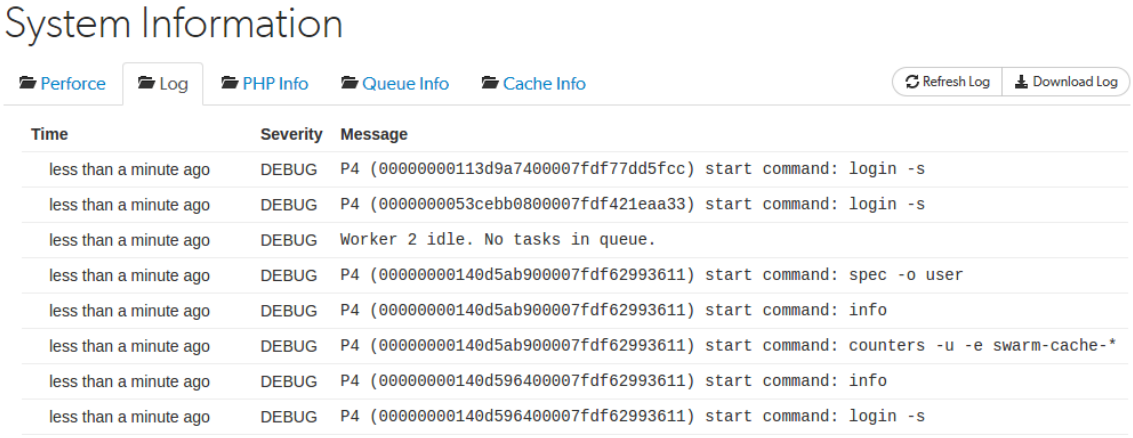
Perforce

The **Perforce** tab provides information similar to the `p4 info` command.



Log

Click the **Log** tab to display the most recent entries, up to 1 megabyte, in Swarm's log file, which resides in *SWARM_ROOT*`/data/log`. Review the logging levels for Swarm logs to ensure that the entries you want to see are included.



If your log entries include a critical error, an arrow appears:

```
> 14 minutes ago    CRIT  exception 'P4\Connection\Exception\CommandException' with message 'Command failed: Partner
                          exited unexpectedly.' in /web/apps/swarm/library/P4/Connection/AbstractConnection.php:986
```

Click the error to display the stack trace that accompanies the error:

```
15 minutes ago    CRIT  exception 'P4\Connection\Exception\CommandException' with message 'Command failed: Partner
                        exited unexpectedly.' in /web/apps/swarm/library/P4/Connection/AbstractConnection.php:986

Stack trace:
#0 library/P4/Connection/AbstractConnection.php(711): P4\Connection\AbstractConnection->handleError(Object(P4\Connection\CommandResult))
#1 library/Record/Cache/Cache.php(240): P4\Connection\AbstractConnection->run('counters', Array)
#2 library/Record/Cache/Cache.php(221): Record\Cache\Cache->getCounters()
#3 library/Record/Cache/Cache.php(200): Record\Cache\Cache->getCounter('groups')
#4 module/Application/Module.php(168): Record\Cache\Cache->removeInvalidatedFiles()
#5 [internal function]: Application\Module->Application\{closure}(Object(Zend\EventManager\Event))
```

Log tab buttons:

- **Refresh Log** button: click to load the latest log entries, up to 1 megabyte.
- **Download Log** button: click to download the log and all of its log entries.

If your log entries include a critical error, an arrow appears:

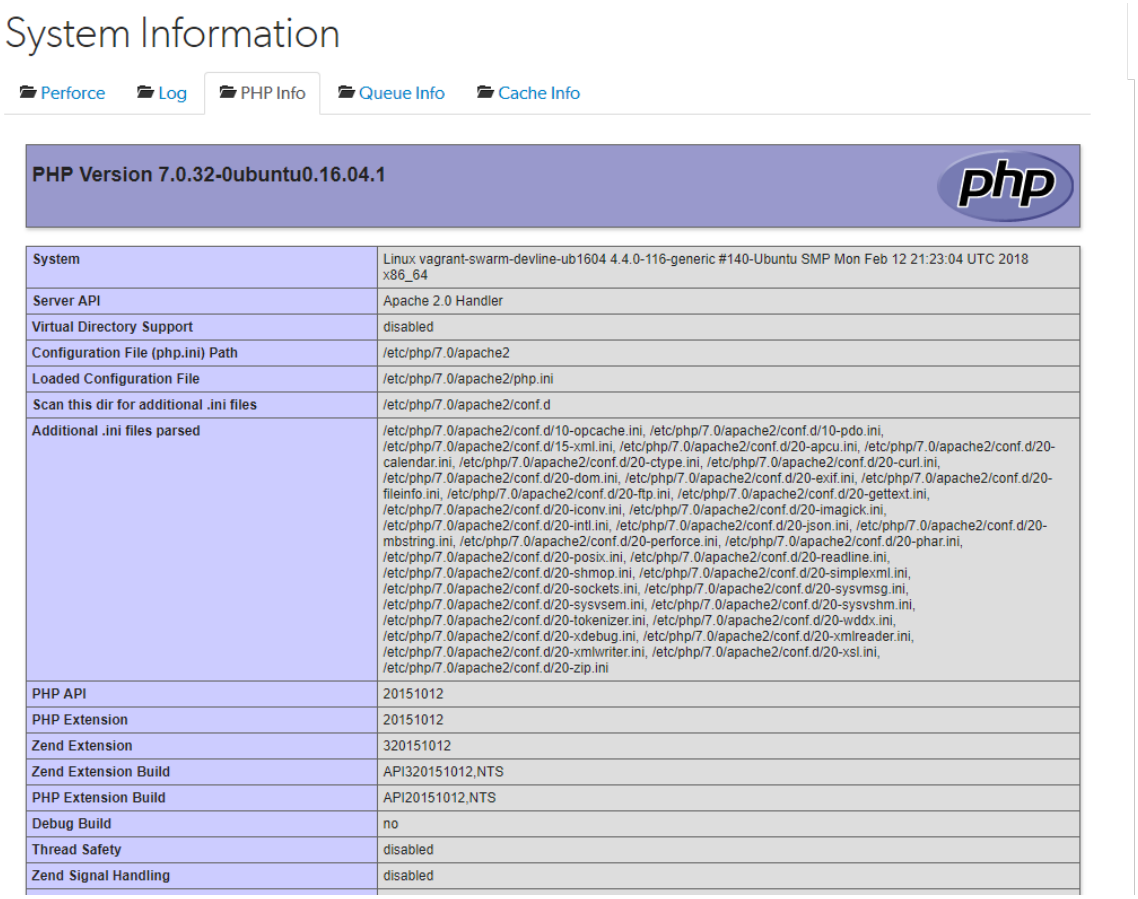
```
14 minutes ago    CRIT  exception 'P4\Connection\Exception\CommandException' with message 'Command failed: Partner
                        exited unexpectedly.' in /web/apps/swarm/library/P4/Connection/AbstractConnection.php:986
```

Click the error to display the stack trace that accompanies the error:

```
15 minutes ago    CRIT  exception 'P4\Connection\Exception\CommandException' with message 'Command failed: Partner
                        exited unexpectedly.' in /web/apps/swarm/library/P4/Connection/AbstractConnection.php:986

Stack trace:
#0 library/P4/Connection/AbstractConnection.php(711): P4\Connection\AbstractConnection->handleError(Object(P4\Connection\CommandResult))
#1 library/Record/Cache/Cache.php(240): P4\Connection\AbstractConnection->run('counters', Array)
#2 library/Record/Cache/Cache.php(221): Record\Cache\Cache->getCounters()
#3 library/Record/Cache/Cache.php(200): Record\Cache\Cache->getCounter('groups')
#4 module/Application/Module.php(168): Record\Cache\Cache->removeInvalidatedFiles()
#5 [internal function]: Application\Module->Application\{closure}(Object(Zend\EventManager\Event))
```

PHP Info

Click the **PHP Info** tab to display PHP's own information display generated by executing `phpinfo()`, PHP's internal diagnostic display.

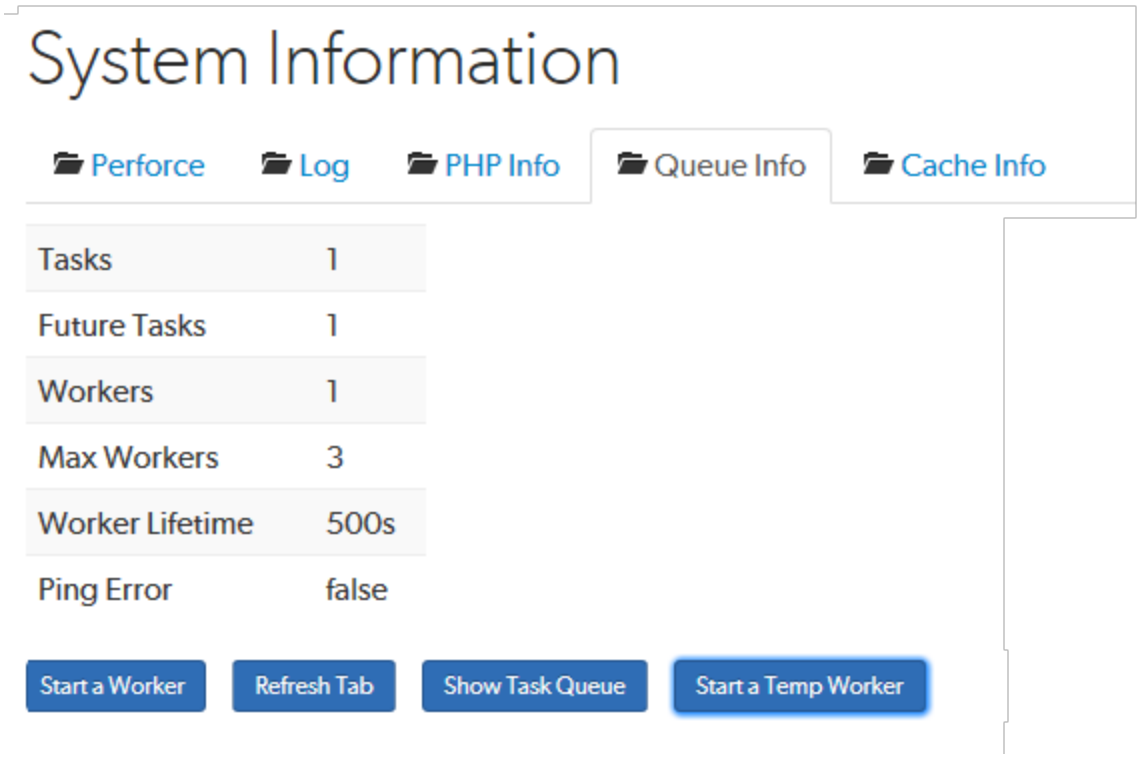


System	Linux vagrant-swarm-devline-ub1604 4.4.0-116-generic #140-Ubuntu SMP Mon Feb 12 21:23:04 UTC 2018 x86_64
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/apache2
Loaded Configuration File	/etc/php/7.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/apache2/conf.d
Additional .ini files parsed	/etc/php/7.0/apache2/conf.d/10-opcache.ini, /etc/php/7.0/apache2/conf.d/10-pdo.ini, /etc/php/7.0/apache2/conf.d/15-xml.ini, /etc/php/7.0/apache2/conf.d/20-apcu.ini, /etc/php/7.0/apache2/conf.d/20-calendar.ini, /etc/php/7.0/apache2/conf.d/20-ctype.ini, /etc/php/7.0/apache2/conf.d/20-curl.ini, /etc/php/7.0/apache2/conf.d/20-dom.ini, /etc/php/7.0/apache2/conf.d/20-exif.ini, /etc/php/7.0/apache2/conf.d/20-fileinfo.ini, /etc/php/7.0/apache2/conf.d/20-ftp.ini, /etc/php/7.0/apache2/conf.d/20-gettext.ini, /etc/php/7.0/apache2/conf.d/20-iconv.ini, /etc/php/7.0/apache2/conf.d/20-imagick.ini, /etc/php/7.0/apache2/conf.d/20-intl.ini, /etc/php/7.0/apache2/conf.d/20-json.ini, /etc/php/7.0/apache2/conf.d/20-mbstring.ini, /etc/php/7.0/apache2/conf.d/20-perforce.ini, /etc/php/7.0/apache2/conf.d/20-phar.ini, /etc/php/7.0/apache2/conf.d/20-posix.ini, /etc/php/7.0/apache2/conf.d/20-readline.ini, /etc/php/7.0/apache2/conf.d/20-shmop.ini, /etc/php/7.0/apache2/conf.d/20-simplexml.ini, /etc/php/7.0/apache2/conf.d/20-sockets.ini, /etc/php/7.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.0/apache2/conf.d/20-sysvsem.ini, /etc/php/7.0/apache2/conf.d/20-sysvshm.ini, /etc/php/7.0/apache2/conf.d/20-tokenizer.ini, /etc/php/7.0/apache2/conf.d/20-wddx.ini, /etc/php/7.0/apache2/conf.d/20-xdebug.ini, /etc/php/7.0/apache2/conf.d/20-xmlreader.ini, /etc/php/7.0/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.0/apache2/conf.d/20-xsl.ini, /etc/php/7.0/apache2/conf.d/20-zip.ini
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled

Queue Info

Click the **Queue Info** tab to display the Swarm task queue information.

> **Note**
> Workers should be started automatically using a recurring task. For instructions on how to set up a recurring task to spawn workers automatically, see "Set up a recurring task to spawn workers" on page 174.



- **Tasks:** displays the number of tasks in the queue.
- **Future Tasks:** displays the number of future tasks in the queue. For example, when a user is making comments, the comment notification is delayed by 30 minutes by default. This is a future task.
- **Workers:** displays the number of workers available to service tasks.

  Tip
  If the number of Workers is 0 (zero):

  - Your workers are not being created automatically. For instructions on how to set up a recurring task to spawn workers automatically, see "Set up a recurring task to spawn workers" on page 174.
  - To process the current task queue while you get your recurring tasks working again, click **Start a Temp Worker**. This is a quick way to get Swarm running again while you fix the problem.

- **Max Workers:** displays the maximum number of workers that are allowed. The default is three.
- **Worker Lifetime:** displays the worker lifetime that is configured. The default is 595 seconds (10 minutes less 5 seconds).
- **Ping Error:** not used.

Queue Info buttons:

- **Start a Worker** button: click to manually start a new Swarm worker. The new worker will only last for the configured Worker Lifetime. If the number of workers running matches the configured limit, the new worker is not started.

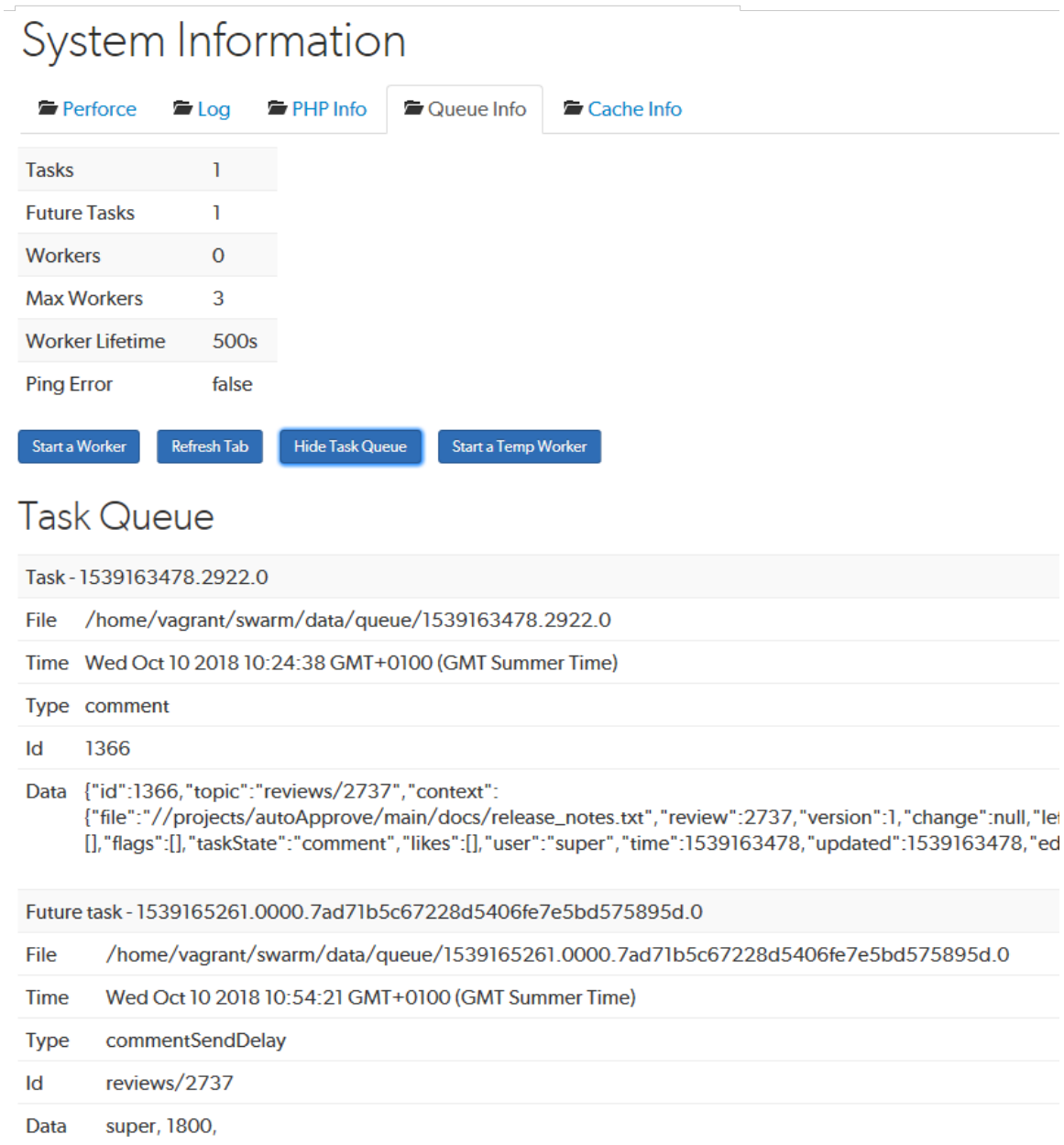
  > **Tip**
  > If your recurring task has stopped working and you need to process the task queue while you get it working again, start a temporary worker by clicking **Start a Temp Worker**. This is a quick way to get Swarm running again while you fix the problem.

- **Refresh Tab** button: click to refresh the queue information in the tab.
- **Show Task Queue** button: click to display the Task Queue information. When the task queue is displayed the button changes to **Hide Task Queue**. The task queue shows more information about the tasks and future tasks in the queue.
- **Start a Temp Worker** button: click to manually start a Swarm temporary worker. Typically you use a temporary worker to process the current tasks in the queue while you fix the recurring task that automatically spawns the workers.

  This is a special Swarm worker that is not subject to the worker lifetime setting and will not stop until it has processed all of the tasks in the current queue or one of the tasks fails.

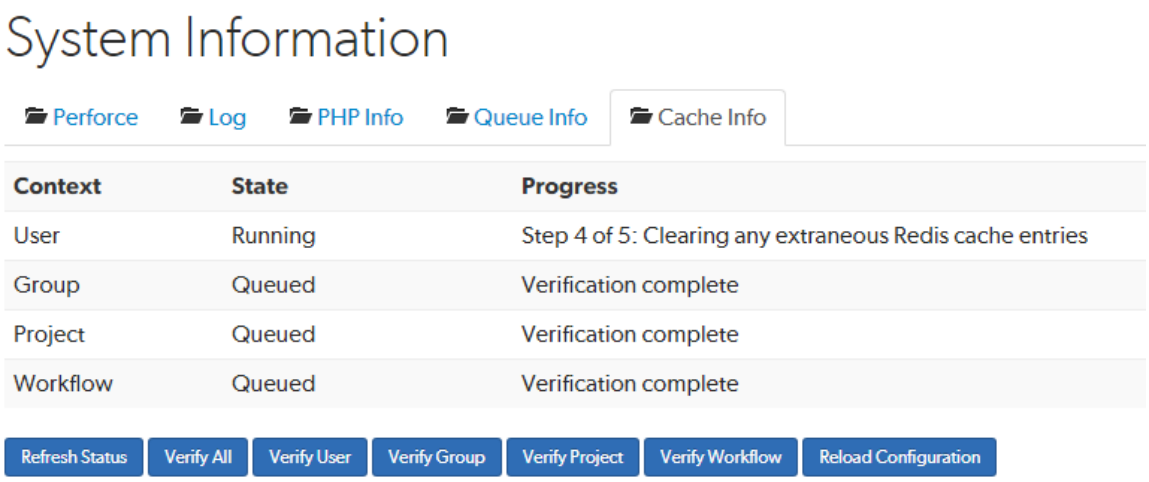
Task Queue

Click the **Show Task Queue** button to display the tasks in the Swarm queue.



Cache Info

Click the **Cache Info** tab to display the Swarm cache information.



Tip
By default, Swarm runs a regular verification check at 03:00 every day to make sure that the Redis caches and Helix server are in sync, see "Swarm connection to Redis " on page 540.

The **Cache Info** tab displays the cache verification state and progress for each of the Redis caches:

- Valid states for verify are; **`Not Queued`**, **`Queued`**, **`Running`**, and **`Failed`**.
- Valid states for progress are, **`Step x of y;`** (where **`x`** is the current step and **`y`** is the total number of steps) followed by:
  - **`Getting Redis cache entries`**
  - **`Calculating checksums for Redis keys`**
  - **`Calculating checksums for Perforce models`**
  - **`Clearing any extraneous Redis cache entries`**
  - **`Indexing any missing Perforce models into the Redis cache`**
  - **`Verification complete`**

Refresh Status button

To refresh the cache information on the tab, click **Refresh Status**.

Verify buttons

Important
On Swarm systems with a large number of users, groups, and projects verification of the caches can take some time and can impact performance.

> In normal operation, the Redis cache and Helix Core server data should stay in sync and there is no need to verify the caches.
>
> If you have a large number of users, groups, and projects, we recommend that you verify the individual user and group caches rather than using **Verify All**.

In normal operation the Redis cache and Helix Core server data should stay in sync and there is no need to verify the caches. However, in some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server.

If you suspect that you have a cache issue, the verify buttons enable you to manually verify the caches. Before using the **Verify All** button, see the Important note above. If verification finds the cache is out of sync with the Helix server, Swarm automatically updates the data in that cache.

For example, to verify the User cache:

1. Navigate to the **Cache Info** tab on the Swarm **System Information** page.
2. Click **Verify User**.

   Swarm responds with a message to say that it has successfully verified the user cache.

Reload Configuration button

The Swarm configuration is stored in the `SWARM_ROOT/data/config.php` file. If you make a configuration change, Swarm will not use it until the configuration cache has been deleted and reloaded, this forces Swarm to use the new configuration.

To delete and reload the Swarm configuration cache:

1. Navigate to the **Cache Info** tab on the Swarm **System Information** page.
2. Click **Reload Configuration**.

   Swarm responds with a message to say that it has successfully reloaded the config.

   Swarm is now using the updated Swarm `config.php` file.

Test definitions

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" above.

project_and_branch_separator

> **Note**
> The **`project_and_branch_separator`** configurable character is only applied to tests created from the **Tests** menu, see "Tests" on page 440.

Specifies the character used to separate the **`{projects}`** and **`{branches}`** IDs in the URL that is passed to your CI system. Some CI systems, such as Jenkins, escape the default colon **`:`** character resulting in a failed test request. If your CI system needs a different separator character, the **`project_and_branch_separator`** character can be changed to a character that you CI system will accept, for example a dash **`-`** or an underscore **`_`**.

Be careful when selecting your separator character, some characters can cause issues when calling tests. For example, a **#** character in a URL call means the branch is treated as an anchor and a **%** character in a JSON body encoding is escaped. Both of these will result in a failed test request.

Configure the **`project_and_branch_separator`** character with the following configuration block in the **`SWARM_ROOT/data/config.php`** file:

```
    'test_definitions' => array(
        'project_and_branch_separator' => ':',
    ),
```

The default value for the **`project_and_branch_separator`** configurable is a colon **`:`**.

Trigger options

The Swarm trigger script, **`swarm-trigger.pl`**, provides the following command line options and configuration items.

Command-line options

Synopsis

```
swarm-trigger.pl -t type -v ID [-c config_file]
```

```
swarm-trigger.pl -o
```

```
swarm-trigger.pl -h
```

Informational options

The following options perform no processing, and simply provide information useful to a Swarm administrator. These are not intended to be used within trigger entries in the Helix server.

- **-h**

  Displays a list of the available options, some guidance on its usage, and a copy of the trigger table entries that should be configured in the Helix server to execute the script in its current path.

- **-o**

  Displays a copy of the trigger table entries that should be configured in the Helix server to execute the script in its current path.

Operational options

The following command-line options are used in the trigger table entries in the Helix server to specify how the Swarm trigger script should be executed.

- **`-t type`**

  Specifies the type of processing that the trigger script undertakes. ***`type`*** can be one of:

  - **`job`**: this type should be used with Helix server **`form-commit`** events for job forms, and informs Swarm when a job is created or modified.
  - **`user`**: this type should be used with Helix server **`form-commit`** events for user forms, which informs Swarm when a user is added or modified.
  - **`userdel`**: this type should be used with Helix server **`form-delete`** events for user forms, which informs Swarm when a user is deleted.
  - **`group`**: this type should be used with Helix server **`form-commit`** events for group forms, and informs Swarm when a group is created or modified.
  - **`groupdel`**: this type should be used with Helix server **`form-delete`** events for group forms, and informs Swarm when a group is deleted.
  - **`changesave`**: this type should be used with Helix server **`form-save`** events for change forms, and informs Swarm when a changelist is created or modified.
  - **`shelve`**: this type should be used with Helix server **`shelve-commit`** events, and informs Swarm when a changelist is shelved, which can create or update a review.
  - **`commit`**: this type should be used with Helix server **`change-commit`** events, and informs Swarm when a changelist is committed.
  - **`shelvedel`**: this type is used with Helix server **`shelve-delete`** events, and informs Swarm when a file is deleted from a shelf.

> **Important**
> Do not use the Swarm user that is configured in the Swarm configuration file when deleting shelves, or deleting files from shelves. The Swarm logic processes the *shelve-delete* trigger event, if the event is invoked by the Swarm user it is rejected. The delete operations will fail.

- `enforce` comment out if the Workflow feature is enabled (enabled by default): this type is used to verify that commits to specific depot paths are associated with approved reviews. If a commit includes a file within the specified depot path, and it is not associated with a review (or a review that is not approved), the commit is rejected.

  Using the enforce type prevents users from committing changes to specific depot paths without those changes being reviewed and approved.

- `strict` comment out if the Workflow feature is enabled (enabled by default): this type is used to verify that the file content in a commit matches the file content of its associated approved review. If one or more files in a commit do not match the content of the file in its associated review, the commit is rejected.

  Using the `strict` type prevents users from making changes to the file content after a review has been approved and then submitting the unapproved changes.

  > **Note**
  > `strict` implies `enforce`.

- **Workflow trigger types** comment out if the Workflow feature is disabled (enabled by default). The `enforce` and `strict` triggers above must be commented out if the Workflow feature is enabled:

Important
Known limitations

The workflow triggers do not support the following trigger functionality available in Swarm 2018.1 and earlier:

- **`EXEMPT_FILE_COUNT`**: When set to a positive integer, commits with a file count greater or equal to this value are exempt from **`enforce`** or **`strict`** verifications.
- **`EXEMPT_EXTENSIONS`**: A comma separated list of file extension. Commits with files having only these extensions are exempt from **`enforce`** or **`strict`** verifications.

To continue to use this functionality, you must keep your existing **`enforce`** and **`strict`** triggers and disable the Workflow feature.

Tip
By default, all group members and users must follow any workflow rules that apply to changelists and reviews. However, your Swarm administrator can configure specific groups and users so that they are excluded from following the workflow rules. This applies to actions taken in the Swarm UI, the P4D command line, a P4D client, and the Swarm API.

The following trigger types are used by Swarm for workflow rules. They are not intended for use in custom trigger scripts.

- `checkenforced`: this type is used by the Swarm **On commit with a review** workflow rule and cannot be configured. `checkenforced` triggers on a Helix server `change-submit` event.

  If a commit is made to a branch that has **On commit with a review** set to **Reject unless approved**, and it is not associated with a review (or the review is not approved), the commit is rejected. If the commit is associated with an approved review, a content check is carried out by `checkstrict`.

- `checkstrict`: this type is used by the Swarm **On commit without a review** workflow rule and cannot be configured. `checkstrict` triggers on a Helix server `change-content` event.

  This check is performed after `checkenforced`. If the **On commit without a review** rule is set to **Reject**, and one or more files in a commit do not match the content of the file in its associated review, the commit is rejected.

- `checkshelve`: this type is used by the Swarm **On update of a review in an end state** workflow rule and cannot be configured. `checkshelve` triggers on a Helix server `shelve-submit` event.

  If a shelf is associated with a review, and it is committed to a branch that has **On update of a review in an end state** set to **Reject**, Swarm checks what state the review is in to make sure it is not in a protected state. If the review is in a protected state, the commit is rejected.

> **Important**
> You cannot mix Swarm trigger types with unrelated Helix server events; the behavior is undetermined, and the information required for each type of processing may not be available to the trigger.

- **`-v ID`**

  Specifies *`ID`*, which is the identifier for the current trigger type.

  When the *`type`* is `job`, `user`, `userdel`, `group`, `groupdel`, or `changesave`, *`ID`* should be `%formname%` to specify the specific form identifier Swarm should process.

  When the *`type`* is `shelve`, `commit`, `shelvedel`, `checkenforced`, `checkstrict`, or `checkshelve` *`ID`* should be `%change%` to specify the specific changelist Swarm should process.

- **`-p port` (optional) Do not use if the Workflow feature is enabled (default)**

  Specifies the Helix server port (`P4PORT`). This value is optional, and is only used for types `enforce` or `strict` as the trigger has to run its own commands against the Helix server during its processing.

- **`-r` (optional) Do not use if the Workflow feature is enabled (default)**

  Specifies that, when types **`enforce`** or **`strict`** are being processed, the verifications should only be performed on commits that are currently in review.

- **`-g` (optional) Do not use if the Workflow feature is enabled (default)**

  Specifies a **`group`** to exclude from **`enforce`** or **`strict`** verifications. Members of the group (including sub-groups) are not subject to **`enforce`** or **`strict`** verifications.

- **`-c config_file` (optional)**

  Specifies an optional ***`config_file`*** which is used to specify configuration items.

Configuration items

The following configuration items are used in the **`swarm-trigger.conf`** (or another file, if the **`-c`** option is used in the trigger entries).

- **`SWARM_HOST` (required)**

  Specifies the host URL of your Swarm instance, with the leading **`http://`** or **`https://`**. For example: **`https://myswarm.url`**.

- **`SWARM_TOKEN` (required)**

  A token used when talking to Swarm. To obtain the token, log into Swarm as a user with *super* privileges and select the About Swarm from the user menu in the main navigation bar.

  You can also manually create additional tokens. Tokens are empty files stored within *`SWARM_ROOT`***`/data/queue/tokens`** (the filename is the token and is reported on the **About Swarm** dialog), that should be readable by the web server.

  You might manually create additional tokens to allow other processes to talk to Swarm, such as JIRA build tasks, and to selectively invalidate access to Swarm without interfering with regular Swarm operations.

- **`ADMIN_USER` (optional) Do not use if the Workflow feature is enabled (default)**

  When **`enforce`** or **`strict`** verifications are to be performed, you may need specify a username of a user in the Helix server that has *admin* privileges. If you do not specify a username, the trigger script uses the Helix server user set in the environment.

- **`ADMIN_TICKET_FILE` (optional) Do not use if the Workflow feature is enabled (default)**

  When **`enforce`** or **`strict`** verifications are to be performed, you may need specify the path to the **`.p4tickets`** file, if your Helix server tickets file is not the default **`$HOME/.p4tickets`**.

> **Important**
> Ensure that the ticket belongs to a user with *admin* privileges in the Helix server, and is a member of a group with an `unlimited` or very long ticket timeout. If this user's authentication times out, `enforce` and `strict` verifications stop working.

- **`P4_PORT` (optional) Do not use if the Workflow feature is enabled (default)**

  When `enforce` or `strict` verifications are to be performed, you may need to set the port value (`P4PORT`) of the Helix server, particularly if the Helix server is on a non-standard port, or if the Helix server is not using the default hostname.

- **`P4` (optional) Do not use if the Workflow feature is enabled (default)**

  Specifies the full path to `p4`, the Helix server command-line client. This is only required when `p4` is not found in the `PATH` of the Helix server environment, and when `enforce` or `strict` verifications are to be performed.

- **`EXEMPT_FILE_COUNT` (optional) Do not use if the Workflow feature is enabled (default)**

  When set to a positive integer, commits with a file count greater or equal to this value are exempt from `enforce` or `strict` verifications.

- **`EXEMPT_EXTENSIONS` (optional) Do not use if the Workflow feature is enabled (default)**

  A comma-separated list of file extensions. Commits with files having only these extensions are exempt from `enforce` or `strict` verifications.

- **`VERIFY_SSL` (optional)**

  If HTTPS is used on the Swarm web server, this controls whether the SSL certificate is validated or not.

  By default `VERIFY_SSL` is set to `1`, this means any SSL certificates must be valid. If the Swarm web server is using a self signed certificate, `VERIFY_SSL` must be set to `0`.

- **`TIMEOUT` (optional)**

  Sets the number of seconds to wait before returning an error when communicating with the Swarm server.

  The default setting is `30` seconds.

- **`IGNORE_TIMEOUT` (optional)**

  Configure the action to take if communication with the Swarm server times out. For example, if you are submitting a large number of files to the Helix server.

  Options are:

  - **0** if communication with the Swarm server times out, the submit will fail. This is the default setting.
  - **1** if communication with the Swarm server times out, the submit will continue. This increases the reliability of submits.

- **`IGNORE_NOSERVER` (optional)**

Configure the action to take if communication with the Swarm server fails. For example, the Swarm server is down.

Options are:

- **0** if communication with the Swarm server fails, the submit will fail. This is the default setting.
- **1** if communication with the Swarm server fails, the submit will continue. This increases the reliability of submits.

Unapprove modified reviews

By default, when an approved review is committed or updated, Swarm changes the state to **Needs Review** if the files have been modified since the review was approved.

If one or more files in a review has the filetype +k (`ktext`), this behavior is undesirable because the files will appear to be modified as the Helix server replaces RCS keywords with their current values. This behavior can be disabled.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

To disable this behavior, edit the *SWARM_ROOT*`/data/config.php` file, and add or update the `unapprove_modified` item to `false`, within the reviews configuration block. For example:

```
<?php
    // this block should be a peer of 'p4'
    'reviews' => array(
        'unapprove_modified'  => false,
    ),
```

For information on file types, see File Types in *Helix Core P4 Command Reference*.

Uninstall Swarm

This section covers the steps required to uninstall Swarm.

Background

The bulk of Swarm's metadata (activity, comments, review records, followers) is stored in p4 keys under `swarm-*`. If you are using a 2012.1+ server, Swarm also defines user groups for each project that you define. The names of these groups correspond 1-to-1 with projects, for example `swarm-project-fantastico`. Swarm manages a pool of client workspaces that it uses to shelve and commit files. These clients are named `swarm-{uuid}`, for example `swarm-5ad4a9c0-06e7-20eb-897f-cbd4cc934295`.

Uninstall steps

1. Uninstall the Swarm triggers.
2. Remove your web server's virtual host configuration for Swarm.
3. Restart your web server.
4. Delete groups/clients/keys that are prefixed with `swarm-*`.

   > **Note**
   > The clients could contain shelved files for reviews. Determine how you want to handle those files prior to deleting the clients.

5. Additional indexed information is stored in the database file `db.ixtext`. Unfortunately, indexed jobs and other generic indexed information would be lost if this table was simply removed, and modifying the database file can be a dangerous operation in a number of Helix server deployment scenarios.

   > **Important**
   > Contact Perforce support for assistance if you feel the need to remove Swarm's indexed information: support@perforce.com.

6. Rebuild the job index. The best approach is to run:

   ```
   $ p4 jobs -R
   ```

   which rebuilds the `db.ixtext` table. There are two caveats that likely require discussion with support@perforce.com:

   - If you make use of the unsupported `p4 index` command, you **cannot** use this approach, as it would remove all of your indexes.
   - If you have indexing turned on for the domain table, you must also run:

     ```
     $ p4d -xf index.domain.owner
     ```

7. If the **P4.Swarm.URL** or **P4.Swarm.CommitURL** properties were set (for details, see "Client integration" on page 467 and "Commit-edge deployment" on page 476 respectively), they should be unset to prevent P4V (and potentially other clients) from attempting Swarm operations:

```
$ p4 property -d -n P4.Swarm.URL
$ p4 property -d -n P4.Swarm.CommitURL
```

If the **P4.Swarm.URL** or **P4.Swarm.CommitURL** properties were set using sequence numbers, you need to add the **-s*N*** flag to the commands, where ***N*** is a sequence number.

To discover all of the definitions of the **P4.Swarm.*** properties and their sequence numbers, run the command:

```
$ p4 property -Al | grep P4.Swarm
P4.Swarm.CommitURL = https://myswarm.url/ (any) #none
P4.Swarm.CommitURL = https://myswarm1.url/ (any) #1
P4.Swarm.URL = https://myswarm.url/ (any) #none
P4.Swarm.URL = https://myswarm3.url/ (any) #3
P4.Swarm.URL = https://myswarm2.url/ (any) #2
P4.Swarm.URL = https://myswarm1.url/ (any) #1
```

To delete all of these property definitions, you would run the following commands:

```
$ p4 property -d -n P4.Swarm.URL
$ p4 property -d -n P4.Swarm.URL -s0
$ p4 property -d -n P4.Swarm.URL -s1
$ p4 property -d -n P4.Swarm.URL -s2
$ p4 property -d -n P4.Swarm.URL -s3
$ p4 property -d -n P4.Swarm.CommitURL
$ p4 property -d -n P4.Swarm.CommitURL -s1
```

Upgrade index

The Swarm Upgrade index process can be configured to suit the performance of your Swarm system via the following config items.

> **Note**
> If you are upgrading from Swarm version 2017.3 or later, the index upgrade step is not required.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

status_refresh_interval

The `status_refresh_interval` sets the refresh rate of the index upgrade status page. The default is 10 seconds.

```
<?php
    // this block should be a peer of 'p4'
    'upgrade' => array(
       'status_refresh_interval' => 10, //Refresh page every 10 seconds
    ),
```

batch_size

The `batch_size` sets the number of reviews held in memory and processed for each batch. This value can be increased or decreased depending on your Swarm machines system memory. The default is 1000 reviews.

For example:

If the Swarm machine does not have a lot of memory, for instance 128MB, a `batch_size` of 1000 might be too high and may make the upgrade run very slowly. In this instance reduce the `batch_size` to a more manageable size.

```
<?php
    // this block should be a peer of 'p4'
    'upgrade' => array(
       'batch_size' => 1000,    //Fetch 1000 reviews to lower memory usage
    ),
```

Users

This section describes the configurables that can be set for users.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Dashboard refresh interval

By default, when your dashboard is open, Swarm checks for new content every five minutes. If Swarm finds new content for a user's dashboard, the **Refresh** button is displayed on their dashboard. For more information about using the **Refresh** button, see Refresh button.

Configure the dashboard refresh interval in milliseconds with the following configuration block in the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
        'users' => array(
           'dashboard_refresh_interval' => 300000, // Default 300000
milliseconds
           ),
        ),
```

Default value is `300000` milliseconds (300 seconds, 5 minutes)

Display full names

By default, Swarm displays *userids*. Set the `display_fullname` configurable to `true` to display full user names on the:

- Project Settings page
- Group Settings page
- Activity streams
- Commits page
- Edit reviewers dialog

To display full user names in Swarm, set the following configuration block in the *SWARM_ROOT*`/data/config.php` file to `true`:

```
<?php
    // this block should be a peer of 'p4'
        'users' => array(
           'display_fullname' => true,
```

```
            ),
        ),
```

Default value is **`false`**, *userids* are displayed.

Review preferences

The review preferences configure the default way that diffs are initially displayed to users when they view them in changelists and code reviews. The review preference settings are used for a user until they change their user preferences, see user profile Settings.

Configure the default user diff view settings with the following configuration block in the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
        'users' => array(
           'settings' => array(
              'review_preferences' => array(
                  'show_comments_in_files'             => true,
                  'view_diffs_side_by_side'            => true,
                  'show_space_and_new_line_characters' => false,
                  'ignore_whitespace'                  => false,
              ),
           ),
        ),
```

`show_comments_in_files`:

- **`true`**: display comments in files and in the **Comments** tab. This is the default value.
- **`false`**: display comments only in the **Comments** tab.

`view_diffs_side_by_side`:

- **`true`**: display diffs in files side by side in two panes. This is the default value.
- **`false`**: display diffs in files inline in a single pane.

`show_space_and_new_line_characters`:

- **`true`**: display whitespace characters as dots, tabs as arrows that point to a bar, and line endings as arrows that point down.
- **`false`**: whitespace, tabs and line endings are not displayed as special characters. This is the default value.

ignore_whitespace:

- **false**: changes made to whitespace are highlighted in file diffs. This makes it easier to identify changes in file types where whitespace is important. This is the default value.
- **true**: changes made to whitespace are not highlighted in file diffs. This makes it easier to see the important changes in file types where whitespace changes are not important.

> **Note**
> **ignore_whitespace** replaces **ignore_whitespace_default** that was used in Swarm version 2017.4 and earlier. By default **ignore_whitespace** is set to false, this is the same original default value set for **ignore_whitespace_default**.

Time preferences

The time preferences configure the default way that time is initially displayed to users. The time preference settings are used for a user until they change their user preferences, see user profile Settings.

Configure the default user time settings with the following configuration block in the *SWARM_ROOT*/**data/config.php** file:

```
<?php
    // this block should be a peer of 'p4'
        'users' => array(
           'settings' => array(
              'time_preferences' => array(
                  'display'  => 'Timeago', // Default to 'Timeago' but can
be set to 'Timestamp'
              ),
           ),
        ),
```

display:

- **'Timeago'**: time is displayed as how long ago the event happened. This is the default value.
- **'Timestamp'**: time is displayed as the date and time the event happened using the local machine browser time.

> **Tip**
> If timestamp is selected, the date format used by Swarm matches the date format configuration of the local machine browser.

Workers

Helix Swarm uses background processes, called *workers*, to respond to events in the Helix server. The default number of workers is 3, and each worker processes events for up to 10 minutes. When a worker terminates, a new one is spawned.

> **Note**
> Each worker maintains a connection to the Helix server for the duration of its lifetime. This may impact your Helix server management practices.

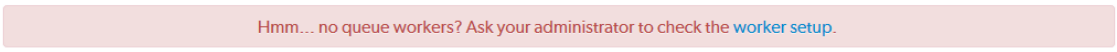
Worker status

To determine the current status of workers, visit the URL: **`https://myswarm.url/queue/status`**.

The response is formatted in JSON, and looks like this:

```
{"tasks":0,"futureTasks":1,"workers":3,"maxWorkers":3,"workerLifetime":"595s"}
```

During normal use of Swarm, the following error message appears for logged-in users when Swarm detects that no workers are running:



Worker configuration

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

To adjust the configuration for workers, add a configuration block to the *SWARM_ROOT*`/data/config.php` file:

```
<?php
    // this block should be a peer of 'p4'
    'queue'  => array(
        'workers'             => 3,    // defaults to 3
        'worker_lifetime'     => 595,  // defaults to 10 minutes (less 5 seconds)
        'worker_task_timeout' => 1800, // defaults to 30 minutes
```

```
        'worker_memory_limit' => '1G', // defaults to 1 gigabyte
    ),
```

where:

- **`workers`** specifies the number of worker processes that should be available. The default is 3. The cron job ensures that new worker processes are started when necessary. If the limit is reached or exceeded, new worker processes are not started.
- **`worker_lifetime`** specifies the amount of time in seconds that a worker process should run for. The default is 595 seconds (10 minutes less 5 seconds). If a worker process exceeds this limit while processing a task, it will complete the active task and then terminate. **`worker_lifetime`** does not cause tasks to terminate mid-processing.
- **`worker_task_timeout`** specifies the maximum amount of time in seconds that a worker process can spend processing a single task. The default is 1800 seconds (30 minutes). This is useful for terminating workers that might get stalled in a variety of situations.
- **`worker_memory_limit`** specifies the maximum amount of memory that a worker process is allowed to use while processing a task. The default is 1G (1 gigabyte).

Manually start workers

To kick off a new worker process, visit the URL: **`https://myswarm.url/queue/worker`**.

When the number of workers running matches the configured limit, the requested worker process is not started.

> **Note**
> This technique does start a worker, but it lasts only for its configured lifetime. Typically, you would always want at least one worker running. See "Set up a recurring task to spawn workers" on page 174 for details.

Manually restart workers

To restart an idle worker process, remove its lock file:

```
rm data/queue/workers/worker_id
```

A worker process that is busy processing a task will continue operation until its task is complete. Immediately afterwards, if the worker notices that its lock file is missing it exits.

If you have a recurring task to start workers, the recurring task starts a fresh worker, if necessary. See "Set up a recurring task to spawn workers" on page 174 for details.

Workflow global rules

This section provides information on how to enable workflow and configure the global workflow rules for Swarm.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

workflow

To disable the workflow feature, edit the *SWARM_ROOT*`/data/config.php` file and set the `enabled` configurable to `false` in the `workflow` section of the codeblock:

```
    'workflow' => array(
        'enabled' => false, // Switches the workflow feature on. Default
is true
    ),
```

The default value is `true`.

> **Tip**
> If you disable the workflow feature in the Swarm `config.php` file, workflow will not be processed by Swarm but a small overhead is still incurred by the Helix server each time it runs a workflow trigger script. This overhead can be eliminated by commenting out the `swarm.enforce change-submit`, `swarm.strict change-content`, and `swarm.shelvesub shelve-submit` workflow triggers.

Global workflow

> **Note**
> - The **Global Workflow** page can be edited by a Swarm user with *super* or *admin* privileges, or users that have been made an owner.
> - In earlier versions of Swarm, global workflow rules were set in the Swarm `config.php` file. From Swarm 2020.1, they are set in the global workflow on the Swarm **Workflows** page. If you upgrade from an earlier version of Swarm, your global workflow settings are automatically migrated to a workflow called **Global Workflow**.

- In earlier versions of Swarm, global tests were set in the Swarm `config.php` file. From Swarm 2020.1, tests are added to the global workflow on the Swarm **Workflows** page so that they operate as global tests. If you upgrade from an earlier version of Swarm, your global tests are automatically migrated. Each global test is migrated as follows: the owner is set as the Swarm admin user (not shared), the name is set to the original test name, the description is set to the original test title, and it is added to the **Global Workflow**.

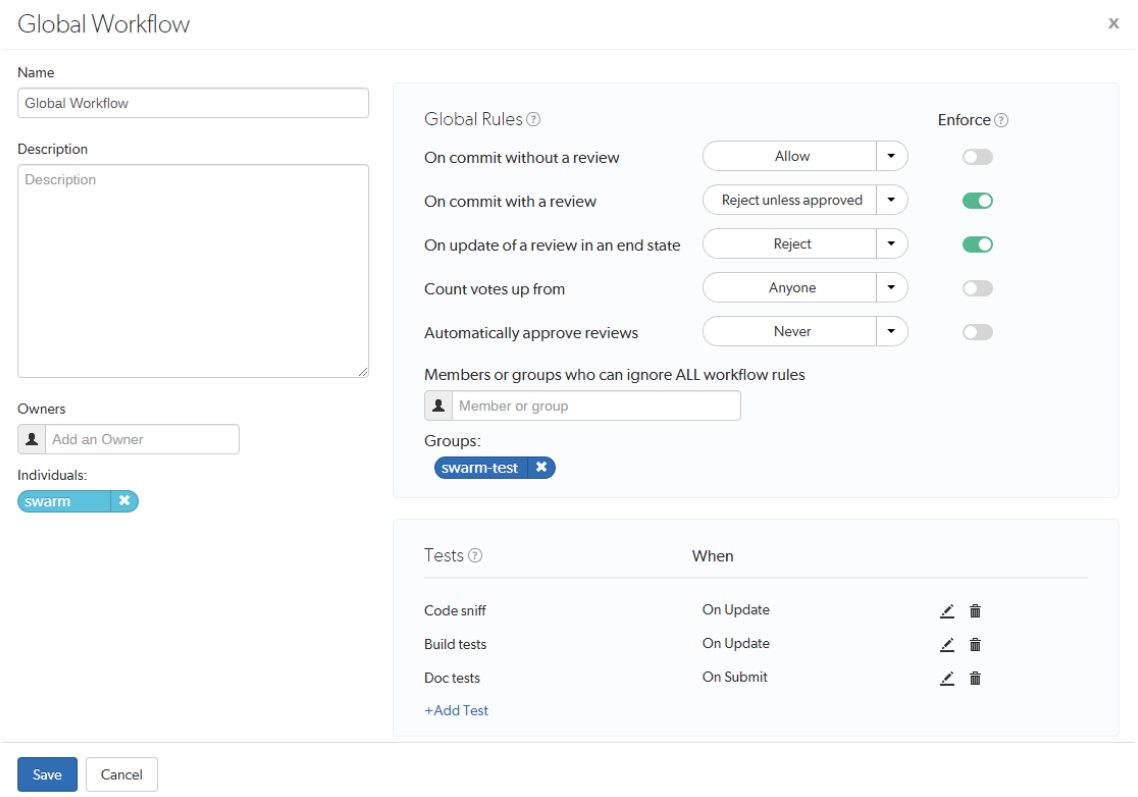
Global workflow rules enable you apply rules globally if required, ensuring basic company policies are followed by every project. Each rule in the global workflow can be set with Enforce off (default) or Enforce on, this determines how that rule is used by Swarm:

- **Enforce off:** applies the workflow rule setting to projects and project branches that don't have an associated workflow. If a project or project branch has an associated Swarm workflow, the global rule is ignored.
- **Enforce on:** applies a minimum workflow rule setting to all projects and project branches. If a project or project branch has an associated workflow, the global rule is merged with the workflow rule and the most restrictive setting is used.

To edit the global workflow rules:

1. On the Swarm **Workflows** page.
2. Click **Global Workflow** to open the settings page:

> **Tip**
> By default, the global workflow is called **Global Workflow**, but it can be changed. To help you identify the global workflow it is always shown as the first workflow in the list and it has a different background color to the other workflows.



3. **Optional:** change the name if required, we recommend that you leave it set to **Global Workflow** if possible for ease of identification.
4. **Optional:** provide a description for the global workflow.
5. **Optional:** add more owners if required. This field auto-suggests groups, and users within Helix server as you type (up to a combined limit of 20 entries).

> **Important**
> - The global workflow must have at least one owner.
> - If you remove yourself as an owner, you cannot edit this workflow configuration later unless you have *super* user rights.

6. **Global Rules:**

   Each rule in the global workflow can be set with Enforce off (default) or Enforce on using the **Enforce** slider next to the rule, this determines how that rule is used by Swarm:

   - **Enforce off:** applies the workflow rule setting to projects and project branches that don't have an associated workflow. If a project or project branch has an associated Swarm workflow, the global rule is ignored.

- **Enforce on:** applies a minimum workflow rule setting to all projects and project branches. If a project or project branch has an associated workflow, the global rule is merged with the workflow rule and the most restrictive setting is used.

> **Important**
> - Changes to global workflow rules that are set with **Enforce off** will change the workflow for projects and project branches that do not have an associated workflow.
> - Changes to global workflow rules that are set with **Enforce on** will change the workflow for all projects and project branches.

On commit without a review:

This rule is applied when a changelist without an associated review is submitted from outside of Swarm.

Select one of the following options:

- **Allow:** the changelist is not checked, the changelist is submitted without a review.
- **Create a review:** the changelist is submitted and a review is created automatically for the changelist.
- **Reject:** the changelist submit is rejected.

> **Tip**
> The selected rule is also applied when a changelist is submitted with `#review` in the description. For more information about creating a review by including a keyword in the changelist description, see "Create a review" on page 548.

On commit with a review: This rule is applied when:

- A Swarm review is committed.
- A changelist with an associated review is submitted from outside of Swarm.

Select one of the following options:

- **Allow:** the changelist review state is not checked. The changelist is committed even if its associated review is not approved.
- **Reject unless approved:** the changelist is only submitted if its associated review is approved and the content of the changelist is identical to the content of the approved review.

> **Tip**
> The selected rule is also applied when a changelist is submitted with `#review-nnnnn`, `#replace-nnnnn`, or `#append-nnnnn` in the description (`nnnnn` = review ID). For more information about adding a changelist to a review by including a keyword in the changelist description, see "Add a changelist to a review" on page 548.

On update of a review in an end state:

Used to stop review content being automatically changed for reviews that are in specific states. By default, the protected end states are **Archived**, **Rejected**, and **Approve:Commit**. The end states are set by the Swarm administrator, see end_states.

> **Tip**
> When a review is in a protected end state, it can still be updated manually by a user from the Swarm UI.

This rule is applied when a changelist is added to a review.

Select one of the following options

- **Allow:** the review is not checked. The changelist is added to the review no matter what the review state is. This is the default setting.
- **Reject:** if the review is in one of the states specified in the `end_state` configurable:
  - The **Add Change** button is disabled for the review.
  - The changelist is rejected if it is added outside of Swarm using `#review-nnnnn`, `#replace-nnnnn`, or `#append-nnnnn` in the description (`nnnnn` = review ID).

Count votes up from:

By default, all of the up votes on a review are counted for the **Minimum up votes** value set on the project/branch the review is associated with. Limit the up votes that are counted to just the members of the project the review is associated with by using this rule.

This rule is applied when a user votes on a review.

Select one of the following options:

- **Anyone:** votes are counted for all reviewers on a review. This is the default setting.
- **Members:** only the up votes of members of the project the review is associated with are counted for the **Minimum up votes** set on projects/branches.

> **Tip**
> For instructions on how to set **Minimum up votes** for projects and branches, see Project minimum up votes and Branch minimum up votes.

Automatically approve reviews:

By default, reviews must be manually approved. Enable automatic approval of reviews with this rule.

This rule is applied when a user votes on a review, a required reviewer is added to a review, or a required reviewer is made an optional reviewer on a review.

Select one of the following options:

- **Never:** reviews are not automatically approved. This is the default setting.
- **Based on vote count:** reviews are automatically approved if:
  - There are no down votes on the review.
  - There are no moderators on the review. If a review has moderators it cannot be automatically approved.
  - All of the Required reviewers on the review have voted up.
  - The **Minimum up votes** on the review has been satisfied for **each** of the projects and branches the review spans.

> **Important**
> Moderators prevent the automatic approval of reviews. For more information about moderators, see "Moderators" on page 379.

> **Tip**
> After a review has been automatically approved it needs to be manually committed.

Members or groups who can ignore ALL workflow rules

Optional: add users and groups to enable the users and groups (including any sub-groups) to ignore all of the workflow rules across the entire depot. This field auto-suggests users, and groups within Helix server as you type (up to a combined limit of 20 entries).

Typically, these permissions are given to a small set of trusted users and groups, for example project owners and administrators.

> **Tip**
> This applies to actions taken in the Swarm UI, the P4D command line, a P4D client, and the Swarm API.

7. **Tests**

   Optional: add tests to the workflow, the tests are run when a review associated with the workflow is either created/updated or submitted. Selected from the **When** dropdown for the test.

> **Tip**
> - Tests are only saved on the workflow when you click the **Save** button.
> - If a review is associated with multiple workflows, all of the tests on all of those workflows are run for the review. If the same test is on more than one of the associated workflows, it is only run once.

Add a test:

a. Click +Add Test in the Tests area.

b. Select the test to run from the **Tests** dropdown list.

c. Select when you want the test to run from the **When** dropdown list.

> **Tip**
> Review content change is when the files or content of files in a review change. A change to the review description does not trigger a test.

- **On Update** the test will run when:
  - a pre-commit review is created.
  - a review is submitted and the review content has changed compared to the previous review revision.
  - a review is updated and the review content has changed since the previous review revision.
  - a review is updated and the review content has not changed since the previous review revision, but this test failed or is still running on the previous revision. Any other tests for the review that passed for the previous review revision are not rerun and their test runs are copied to the new revision of the review.
- **On Submit** the test will run when a pre-commit review is submitted or a post-commit review is created.

d. Click the **Accept** ✔ button to add the test to the workflow.

Edit a test on the workflow:

a. Click the **Edit** button next to the test you want to edit.

b. The **Test** and **When** dropdown lists are displayed for the test.

c. Make changes as required.

d. Click the **Accept** ✔ button to confirm your changes.

Remove a test from the workflow:

Click the **Delete** button next to the test to remove it from the workflow.

The test is removed immediately, no confirmation is requested. You must save the workflow to complete the test removal.

8. Click **Save**.

> **Note**
> The **Save** button is disabled if any required fields are empty.

12 | Extending Swarm

Helix Swarm is built with open source technologies, using modern development methods, resulting in a platform that is capable, modular, and can be extended in a variety of ways. While this section is currently incomplete, it does identify the technologies that make up Swarm and provides notes regarding some of the available extension points.

Resources

Swarm is built with a number of different technologies. As such it can be a bit of a challenge to ramp up on if you haven't done web development before. This section lists some of the best resources we've found for learning how to develop for Swarm.

jQuery

A free three hour course on jQuery basics. Highly recommended if you haven't used jQuery before. It also covers some JavaScript and CSS basics:

http://try.jquery.com

JavaScript resources

A free, in-browser JavaScript course:

https://www.w3schools.com/js/default.asp

PHP resources

A free PHP tutorial:

https://www.w3schools.com/php7/

Laminas Framework resources

> **Note**
> Swarm supports the Laminas component versions in the **`LICENSE.txt`** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **`LICENSE.txt`** file is in the **`readme`** folder of your Swarm installation.

- The documentation overview page for Laminas Framework:

  https://docs.laminas.dev/
- Laminas tutorial that covers most of the basics:

  https://docs.laminas.dev/tutorials/
- Laminas Quick Start documentation:

  https://docs.laminas.dev/laminas-mvc/quick-start/

Development mode

Swarm has a *development* mode that, when enabled, changes Swarm's behavior in the following ways:

- CSS and JavaScript code is not aggregated. Each CSS or JavaScript file is fetched individually, which makes it much easier to identify where styles or functions exist and how they apply to Swarm. Due to the additional HTTP requests, development mode should not be used in production environments.
- Errors and exceptions that may occur are displayed in Swarm's UI. This is particularly useful to developers of Swarm modules. As the error information might disclose system paths or configuration, development mode should not be used in production environments.

> **Tip**
> If you make a configuration change, Swarm will not use it until the configuration cache has been reloaded, this forces Swarm to use the new configuration. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

To enable development mode:

Adjust the *`SWARM_ROOT`***`/data/config.php`** file to include:

```
<?php
    return array(
        'p4' => ...
        'environment'   => array(
            'mode'  => 'development'
```

```
        )
    );
```

To disable development mode:

Adjust the ***SWARM_ROOT*/data/config.php** file to exclude the `'mode' => 'development'` line.

Modules

You can design and implement your own custom Swarm modules to modify the behavior of Swarm. This section covers the design and implementation of Swarm custom modules.

Important
The operation and testing of custom modules is the responsibility of the module creator. Perforce Software, Inc. is not responsible for the operation of your custom modules, nor does Perforce Software, Inc. make any representations or warranties regarding the interoperability of your custom modules with Helix Swarm.

Tip
You must test your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

Tip
If you add or edit a module, Swarm will not use that change until the config cache has been reloaded, this forces Swarm to use the module change. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Module overview

Important
The operation and testing of custom modules is the responsibility of the module creator. Perforce Software, Inc. is not responsible for the operation of your custom modules, nor does Perforce Software, Inc. make any representations or warranties regarding the interoperability of your custom modules with Helix Swarm.

Tip
You must test your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

Tip
If you add or edit a module, Swarm will not use that change until the config cache has been reloaded, this forces Swarm to use the module change. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

A Swarm custom module is created in a folder in the `modules` folder of your Swarm installation. The folder name must match the custom module name and must contain at least a `Module.php` file and a `module.config.php` file. A custom module will not work in Swarm until you enable it. Enable your custom modules in Swarm by adding them to the `custom.modules.config.php` file in the `config` directory (a peer of the `modules` directory).

This chapter provides basic information about how custom modules integrate with Swarm and assumes a basic knowledge of the Laminas framework. For more information about the Laminas framework, see Laminas Quick Start documentation.

This section includes:

Migrate existing custom modules to the Laminas framework

Swarm 2020.1 and later uses the Laminas framework, previous Swarm versions used the Zend framework. The move to Laminas is part of our commitment to move away from using versions of platforms that have reached End-of-Life (EOL).

If you have any custom Swarm modules that were created for Swarm 2019.3 or earlier, you must modify them so that they will work in Swarm 2020.1 and later. The modifications required depend on the version of Swarm you were using the custom modules with before your upgraded Swarm:

- "Custom modules on Swarm 2019.1, 2019.2, or 2019.3 " below
- "Custom modules on Swarm 2018.3 and earlier" below

Custom modules on Swarm 2019.1, 2019.2, or 2019.3

> **Note**
> Migrate your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

Custom modules used with Swarm 2019.1, 2019.2, and 2019.3 use the Zend 3 framework, they must be migrated to the Laminas framework before they can be used with Swarm 2020.1 and later:

1. Migrate your Zend 3 custom modules to the Laminas framework by pointing the Laminas migration tool at the folder you store your custom modules in. For the migration tool and full instructions on migrating your custom Zend 3 modules to Laminas, see https://docs.laminas.dev/migration/.
2. Swarm will not use the custom modules until the Swarm config cache has been deleted. For instructions on deleting the config cache, see "Swarm config cache file delete" on page 680.
3. Check that your custom modules work as expected before putting them on your production system.

Custom modules on Swarm 2018.3 and earlier

> **Note**
> Update and migrate your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

Custom modules used with Swarm 2018.3 and earlier use the Zend 2 framework, they must be migrated to the Laminas framework before they can be used with Swarm 2020.1 and later:

1. Convert your Zend 2 custom modules to the Zend 3 framework, see Upgrade custom modules to work with Zend 3.

> **Tip**
> Laminas is based on the Zend 3 framework, this makes it is relatively simple to convert your Zend 3 custom modules to the Laminas framework:
>
> - If you only have one or two custom modules, you can manually change all of the **`Zend`** references in your modules to **`Laminas`** while you are converting them from Zend 2. This means you can skip the Laminas migration step.
> - If you have a larger number of custom modules, it is probably quicker to automatically change all of the references in your custom modules by running the Laminas migration tool as described in the next step.

2. Migrate your Zend 3 custom modules to the Laminas framework by pointing the Laminas migration tool at the folder you store your custom modules in. For the migration tool and full instructions on migrating your custom Zend 3 modules to Laminas, see https://docs.laminas.dev/migration/.
3. Swarm will not use the custom modules until the Swarm config cache has been deleted. For instructions on deleting the config cache, see "Swarm config cache file delete" on page 680.
4. Check that your custom modules work as expected before putting them on your production system.

Influence activity events, emails, etc.

When something happens in Helix server (change submitted, files shelved, job added/edited), or state changes within Swarm (comment added, review state changed, etc.), the event is pushed onto the Swarm task queue. A background worker process pulls events off of the queue and publishes an event alerting modules about activity they may be interested in processing. This architecture allows the Swarm user interface to be fairly quick while accommodating tasks that might require notable processing time, or need to wait for related information to become available.

Subscribers to the worker event flesh the item out (fetch the change/job details, for example) and indicate if it should result in an activity entry, email notification, etc. By subscribing to various event topics, your custom module can pay attention to specific types of events. While your custom module is processing an event, it can modify the text of activity events, change the contents of emails, drop things entirely from activity, etc.

When your custom module subscribes to an event, set the priority to influence how early or late in the process it runs. You will likely want your module to run after most other modules have done their work to flesh out the event, but before Swarm's events module processes it. The events module is a good example of subscribing to these events:

***`swarm_install`*`/module/Events/config/module.config.php`**

> **Tip**
> Note that its priority is set to **`-100`**. Select a value before that for your own module (for example, **`0`** would be neutral and **`-90`** would indicate that you are interested in being last).

> Event priority is from the highest positive number to the lowest negative number with **0** in the middle. For a full list of tasks and their event priorities, see "Task details" on page 625.

Task types

For details of each of the following tasks, see "Task details" on page 625.

- `task.commit`
- `task.shelve`
- `task.review`
- `task.change`
- `task.mail`
- `task.cleanup.attachment`
- `task.cleanup.archive`
- `task.shelvedel`
- `task.changesave`
- `task.changesaved`
- `task.comment`
- `task.comment.batch`
- `task.commentSendDelay`
- `task.group`
- `task.groupdel`
- `task.job`
- `task.user`
- `task.userdel`
- `task.workflow.created`
- `task.workflow.updated`

Templates

Override existing view templates using your custom module. Have a look at this example module that demonstrates how to customize the email templates Swarm uses for comment notifications.

For more information about views, see the Laminas-View Quick Start.

> **Note**
> Swarm supports the Laminas component versions in the **`LICENSE.txt`** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **`LICENSE.txt`** file is in the **`readme`** folder of your Swarm installation.

Custom module file locations

The custom module files must be stored using the PSR-4 standard locations shown below:

```
config/
      custom.modules.config.php (required)
module/
      ModuleName/ (required)
                config/ (required)
                      module.config.php (required)
                src/ (php files here, required)
                   Controller/
                             MyIndexController.php
                Listener/ (optional)
                        MyListener.php (optional)
                   ...
                   other directories (non-php files here, optional)
                   ...
                Module.php (required)
```

IndexControllers

Create your **`IndexControllers`** using factories. The **`IndexControllers`** should extend **`Application\Controller\AbstractIndexController`** and use the Swarm **`IndexControllerFactory`**, this means that the **`IndexControllerFactory`** can automatically inject the services .

> **Tip**
> We recommend you follow the **`::class`** description to avoid errors in string representation.

```
'controllers' => array(
    'factories' => [
        MyModuleName\Controller\IndexController::class =>
Application\Controller\IndexControllerFactory::class,
```

```
    ],
),
```

View helpers

Create your view helpers using factories. The factories that create the view helpers must inject any services that are required.

> **Tip**
> We recommend using **`::class`** rather than arbitrary strings as this can limit the possibility of errors. The use of classes also makes your factory simpler because reflection can be used to construct the helpers if they all extend a common parent class.

```
'view_helpers' => array(
        'factories' => array_fill_keys(
            [
                MyModuleName\View\Helper\Helper1::class,
                MyModuleName\View\Helper\Helper2::class,
            ],
            <YourHelperFactory>::class
        )
    )
```

Set options on existing helpers

It is possible to influence the behavior of existing view helpers by setting options on them; for an example see: ***swarm_install*/module/Application/Module.php**.

Register new helpers

It is also possible to register new view helpers by placing them within your module's hierarchy, for example, **MyModule/src/View/Helper/Foo.php**. Use the following Swarm view helper for inspiration: ***swarm_install*/module/Activity/src/View/Helper/Activity.php**.

Then register your view helper with the view manager via your **ModuleConfig**: ***swarm_root*/module/Activity/config/module.config.php**.

Event listeners

Your event listeners should extend **Events\Listener\AbstractEventListener**. This means that the **ListenerFactory** can create them without you having to write your own factory.

The **Listener.php** file contains the implementation of your event listener and the **module.config.php** file configures your event listeners.

Tip

- Look at the code in the **AbstractEventListener.php** file in **module/events/src/Listener** to see the functions that are available to you.
- We strongly recommend that you create your event listeners to use the declarative model because it has a number of advantages over the non-declarative model:
  - **AbstractEventListener** provides common code for listeners to use
  - The declarative model offers better debug options (logging)
  - The declarative model is neater, easier to read, easier to support, and easier to maintain

Example module.config.php file that uses the declarative model for event listeners:

Tip
Listener::class example details:

- **Events\Listener\ListenerFactory::ALL => [**

  We are listening to **ALL** here for convenience because we are listening for mail events but this means it will trigger on every event. Usually is better to just listen for the events you are interested in. For example, if you are interested in commits and shelves you would listen on **COMMIT** and **SHELVE** events.
- **Events\Listener\ListenerFactory::PRIORITY => -199,**

  Declares an event priority of **-199** for the event listener because email delivery events are processed with a priority of **-200** and the example event needs to run just before the email delivery event.
- **Events\Listener\ListenerFactory::CALLBACK => 'handleEmail',**

  Declares the function name within the listener class that is called.
- **Events\Listener\ListenerFactory::MANAGER_CONTEXT => 'queue'**

  Triggers your custom listener when Swarm processes the Swarm event queue.

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/
```

```
$listeners = [EmailExample\Listener\Listener::class];
return [
    'listeners' => $listeners,
    'service_manager' =>[
        'factories' => array_fill_keys(
            $listeners,
            Events\Listener\ListenerFactory::class
        )
    ],
    Events\Listener\ListenerFactory::EVENT_LISTENER_CONFIG => [
        Events\Listener\ListenerFactory::ALL => [
            EmailExample\Listener\Listener::class => [
                [
                    Events\Listener\ListenerFactory::PRIORITY => -199,
                    Events\Listener\ListenerFactory::CALLBACK =>
'handleEmail',
                    Events\Listener\ListenerFactory::MANAGER_CONTEXT =>
'queue'
                ]
            ]
        ]
    ]
];
```

Example of an email Listener.php file:

```
namespace MyModuleName\Listener;

use Events\Listener\AbstractEventListener;
use Laminas\EventManager\Event;

class Listener extends AbstractEventListener
{
    public function handleEmail(Event $event)
    {
        $mail = $event->getParam('mail');
        if (!$mail || !isset($mail['htmlTemplate'], $mail
```

```
['textTemplate'])) {
            return;
        }
    }
}
```

Enabling your custom modules

Swarm uses the **custom.modules.config.php** file to check which custom modules it should load. This gives you control over which modules Swarm loads and prevents modules from being loaded by mistake.

Create the **custom.modules.config.php** file in the **config** directory (a peer of the **modules** directory) if it does not already exist. Edit the file so that it contains the following details for each of your modules:

- **namespaces** array: enter your custom module names and paths
- **return** array : enter your custom module names

For example, if you have three modules called **MyModuleName**, **AnotherModuleName**, and **NewModuleName** the file content would look similar to:

```
<?php
\Laminas\Loader\AutoloaderFactory::factory(
    array(
        'Laminas\Loader\StandardAutoloader' => array(
            'namespaces' => array(
                'MyModuleName'       => BASE_PATH .
'/module/MyModuleName/src',
                'AnotherModuleName'  => BASE_PATH .
'/module/AnotherModuleName/src',
                'NewModuleName'      => BASE_PATH .
'/module/NewModuleName/src',
            )
        )
    )
);
return [
    'MyModuleName',
    'AnotherModuleName',
```

```
    'NewModuleName',
];
```

Further reading

For detailed information about the Laminas framework and examples see the following:

- The Laminas Quick Start documentation and Laminas framework documentation portal, it is useful to have a basic knowledge of the Laminas framework before you create your own modules. .

  > **Note**
  > Swarm supports the Laminas component versions in the **`LICENSE.txt`** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **`LICENSE.txt`** file is in the **`readme`** folder of your Swarm installation.

- "Example linkify module" on page 628, a simple custom module that replaces a specific piece of text with a link for changelists, jobs, code review descriptions, comments, and activity entries.
- "Example email module" on page 633, a simple custom module that makes Swarm use a custom email template when it sends out when a comment notification is sent out.
- Swarm JIRA module, a simple module implementation within Swarm: ***`swarm_install`*`/module/Jira`**

Task details

Swarm has lots of events that can been listened to and it uses these to process the tasks to generate data for Swarm.

This section lists the Swarm tasks available for use with your modules. They are listed along with the priorities of the task events. Events for each task are listed in highest to lowest priority order.

task.commit

- **Priority 300:** the listener checks if this is a Git Fusion change.
- **Priority 200:** the listener searches for all of the projects that are impacted by the change and checks for any user mentions that need to be linked. It sets up the email list here and queues the activity to be processed.
- **Priority 100:** the listener determines whether it should update or create any reviews for the changelist.
- **Priority -100:** the listener checks if any users have the Review daemon setup and attaches those users to the email to list.

- **Priority -200:** has two Listeners:
  - One listener checks the `disable_change_emails` configurable setting:

    `true`, the email will not be sent.

    `false`, Swarm adds any reviews that might not be part of the *to list* because they are not a member of the project.
  - The other listener creates the activity for all users/projects that are linked to this event.
- **Priority -300:** the listener sends out the email that is related to the committed change.
- **Priority -400:** the listener links to JIRA for this change list.

task.shelve

- **Priority 300:** the listener checks if this is a Git Fusion change.
- **Priority 200:** the listener post processes a Git Fusion shelve changelist to ensure it complies with standard P4D shelved reviews.
- **Priority 100:** the listener processes the shelve task to determine whether it should create or updates any reviews.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.
- **Priority -300:** the listener sends out the email that is related to the shelved change.

task.review

- **Priority 100:** the listener processes the review and updates any records that are linked to the review. For example, @mention, new author, new participants, activity, and requirement level.
- **Priority -200:** the listener creates the activity for all users/project that are linked to this event.
- **Priority -300:** has two Listeners:
  - One listener sends out the email that is related to the review change.
  - The other listener fetches any associated changes and ensures they are linked to JIRA.

task.change

Priority 1: the listener ensures the change and any mentions in the description are linked to JIRA issues.

task.mail

Priority 1: this event is manually triggered from the `review task`. Other events do not manually trigger this task, they wait for it to run at the end. Once we are here Swarm will validate all of the email settings and send the email out.

task.cleanup.attachment

Priority 1: the listener will delete the attachment if a cleanup attachment task has been queued.

task.cleanup.archive

Priority 1: the listener will clean up any archive files if their life time has expired.

task.shelvedel

- **Priority 1:** the listener processes any changelist that has had files deleted from it.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.

task.changesave

Priority 1: the listener processes the change to ensure that it is valid and puts a future task in the queue to do the `task.changesaved` tasks.

task.changesaved

Priority 1: the listener takes the changelist description and updates the review description if `sync description` is enabled.

task.comment

- **Priority 1:** the listener processes the comment being created.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.
- **Priority -300:** the listener sends out an email related to the committed change.

task.comment.batch

- **Priority 1:** If comments are being batched, the listener will process the comment(s) and put a task into the queue to be processed when the `notification_delay_time` has elapsed.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.
- **Priority -300:** the listener sends out an email related to the committed change.

task.commentSendDelay

- **Priority 1:** if the user or system has forced a batched comment to be sent the listener will setup the email here.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.
- **Priority -300:** the listener sends out an email related to the committed change.

task.group

Priority 1: the listener invalidates the group cache if the group is changed.

task.groupdel

Priority 1: the listener invalidates the group cache if the group has been deleted.

task.job

- **Priority 1:** the listener sets up the Activity for the job and attaches any users/projects to the activity. It also checks if any users want to be notified about Job changes.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.
- **Priority -300:** has two Listeners:
  - One listener sends out an email related to the committed change.
  - The other listener fetches associated changes and ensures they are linked to JIRA.

task.user

Priority 1: the listener invalidates the user cache if the user is changed.

task.userdel

Priority 1: the listener invalidates the user cache if the user has been deleted.

task.workflow.created

- **Priority 1:** the listener sets up the Activity for the new workflow and attaches any users/projects to the activity.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.

task.workflow.updated

- **Priority 1:** the listener sets up the Activity for the updated workflow and attaches any users/projects to the activity.
- **Priority -200:** the listener creates the activity for all users/projects that are linked to this event.

Example linkify module

The following example module demonstrates how to turn text that appears in changelist, job, or code review descriptions, comments, and activity entries into links.

> **Note**
> Swarm supports the Laminas component versions in the **`LICENSE.txt`** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **`LICENSE.txt`** file is in the **`readme`** folder of your Swarm installation.

Tip
You must test your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

Tip
If you add or edit a module, Swarm will not use that change until the config cache has been reloaded, this forces Swarm to use the module change. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Basic steps needed to create the Email Example module are:

- "Create the Module.php file" below
- "Create the module.config.php file" on page 631
- "Enable the Rickroll module for Swarm in custom.modules.config.php" on page 632

File locations

For reference, the Rickroll module uses the following filenames and locations:

```
config/
      custom.modules.config.php
module/
      Rickroll/
                config/
                       module.config.php
                Module.php
```

Create the Module.php file

`Module.php` will:

- Declare the module namespace, this must match the directory name of the module.
- Provide the Linkify application.
- Provide the **`onBootstrap()`** function.
- Add a callback to the Linkify module that declares what text is to be searched for (**`rickroll`**) and, when found, how to compose the link for that text.
- Provide the **`getConfig()`** function.

Create the Module.php file:

1. Create a directory called **`Rickroll`** in the module directory.
2. Create the file **`Module.php`** in **`module/Rickroll`**.
3. Edit **`Module.php`** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

namespace Rickroll;

use Application\Filter\Linkify;

class Module
{
    public function onBootstrap()
    {
        Linkify::addCallback(
            // Linkify requires 3 parameters: "$linkify", "$value"
and "$escaper"
            function ($linkify, $value, $escaper) {
                if (strcasecmp($value, 'rickroll')) {
                    // not a hit; tell caller we did not handle this
one
                    return false;
                }

                return '<a target="_new"
href="https://www.youtube.com/watch?v=dQw4w9WgXcQ">'
                            . $escaper->escapeHtml($value) . "</a>";
```

```
            },
            'rickroll',
            strlen('rickroll')
        );
    }

    public function getConfig()
    {
        return include __DIR__ . '/config/module.config.php';
    }
}
```

4. Now "Create the module.config.php file" below.

Create the module.config.php file

The **`module.config.php`** file is required but for the linkify module only has an empty return.

Create the module.config.php file:

1. Create a file called **`module.config.php`** in **`Rickroll/config`**.
2. Edit **`module.config.php`** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

{
    return array();
}
```

3. Now "Enable the Rickroll module for Swarm in custom.modules.config.php" on the next page.

Enable the Rickroll module for Swarm in custom.modules.config.php

Swarm uses the **`custom.modules.config.php`** file to auto-load classes and to check which custom modules it should run. This gives you control over which modules Swarm loads and prevents modules from being loaded by mistake.

Create the custom.modules.config.php file:

1. Create the **`config`** directory at the same level as the **`module`** directory if it does not exist.
2. Create the **`custom.modules.config.php`** file in the **`config`** directory if it does not exist.
3. Edit the **`custom.modules.config.php`** file so that it contains the auto-loader and the **`Rickroll`** module details:

   > **Tip**
   > If you already have one or more custom modules enabled for Swarm, the auto-loader and the existing module details will already be in the file.
   >
   > Just add **`Rickroll`** to the **`namespaces`** and **`return`** arrays of the **`custom.modules.config.php`** file.

   ```
   <?php
   \Laminas\Loader\AutoloaderFactory::factory(
       array(
           'Laminas\Loader\StandardAutoloader' => array(
               'namespaces' => array(
                   'Rickroll'      => BASE_PATH . '/module/Rickroll',
               )
           )
       )
   );
   return [
       'Rickroll'
   ];
   ```

4. The Rickroll module is now enabled for Swarm. Swarm will now turn text that appears in changelist, job, or code review descriptions, comments, and activity entries into links.
5. The Rickroll module is now enabled for Swarm. Swarm will now replace the text "rickroll " with a link to "https://www.youtube.com/watch?v=dQw4w9WgXcQ".
6. Check that the module works correctly before moving it to your production server.

Example email module

The following example module demonstrates how to customize the email template Swarm uses when sending notifications for comments.

> **Note**
> Swarm supports the Laminas component versions in the `LICENSE.txt` file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The `LICENSE.txt` file is in the `readme` folder of your Swarm installation.

> **Tip**
> You must test your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

> **Tip**
> If you add or edit a module, Swarm will not use that change until the config cache has been reloaded, this forces Swarm to use the module change. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Basic steps needed to create the Email Example module are:

- "Create the Module.php file" on the next page
- "Create the module.config.php file" on page 635
- "Create the Listener.php file" on page 637
- "Create the comment-html.phtml file" on page 640
- "Create the comment-text.phtml file" on page 642
- "Enable the EmailExample module for Swarm in custom.modules.config.php" on page 643

Further reading:

"Extending the EmailExample module to other email templates" on page 644

File locations

For reference, the EmailExample module uses the following filenames and locations:

```
config/
      custom.modules.config.php
module/
      EmailExample/
```

```
config/
       module.config.php
src/
   Listener/
            Listener.php
view/
     mail/
          commit-html.phtml
          commit-text.phtml
          comment-html.phtml
          comment-text.phtml
          review-html.phtml
          review-text.phtml
Module.php
```

Create the Module.php file

`Module.php` will:

- Declare the module namespace, this must match the directory name of the module
- Provide the **`getConfig()`** function

> **Tip**
> **Optional:** You can also implement the **`onBootstrap (Event $event)`** function if you want to do some early setup when the module is loaded:
>
> ```
> public function onBootstrap(Event $event)
> {
> }
> ```
>
> **`Event`** is a Laminas class and is added after namespace:
>
> ```
> namespace EmailExample
> use Laminas\EventManager\Event
> ```

Create the Module.php file:

1. Create a directory called **`EmailExample`** in the module directory.
2. Create the file **`Module.php`** in **`module/EmailExample`**.
3. Edit **`Module.php`** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

namespace EmailExample;

class Module
{
    public function getConfig()
    {
        return include __DIR__ . '/config/module.config.php';
    }
}
```

4. Now "Create the module.config.php file" below.

Create the module.config.php file

The **`module.config.php`** file will:

- Configure your module including event listeners
- Declare an event priority of **`-199`** for the event listener because email delivery events are processed with a priority of **`-200`** and the example event needs to run just before the email delivery event.

Tip
`Listener::class` example details:

- **`Events\Listener\ListenerFactory::ALL => [`**

  We are listening to **`ALL`** here for convenience because we are listening for mail events but this means it will trigger on every event. Usually is better to just listen for the events you are interested in. For example, if you are interested in commits and shelves you would listen on **`COMMIT`** and **`SHELVE`** events.

- **`Events\Listener\ListenerFactory::PRIORITY => -199,`**

Declares an event priority of **-199** for the event listener because email delivery events are processed with a priority of **-200** and the example event needs to run just before the email delivery event.

- **Events\Listener\ListenerFactory::CALLBACK => 'handleEmail',**

  Declares the function name within the listener class that is called.
- **Events\Listener\ListenerFactory::MANAGER_CONTEXT => 'queue'**

  Triggers your custom listener when Swarm processes the Swarm event queue.

Create the module.config.php file:

1. Create a directory called **config** in the **EmailExample** directory.
2. Create a file called **module.config.php** in **config**.
3. Edit **module.config.php** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

$listeners = [EmailExample\Listener\Listener::class];
return [
    'listeners' => $listeners,
    'service_manager' =>[
        'factories' => array_fill_keys(
            $listeners,
            Events\Listener\ListenerFactory::class
        )
    ],
    Events\Listener\ListenerFactory::EVENT_LISTENER_CONFIG => [
        Events\Listener\ListenerFactory::ALL => [
            EmailExample\Listener\Listener::class => [
                [
```

```
                    Events\Listener\ListenerFactory::PRIORITY => -
199,
                    Events\Listener\ListenerFactory::CALLBACK =>
'handleEmail',
                    Events\Listener\ListenerFactory::MANAGER_CONTEXT
=> 'queue'
                ]
            ]
        ]
    ]
];
```

4. Now "Create the Listener.php file" below.

Create the Listener.php file

The **`Listener.php`** file will:

- Contains the implementation of your event listener
- Allow logging

Create the Listener.php file:

1. Create a directory called **`src`** in the **`EmailExample`** directory.
2. Create a directory called **`Listener`** in the **`src`** directory.
3. Create a file called **`Listener.php`** in **`Listener`**.
4. Edit **`Listener.php`** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

namespace EmailExample\Listener;

use Events\Listener\AbstractEventListener;
use Laminas\EventManager\Event;

class Listener extends AbstractEventListener
{
    /**
     * Automatically uses any custom email templates found under this
     * module's view/mail folder (e.g. Example/view/mail/commit-
html.phtml).
     *
     * Valid templates include:
     *
     *  commit-html.phtml (HTML version of commit notification)
     *  commit-text.phtml (text version of commit notification)
     *  comment-html.phtml (HTML version of comment notification)
     *  comment-text.phtml (text version of comment notification)
     *  review-html.phtml (HTML version of review notification)
     *  review-text.phtml (text version of review notification)
     *
     * Note: you need to provide custom templates for both HTML and
text;
     * if you do not provide both, it is possible that the search for
```

```
     * customized templates only finds the non-customized versions,
making
     * it appear that this module is not working.
     */

    /**
     * Handle the Email and set the new templates.
     *
     * @param Event $event
     */
    public function handleEmail(Event $event)
    {
        $logger = $this->services->get('logger');
        $logger->info("EmailExample: handleEmail");
        $mail = $event->getParam('mail');
        if (!$mail || !isset($mail['htmlTemplate'], $mail
['textTemplate'])) {
            return;
        }

        $html = __DIR__ . '/view/mail/' . basename($mail
['htmlTemplate']);
        $text = __DIR__ . '/view/mail/' . basename($mail
['textTemplate']);

        if (file_exists($html)) {
            $mail['htmlTemplate'] = $html;
        }
        if (file_exists($text)) {
            $mail['textTemplate'] = $text;
        }

        $event->setParam('mail', $mail);
        $logger->info("EmailExample: handleEmail end.");
```

```
    }
}
```

5. Now "Create the comment-html.phtml file" below.

Create the comment-html.phtml file

The **comment-html.phtml** file is a view script that provides the content for the HTML portion of the comment notification email.

> **Tip**
> It is considered best practice to use inline CSS for styling emails.

Create the comment-html.phtml file:

1. Create a directory called **view** in the **module/Example** directory.
2. Create a directory called **mail** in the **module/Example/view** directory.
3. Create the **comment-html.phtml** file in **module/Example/view/mail**.
4. Edit **comment-html.phtml** to contain:

```
<?php
    $user       = $activity->get('user');
    $userLink   = $user
                ? $this->qualifiedUrl('user', array('user' => $user))
                : null;
    $targetLink = $activity->getUrl($this->plugin('qualifiedUrl'));
?>
<html>
  <body style="font-family: sans-serif; background-color: #eee;
padding: 1em;">
    <div style="background-color: #fff; border: 1px solid #ccc;
padding: 1em;">
      <div style="font-size: 115%;">
        <?php if ($user): ?>
          <a style="text-decoration: none;" href="<?php echo
$userLink ?>">
            <?php echo $this->escapeHtml($user) ?>
          </a>
        <?php endif; ?>
        <?php echo $this->escapeHtml($activity->get('action')) ?>
```

```
        <a style="text-decoration: none;" href="<?php echo
$targetLink ?>">
          <?php echo $this->escapeHtml($activity->get('target'))?>
        </a>
      </div>
      <br/>
      <?php
          // if the comment has file context, show it.
          $comment = $event->getParam('comment');
          $context = $comment
                   ? $comment->getFileContext()
                   : array('content' => null, 'line' => null);
          if (is_array($context['content']) && $context['line']) {
             $line = $context['line'] - count($context['content']) +
1;
             echo '<div style="font-family: monospace; white-space:
nowrap;'
                . ' padding: .5em 1em; overflow-x: auto; color:
#444;'
                . ' border: 1px solid #ddd; background-color:
#f7f7f7;">';
             foreach ((array) $context['content'] as $i => $content)
{
                echo '<div><span style="color: #999;">'
                    . str_pad($line + $i,
                              strlen($context['line']),
                              "0",
                              STR_PAD_LEFT
                      )
                    . '.</span> '
                    . $this->preformat($content)
                          ->setLinkify(false)
                          ->setEmojify(false)
                          ->setWordWrap(900)
                    . "</div>\n";
```

```
            }
            echo '</div><br/>';
        }
    ?>
    <div style="padding-bottom: .5em;">
    <?php
        echo $this->preformat($activity->get('description'))
                ->setBaseUrl($this->qualifiedUrl())
                ->setEmojify(false)
                ->setWordWrap(900)
    ?>
    </div>
  </div>
</body>
</html>
```

5. Now "Create the comment-text.phtml file" below.

Create the comment-text.phtml file

The **comment-text.phtml** is a view script that provides the content for the text-only portion of the comment notification email.

Create the comment-text.phtml file:

1. Create the file **comment-text.phtml** in the **module/Example/view/mail** directory.
2. Edit **comment-text.phtml** to contain:

```
<?php
    echo trim($activity->get('user')
        . ' commented on '
        . $activity->get('target'));

    // if the comment has file context, show it.
    $comment = $event->getParam('comment');
    $context = $comment
             ? $comment->getFileContext()
             : array('content' => null);
    if (is_array($context['content'])) {
        echo "\n\n> " . $this->wordWrap(
```

```
            implode("\n> ", $context['content']), 900
        );
    }

    echo "\n\n" . trim($this->wordWrap($activity->get('description'),
900));
    echo "\n\n" . $activity->getUrl($this->plugin('qualifiedUrl'));
?>
```

3. Now "Enable the EmailExample module for Swarm in custom.modules.config.php" below.

Enable the EmailExample module for Swarm in custom.modules.config.php

Swarm uses the **`custom.modules.config.php`** file to auto-load classes and to check which custom modules it should run. This gives you control over which modules Swarm loads and prevents modules from being loaded by mistake.

Create the custom.modules.config.php file:

1. Create the **`config`** directory at the same level as the **`module`** directory if it does not exist.
2. Create the **`custom.modules.config.php`** file in the **`config`** directory if it does not exist.
3. Edit the **`custom.modules.config.php`** file so that it contains the auto-loader and the **`EmailExample`** module details:

> **Tip**
> If you already have one or more custom modules enabled for Swarm, the auto-loader and the existing module details will already be in the file.
>
> Just add **`EmailExample`** to the **`namespaces`** and **`return`** arrays of the **`custom.modules.config.php`** file.

```
<?php
\Laminas\Loader\AutoloaderFactory::factory(
    array(
        'Laminas\Loader\StandardAutoloader' => array(
            'namespaces' => array(
                'EmailExample'      => BASE_PATH .
'/module/EmailExample/src',
            )
        )
    )
);
return [
    'EmailExample'
];
```

4. The Email Example module is now enabled for Swarm. Swarm will use the new custom email template whenever a comment notification is sent out.
5. Check that the module works correctly before moving it to your production server.

Extending the EmailExample module to other email templates

If you need to customize any other types of Swarm notification email messages, locate the view scripts (both HTML and text) and copy them into **`module/Example/view/mail`**, maintaining the existing filenames, then modify the new files as desired.

> **Note**
> If you do not copy both the HTML and text templates, it is possible for the search for customized templates to only find non-customized versions, making it appear that your module is not working.

Example Slack module

The following example module demonstrates how to integrate Swarm with Slack so that Swarm posts a message in a Slack channel each time a change is committed, a review is created, or a review is updated.

> **Note**
> Swarm supports the Laminas component versions in the **`LICENSE.txt`** file, features and functions in the Laminas documentation that were introduced in later versions of Laminas will not work with Swarm. The **`LICENSE.txt`** file is in the **`readme`** folder of your Swarm installation.

Tip
You must test your custom modules on a test system before transferring them to your production system. This avoids any negative impact on the operation of your production system. If you have more than one custom module, the modules should all be tested at the same time on the same test system as this ensures that the modules operate correctly with each other and with Helix Swarm.

Tip
If you add or edit a module, Swarm will not use that change until the config cache has been reloaded, this forces Swarm to use the module change. You must be an *admin* or *super* user to reload the Swarm config cache. Navigate to the **User id** dropdown menu, select **System Information**, click the **Cache Info** tab, and click the "Reload Configuration button" on page 588.

Basic steps needed to create the Slack module are:

- "Create the Module.php file" below
- "Create the module.config.php file" on page 647
- "Create the SlackActivityListener.php file" on page 649
- "Enable the Slack module for Swarm in custom.modules.config.php" on page 655

File locations

For reference, the Slack module uses the following filenames and locations:

```
config/
      custom.modules.config.php
module/
      Slack/
            config/
                  module.config.php
            src/
                Listener/
                        SlackActivityListener.php
            Module.php
```

Create the Module.php file

`Module.php` will:

- Declare the module namespace, this must match the directory name of the module.
- Provide the **`getConfig()`** function

> **Tip**
> **Optional:** You can also implement the **`onBootstrap (Event $event)`** function if you want to do some early setup when the module is loaded:
>
> ```
> public function onBootstrap(Event $event)
> {
> }
> ```
>
> **`Event`** is a Laminas class and is added after namespace:
>
> ```
> namespace Slack
> use Laminas\EventManager\Event
> ```

Create the Module.php file:

1. Create a directory called **`Slack`** in the module directory.
2. Create the file **`Module.php`** in **`module/Slack`**.
3. Edit **`Module.php`** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

namespace Slack;

class Module
{

    public function getConfig()
    {
        return include __DIR__ . '/config/module.config.php';
    }
```

```
}
```

4. Now "Create the module.config.php file" below.

Create the module.config.php file

The **module.config.php** file will:

- Configure your module including event listeners.
- Provide an icon for Swarm that is displayed in Slack
- Declare an event priority of **-110** for the **COMMIT** and **REVIEW** event listeners because the Activity event is processed with a priority of **-200** and this is a sensible time to post the message to Slack.

> **Tip**
> **Listener::class** example details for the **COMMIT** and **Review** classes:
>
> - **Events\Listener\ListenerFactory::COMMIT => [**
>
>   We are listening to **COMMIT** here because we are interested in commits.
> - **Events\Listener\ListenerFactory::Review => [**
>
>   We are listening to **Review** here because we are interested in when a review is created or updated.
> - **Events\Listener\ListenerFactory::PRIORITY => -110,**
>
>   Declares an event priority of **-110** for the event listener because the Activity event is processed with a priority of **-200** and this is a sensible time to post the message to Slack.
> - **Events\Listener\ListenerFactory::CALLBACK => 'handleCommit',**
>
>   Declares the function name within the listener class that is called.
> - **Events\Listener\ListenerFactory::CALLBACK => 'handleReview',**
>
>   Declares the function name within the listener class that is called.
> - **Events\Listener\ListenerFactory::MANAGER_CONTEXT => 'queue'**
>
>   Triggers your custom listener when Swarm processes the Swarm event queue.

Create the module.config.php file:

1. Create a directory called **config** in the **Slack** directory.
2. Create a file called **module.config.php** in **config**.
3. Edit **module.config.php** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

use Events\Listener\ListenerFactory as EventListenerFactory;

$listeners = [Slack\Listener\SlackActivityListener::class];

return [
    'listeners' => $listeners,
    'service_manager' =>[
        'factories' => array_fill_keys(
            $listeners,
            Events\Listener\ListenerFactory::class
        )
    ],
    EventListenerFactory::EVENT_LISTENER_CONFIG => [
        EventListenerFactory::TASK_COMMIT => [
            Slack\Listener\SlackActivityListener::class => [
                [
                    Events\Listener\ListenerFactory::PRIORITY => -
110,
                    Events\Listener\ListenerFactory::CALLBACK =>
'handleCommit',
                    Events\Listener\ListenerFactory::MANAGER_CONTEXT
=> 'queue'
                ]
            ]
```

```
        ],
        EventListenerFactory::TASK_REVIEW => [
            Slack\Listener\SlackActivityListener::class => [
                [
                    Events\Listener\ListenerFactory::PRIORITY => -
110,
                    Events\Listener\ListenerFactory::CALLBACK =>
'handleReview',
                    Events\Listener\ListenerFactory::MANAGER_CONTEXT
=> 'queue'
                ]
            ]
        ]
    ],
    'slack' => [
        'user'        => 'Swarm',
        'icon'        =>
            'https://swarm.workshop.perforce.com/view/guest/perforce_
software/slack/main/images/60x60-Helix-Bee.png',
        'max_length'  => 80,
    ]
];
```

4. Now "Create the SlackActivityListener.php file" below.

Create the SlackActivityListener.php file

The **`SlackActivityListener.php`** file will:

- Contains the implementation of your event listener
- Contains the Slack URLs and messages for a commit and a review

> **Tip**
> The configuration of your Slack URLs and messages will depend on your Slack system and your requirements. This is a basic example but you can edit it so that you post your messages to a different channel for each project branch. You can even add to the **Project settings** page using CSS and JavaScript so that users can configure the Slack channel URLs and messages for each project.

- Allow logging

Create the SlackActivityListener.php file:

1. Create a directory called **`src`** in the **`Slack`** directory.
2. Create a directory called **`Listener`** in the **`src`** directory.
3. Create a file called **`SlackActivityListener.php`** in **`Listener`**.
4. Edit **`SlackActivityListener.php`** to contain:

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2019 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
*/

namespace Slack\Listener;

use Events\Listener\AbstractEventListener;
use P4\Spec\Change;
use P4\Spec\Exception\NotFoundException;
use Reviews\Model\Review;
use Laminas\EventManager\Event;
use Laminas\Http\Client;
use Laminas\Http\Request;

class SlackActivityListener extends AbstractEventListener
{
    public function handleReview(Event $event)
    {
        $logger = $this->services->get('logger');
        $logger->info("Slack: handleReview");
        $p4Admin = $this->services->get('p4_admin');
        try {
            $review = Review::fetch($event->getParam('id'),
$p4Admin);

            // URL to POST messages to Slack
            // TODO Need a URL
            $url = '<YourSlackUrl>';
```

```
            // Construct your Slack Review message here
            $text = 'Review ' . $review->getId();
            $this->postSlack($url, $text);
        } catch (\Exception $e) {
            $logger->err("Slack:" . $e->getMessage());
            return;
        }
        $logger->info("Slack: handleReview end.");
    }

    public function handleCommit(Event $event)
    {
        // connect to all tasks and write activity data
        // we do this late (low-priority) so all handlers have
        // a chance to influence the activity model.
        $logger = $this->services->get('logger');
        $logger->info("Slack: handleCommit");

        // task.change doesn't include the change object; fetch it if
we need to
        $p4Admin = $this->services->get('p4_admin');
        $change  = $event->getParam('change');
        if (!$change instanceof Change) {
            try {
                $change = Change::fetchById($event->getParam('id'),
$p4Admin);
                $event->setParam('change', $change);
            } catch (NotFoundException $e) {
            } catch (\InvalidArgumentException $e) {
            }
        }

        // if this isn't a submitted change; nothing to do
        if (!$change instanceof Change || !$change->isSubmitted()) {
```

```
            $logger->info("Slack: not a change...");
            return;
        }
        try {
            // URL to POST messages to Slack
            // TODO Need a URL
            $url = '<YourSlackUrl>';
            // Construct your Slack Commit message here
            $text = 'Review ' . $change->getId();
            $this->postSlack($url, $text);
        } catch (\Exception $e) {
            $logger->err('Slack: ' . $e->getMessage());
        }
        $logger->info("Slack: handleCommit end.");
    }

    private function postSlack($url, $msg)
    {
        $logger = $this->services->get('logger');
        $config = $this->services->get('config');

        $icon = $config['slack']['icon'];
        $user = $config['slack']['user'];

        $logger->info("Slack: POST to $url");
        $logger->info("Slack: user=$user");
        $logger->info("Slack: icon=$icon");

        try {
            $json = [
                'payload' => json_encode(
                    [
                        'username' => $user,
                        'icon_url' => $icon,
```

```
                        'text' => $msg
                    ]
                )
            ];

            $request = new Request();
            $request->setMethod(Request::METHOD_POST);
            $request->setUri($url);
            $request->getPost()->fromArray($json);

            $client = new Client();
            $client->setEncType(Client::ENC_FORMDATA);

            // set the http client options; including any special
overrides for our host
            $options = $config + ['http_client_options' => []];
            $options = (array) $options['http_client_options'];
            if (isset($options['hosts'][$client->getUri()->getHost
()])) {
                $options = (array) $options['hosts'][$client->getUri
()->getHost()] + $options;
            }
            unset($options['hosts']);
            $client->setOptions($options);

            // POST request
            $response = $client->dispatch($request);

            if (!$response->isSuccess()) {
                $logger->err(
                    'Slack failed to POST resource: ' . $url . ' (' .
                    $response->getStatusCode() . " - " . $response-
>getReasonPhrase() . ').',
                    [
```

```
                        'request'   => $client->getLastRawRequest(),
                        'response'  => $client->getLastRawResponse()
                    ]
                );
                return false;
            }
            return true;

        } catch (\Exception $e) {
            $logger->err($e);
        }
        return true;
    }
}
```

5. Now "Enable the Slack module for Swarm in custom.modules.config.php" below.

Enable the Slack module for Swarm in custom.modules.config.php

Swarm uses the **`custom.modules.config.php`** file to auto-load classes and to check which custom modules it should run. This gives you control over which modules Swarm loads and prevents modules from being loaded by mistake.

Create the custom.modules.config.php file:

1. Create the **`config`** directory at the same level as the **`module`** directory if it does not exist.
2. Create the **`custom.modules.config.php`** file in the **`config`** directory if it does not exist.
3. Edit the **`custom.modules.config.php`** file so that it contains the auto-loader and the **`Slack`** module details:

> **Tip**
> If you already have one or more custom modules enabled for Swarm, the auto-loader and the existing module details will already be in the file.
>
> Just add **`Slack`** to the **`namespaces`** and **`return`** arrays of the **`custom.modules.config.php`** file.

```
<?php
\Laminas\Loader\AutoloaderFactory::factory(
    array(
        'Laminas\Loader\StandardAutoloader' => array(
            'namespaces' => array(
                'Slack'      => BASE_PATH . '/module/Slack/src',
            )
        )
    )
);
return [
    'Slack'
];
```

4. The Slack module is now enabled for Swarm. Swarm will post a message on a Slack channel each time a change is committed, a review is created, or a review is updated.
5. Check that the module works correctly before moving it to your production server.

CSS & JavaScript

Custom CSS and JavaScript files will be loaded automatically if you place them under either:

SWARM_ROOT/public/custom/*.(css|js)

SWARM_ROOT/public/custom/sub-folder/*.(css|js)

Note

- Swarm only supports customizations placed directly within the ***SWARM_ROOT*/public/custom** folder or one sub-folder down. Customizations are added after all standard CSS and JavaScript. If more than one custom file is present they are added in alphabetical order.
- Prior to creating any customizations, ensure that the ***SWARM_ROOT*/public/custom** folder exists; Swarm does not ship with or create this folder.

Sample Javascript extensions

The following are example Javascript customizations that you might wish to apply to your Swarm installation. Each example can be implemented separately. Ideally, you would apply the Javascript customizations in a single file to reduce the number of web requests required.

> **Warning**
> Coding errors in your custom JavaScript files could cause the Swarm UI to stop working. If this occurs, use your browser's development tools to identify which file contains the problem, and move that file out of the *SWARM_ROOT*`/public/custom` folder. When the problem has been resolved, the file can be returned.

Make the *Created* column on the Reviews page clickable

```
$(document).on( 'click', '.reviews-table td.created', function() {
    var change = $(this).closest('tr').data('id');
    window.location = '/reviews/' + change;
});
```

Save these lines in a file, perhaps `reviews-created-clickable.js`, within the ***SWARM_ROOT*/public/custom** folder. Swarm automatically includes the JavaScript file, making the entries in the *Created* column clickable immediately for all for subsequent review page views.

Customize the review state options text

```
var original = swarm.review.buildStateMenu;
swarm.review.buildStateMenu = function(){
    original();
    var needsReview = $('.icon-review-needsReview').parent();
    needsReview.html(needsReview.html().replace(
        'Needs Review', 'You need to review'
    ));
}
```

Replace the *You need to review* with the text you'd prefer to see instead of `Needs Review`.

Save this line in a file, perhaps `customize-review-states.js`, within the ***SWARM_ROOT*/public/custom**. Swarm automatically includes the JavaScript file, adjusting the text of the review states immediately for subsequent page views.

Sample CSS customizations

The following are example CSS customizations that you might wish to apply to your Swarm installation. Each example can be implemented separately. Ideally, you would apply the CSS customizations in a single file to reduce the number of web requests required.

> **Tip**
> If your Swarm administrator has configured Swarm to connect to more than one Helix server instance you can customize Swarm for each server instance it is connected to. For instructions on customizing Swarm for individual Helix server instances, see "CSS customization when Swarm is connected to multiple-Helix server instances" on page 661.

Adjust the default tab size

```
body {
   tab-size: 4;
}
```

Replace the ***4*** with the tab size you prefer.

Save these lines in a file, perhaps **`tab-size.css`**, within the *SWARM_ROOT*`/public/custom` folder. Swarm automatically includes the CSS file, adjusting the tab size immediately for subsequent page views.

Apply a custom background to the login screen

```
.session-container .modal-overlay.login::after {
    background-image: url('/custom/login_background.jpg');
    background-position: top center;
    background-size: 100%;
}
```

Replace the ***`/custom/login_background.jpg`*** URL fragment with the image you want to use. If you do not specify a full URL, the image you specify must exist within the ***`SWARM_ROOT`*`/public`** folder, preferably within the ***`SWARM_ROOT`*`/public/custom`** folder.

Save these lines in a file, perhaps **`login-background.css`**, within the ***`SWARM_ROOT`*`/public/custom`** folder. Swarm automatically includes the CSS file, immediately replacing the login screen's background for subsequent page views.

Replace Swarm's logo in the main navigation bar

```
#react-swarm-app-container .menu-container .logo a {
    background-repeat: no-repeat;
    background-image: url('/custom/navbar_logo.jpg');
    background-size: 116px 25px;
    background-position: 15px center;
    background-clip: content-box;
}
```

Replace the `/custom/navbar_logo.jpg` URL fragment with the image you want to use. If you do not specify a full URL, the image you specify must exist within **the *SWARM_ROOT*/public** folder, preferably within the ***SWARM_ROOT*/public/custom** folder.

Save these lines in a file, perhaps **navbar-logo.css**, within the ***SWARM_ROOT*/public/custom** folder. Swarm automatically includes the CSS file, immediately replacing the Swarm logo for subsequent page views.

> **Note**
> Swarm's navbar design supports logos up to 24 pixels tall. Even if your logo fits within that height, you may need to also adjust the width, height, margins, or padding to suit your logo.

Apply a custom background color to the navigation bar

```
#react-swarm-app-container .swarm-navigation {
    background-color: #000000;
}
```

Replace the **#000000** with the Hex color you want to use for the navigation bar.

Save these lines in a file, perhaps **navbar-color.css**, within the ***SWARM_ROOT*/public/custom** folder. Swarm automatically includes the CSS file, immediately replacing the navigation bar's background color for subsequent page views.

Adjust the appearance of avatars

The Swarm UI is changing and moving toward using React, currently there is a mix of React and legacy PHP controlling the UI. This means that there are currently two methods of rendering avatars and you need to customize both.

- For avatars rendered by Swarm using PHP, see "Adjust avatar appearance when rendered by PHP" below
- For avatars rendered by Swarm using React, see "Adjust avatar appearance when rendered by React" on the next page

For instructions on adding custom avatars, see "Add custom avatars" on the next page

Adjust avatar appearance when rendered by PHP

```
img.avatar {
    border: 2px dashed red;
    border-radius: 10%;
}
```

You can make a number of adjustments to the way Swarm presents avatars rendered by PHP, such as adding a border and adjusting the border radius, as the example above demonstrates. You should avoid attempting to set specific sizes because Swarm uses different sizes depending on where the avatar is displayed.

Save these lines in a file, perhaps **`avatars.css`**, within the ***`SWARM_ROOT`*`/public/custom`** folder. Swarm automatically includes the CSS file, immediately changing the appearance of avatars for subsequent page views.

Adjust avatar appearance when rendered by React

```
#react-swarm-app-container .userAvatar img{
    border: 2px dashed red;
    border-radius: 10%;
}
```

You can make a number of adjustments to the way Swarm presents avatars rendered by React, such as adding a border and adjusting the border radius, as the example above demonstrates. You should avoid attempting to set specific sizes because Swarm uses different sizes depending on where the avatar is displayed.

Save these lines in a file, perhaps **`avatars-react.css`**, within the ***`SWARM_ROOT`*`/public/custom`** folder. Swarm automatically includes the CSS file, immediately changing the appearance of avatars for subsequent page views.

Add custom avatars

Custom avatars for individual users or groups are added to Swarm using custom CSS. Custom avatars are intended for use with system users and groups.

To add a custom avatar for a user:

```
img.avatar[data-user="<userid>"], .userAvatar img[alt="<user
name>"]{
    content: url(<path to avatar>);
}
```

To add a custom avatar for a group:

```
img.avatar[data-user="<groupid>"], .userAvatar img[alt="<group
name>"]{
    content: url(<path to avatar>);
}
```

Save these lines in a file, perhaps **avatars-custom.css**, within the ***SWARM_ROOT*/public/custom** folder. Swarm automatically includes the CSS file, immediately using your custom avatars for subsequent page views.

Add a custom icon to a menu item

When you create a custom menu item in the menu_helpers configuration block, the **cssClass** specified for the menu item is appended to **h2.menu-** in the Swarm CSS.

This enables you to add some custom CSS to Swarm to change the default icon for the menu item. For example, the CSS below adds the **double-speech-bubbles.svg** icon to the menu item with the **cssClass** of **custom-menu**:

```
.swarm-menu-item.menu-custom-menu .svgIcon{
    display:none;
}
.swarm-menu-item.menu-googleMain a::before{
    content: '';
    background-image: url('/swarm/img/icons/line/double-speech-
bubbles.svg');
    background-size: 16px;
    background-repeat: no-repeat;
    padding-left: 18px;
}
```

CSS customization when Swarm is connected to multiple-Helix server instances

If Swarm is configured to connect to more than one Helix server instance, Swarm can be customized for each of the Helix server instances it is connected to. This gives users an simple visual prompt as to which Helix server instance they are currently viewing in Swarm. All of the customizations described in the section above are available for multiple-Helix server instances.

Note

- The Swarm Global Dashboard cannot be customized.
- For instructions on how to configure Swarm to connect to multiple-Helix server instances, see "Multiple-Helix server instances" on page 519.

To customize Swarm for a Helix server instance, include the **[data-server-id="*<server-name>*"]** data attribute in the CSS selector you are customizing. Replace **<server-name>** with the Helix server name the customization is for.

> **Tip**
> To apply a customization to all Swarm Helix server instances, don't include the data attribute in the CSS selector you are customizing.

See the examples below to see how the data attribute is used to customize Swarm for the individual Helix server instances. As with the customization of Swarm when it is connected to a single Helix server, you should ideally apply the CSS customizations in a single file to reduce the number of web requests required.

Apply a custom background color to the navigation bar of each of the Helix server instances

```
body[data-server-id="serverA"] #react-swarm-app-container .swarm-
navigation {
    background-color: #d21544;
}
body[data-server-id="serverB"] #react-swarm-app-container .swarm-
navigation {
    background-color: pink;
}
```

For more information about customizing the navigation bar color, see "Apply a custom background color to the navigation bar" on page 659.

Apply custom background images to the login screen of each of the Helix server instances

```
body.route-login[data-server-id="serverA"] .session-container .modal-
overlay.login::after {
    background-position: top center;
    background-size: 100%;
    background-image: url(/custom/login_background_A.jpg);
}
body.route-login[data-server-id="serverB"] .session-container .modal-
overlay.login::after {
    background-position: top center;
    background-size: 100%;
    background-image: url(/custom/login_background_B.jpg);
}
```

For more information about customizing the login screen background image, see "Apply a custom background color to the navigation bar" on page 659.

Swarm API

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

This chapter describes the REST-like API provided by Swarm, which can be used to automate common Swarm interactions or integrate with external systems.

API versions

Beginning with Swarm 2019.3, we have started adding a new set of v10 APIs to Swarm. These will provide a new endpoint and response pattern, and are designed for use with the new rich User Interface that is being introduced in 2020. The v9 APIs will continue to be available for some time to come, but their functionality will be duplicated and expanded in the v10 APIs over the next few releases.

v10 does not currently provide a full set of features, so it is recommended that you use v9 for now if the features that you need are not present in v10. APIs older than v9 are being deprecated, and support for them will be removed in a future release.

Important

- Swarm will continue to support the v9 APIs and you can continue to use them if you prefer.
- Any improvements made in the migration to v10 will not be backported to v9.
- New API endpoints will be created as v10 and will not be backported to v9.

Current API versions:

API version	Swarm Release	Date	Description
v10	2019.3	October 2019	Include support for integration with CI tools.
v9	2018.1	April 2018	Include support for append and replace changelist to a review, 2 Factor Authentication and mark a comment as read.

Deprecated API versions:

From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

API version	Swarm Release	Date	Description
v8	2017.4	December 2017	Include support for default reviewers on a project or project branch.
v7	2017.3	October 2017	Include support for groups as participants of a review and groups as moderators of a project branch.
v6	2017.1	May 2017	Include support for activity dashboard, archiving of inactive reviews, cleaning completed reviews and for voting reviews up and down.
v5	2016.3	December 2016	Include support for limiting comments to a specific review version.
v4	2016.2	October 2016	Include support for private projects, as well as file-level and line-level inline comments.
v3	2016.1 SP1	September 2016	Include new endpoint for comments.
v2	2016.1	May 2016	Include new endpoints for projects, groups, etc.
v1.2	2015.3	October 2015	Add author filter to the list reviews endpoint.
v1.1	2014.4	January 2015	Addition of required reviewers, and `apiVersions`.
v1	2014.3	July 2014	Initial release.

Swarm API basics

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

This section describes the API endpoints that are commonly used when constructing an API call to Swarm.

Authentication

Swarm's API requires an authenticated connection for all data-modifying endpoints. Authenticated connections are achieved using HTTP Basic Access Authentication.

> **Note**
> API endpoints require authentication unless `require_login` is set to **`false`**.

For example:

```
curl -u "apiuser:ticket" https://myswarm.url/api/v9/projects
```

Swarm accepts a ticket from the Helix Core server (previous versions of Swarm required this ticket to be host-unlocked, this is no longer true since Swarm 2017.1). It might also be possible to use a password in place of the ticket if your Helix server allows it.

To acquire a ticket, run the following command:

```
p4 -p myp4host:1666 -u apiuser login -p
```

> **Important**
> For a Helix Core server that has been configured for security level 3, passwords are not accepted. For more information on security levels, see on Server security levels in the *Helix Core Server Administrator Guide*.

> **Note**
> If you use a ticket to authenticate against the Swarm API and the ticket expires, you need to acquire a new ticket to continue using the API.

If you make a request that requires authentication and you have not authenticated, the response is:

```
HTTP/1.1 200 OK


{
  "error": "Unauthorized"
}
```

Requests

> **Important**
> Some API endpoints have undocumented metadata parameters, these are for internal Swarm use only. The metadata returned is subject to change without notice and is not suitable for use in customer API calls.

Swarm's API includes endpoints that provide, create, and update information within Swarm.

If you make a request against an endpoint with a method that is not supported, the response is:

```
HTTP/1.1 200 OK


{
  "error": "Method Not Allowed"
}
```

GET information

> **Tip**
> GET requests require authentication unless `require_login` is set to **`false`**.

Use HTTP **`GET`** requests to ask for information from the API.

For example, to get the list of reviews using the v10 API:

```
curl -u "username:password" "https://my-swarm-host/api/v10/reviews
```

Certain API calls support a **`fields`** parameter that allows you to specify which fields to include in the response, enabling more compact data sets.

Fields can be expressed as a comma-separated list, or using array-notation. For example:

```
curl -u "username:password" "https://my-swarm-
host/api/v10/projects/jam?fields=id,description,members"
```

The following endpoints support fields:

API (v9)

- **`/api/v9/activity`**
- **`/api/v9/comments`**
- **`/api/v9/groups`**
- **`/api/v9/groups/{id}`**
- **`/api/v9/projects`**
- **`/api/v9/reviews`**
- **`/api/v9/reviews/{id}`**
- **`/api/v9/users`**
- **`/api/v9/workflows`**
- **`/api/v9/workflows/{id}`**

API (v10)

- `/api/v10/files/{id}`
- `/api/v10/projects`
- `/api/v10/projects/{id}`
- `/api/v10/reviews`
- `/api/v10/reviews/{id}`
- `/api/v10/reviews/{id}/files`
- `/api/v10/reviews/{id}/testruns`
- `/api/v10/search`
- `/api/v10/specs/{spec}/fields`
- `/api/v10/testdefinitions`
- `/api/v10/workflows`
- `/api/v10/workflows/{id}`

POST new information

Use HTTP `POST` requests to create information via the API.

For example, to create a review using form-encoded values:

```
curl -u "apiuser:password" -X POST -d"change=12345"
https://myswarm.url/api/v9/reviews
```

The response should be similar to:

```
HTTP/1.1 200 OK


{
  "isValid": true,
  "id": 12206
}
```

To create a review using JSON:

```
curl -u "apiuser:password" -H "Content-type: application/json" -X
POST -d'{"change": 12345}' https://myswarm.url/api/v9/reviews
```

Update

Use HTTP `PATCH` requests to update information via the API.

If your HTTP client does not support `PATCH` requests, you can emulate this behavior by submitting an HTTP `POST` with a `"?_method=PATCH"` parameter.

Pagination

Most Swarm endpoints that provide data include the ability to paginate their results.

Each time data is requested, up to `max` results are included in the response, as is a value called `lastSeen`. `lastSeen` identifies the `id` of the last entry included in the results. If there are no further results, `lastSeen` is `null`.

To get the next set of results, include `after` set to the value of `lastSeen` in the API request. Entries up to and including the `id` specified by `after` are excluded from the response, and the next `max` entries are included.

See the Activity endpoint for example usage that demonstrates pagination.

Responses

Swarm's API responses are JSON formatted.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Swarm API endpoints (v9)

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

This section includes coverage for the endpoints provided by the API (v9).

Activity : Swarm Activity List

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

List Activity Entries

Summary

List Activity Entries

GET /api/v9/activity

Description

Retrieve the Activity List.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`change`	Optionally filter activity entries by associated Changelist ID. This only includes records for which there is an activity entry in Swarm.	integer	form	No	
`stream`	Optional activity stream to query for entries. This can include user-initiated actions (`user-alice`), activity relating to a user's followed projects/users (`personal-alice`), review streams (`review-1234`), and project streams (`project-exampleproject`).	string	form	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`type`	Type of activity, for example, `change`, `comment`, `job`, or `review`.	string	form	No	
`after`	An activity ID to seek to. Activity entries up to and including the specified ID are excluded from the results and do not count towards `max`. Useful for pagination. Commonly set to the `lastSeen` property from a previous query.	integer	query	No	
`max`	Maximum number of activity entries to return. This does not guarantee that `max` entries are returned. It does guarantee that the number of entries returned won't exceed `max`. Server-side filtering may exclude some activity entries for permissions reasons.	integer	query	No	100
`fields`	An optional comma-separated list (or array) of fields to show. Omitting this parameter or passing an empty value shows all fields.	string	query	No	

Example response

Successful Response:

To get the latest activity entry on a review:

```
curl -u "username:password"
http://myswarm.url/api/v9/activity?stream=review-290&max=1
```

Swarm responds with an array of activity entities, and a `lastSeen` value that can be used for pagination:

```
HTTP/1.1 200 OK

{
  "activity": [
    {
      "id": 619,
      "action": "updated files in",
      "behalfOf": null,
      "behalfOfExists": false,
      "behalfOfFullName": "",
      "change": 290,
      "comments": [
        0,
        0
      ],
      "date": "2019-01-23T02:46:59-08:00",
      "depotFile": null,
      "description": "Start of Project Blue Book.",
      "details": [

      ],
      "followers": [

      ],
      "link": [
        "review",
        {
          "review": "290",
          "version": 2
        }
      ],
      "preposition": "for",
      "projectList": {
        "blue-book": [
          "main"
```

```
        ]
      },
      "projects": {
        "blue-book": [
          "main"
        ]
      },
      "streams": {
        "0": "review-290",
        "1": "user-allison.clayborne",
        "2": "personal-allison.clayborne",
        "3": "project-blue-book",
        "7": "group-triage",
        "8": "personal-alex.randolph",
        "9": "personal-jack.boone"
      },
      "target": "review 290 (revision 2)",
      "time": 1548240419,
      "topic": "reviews/290",
      "type": "review",
      "url": "/main/reviews/290/v2/",
      "user": "allison.clayborne",
      "userExists": true,
      "userFullName": "Allison Clayborne"
    }
  ],
  "lastSeen": 618
}
```


Examples of usage

Fetching review history

To get the latest activity entries on a review:

> **Tip**
> In this example, the review has more than 2 activities but the request has been made to limit the response to a `max` of `2` activities. This will return the most recent activities unless `lastseen` is included in the request. This allows for pagination of your results, see "Activity pagination" below.

```
curl -u "username:password"
http://myswarm.url/api/v9/activity?stream=review-
1234&fields=id,action,date,description,type&max=2
```

You can tweak `max` and `fields` to fetch the data that works best for you.

Swarm responds with an array of activity entities, and a `lastSeen` value that can be used for pagination:

```
HTTP/1.1 200 OK

{
  "activity": [
    {
      "id": 619,
      "action": "updated files in",
      "date": "2019-01-23T02:46:59-08:00",
      "description": "Start of Project Blue Book.",
      "type": "review"
    },
    {
      "id": 618,
      "action": "commented on",
      "date": "2019-01-23T02:46:19-08:00",
      "description": "This is a short comment.",
      "type": "comment"
    }
  ],
  "lastSeen": 618
}
```

Activity pagination

To get the second page of activity entries for a review (based on the previous example):

```
curl -u "username:password"
"https://myswarm.url/api/v9/activity?stream=review-
1234&fields=id,date,description,type&max=2&lastSeen=618"
```

Swarm again responds with a list of activity entities and a **`lastSeen`** value:

```
HTTP/1.1 200 OK

{
  "activity": [
    {
      "id": 617,
      "action": "commented on",
      "date": "2018-12-30T12:12:12-07:00",
      "description": "This is the first test comment.",
      "type": "comment"
    },
    {
      "id": 616,
      "action": "commented on",
      "date": "2018-12-27T12:13:14-07:00",
      "description": "Start of Project Blue Book.",
      "type": "review"
    }
  ],
  "lastSeen": 616
}
```

Create Activity Entry

Summary

Create Activity Entry

POST /api/v9/activity

Description

Creates an entry in the Activity List.

Important
The authenticated user must match the user provided unless the authenticated user has minimum admin-level privileges in which case any user may be specified.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`type`	Type of activity, used for filtering activity streams (values can include `change`, `comment`, `job`, `review`).	string	form	Yes
`user`	User who performed the action.	string	form	Yes
`action`	Action that was performed - past-tense, for example, `created` or `commented on`.	string	form	Yes
`target`	Target that the action was performed on, for example, `issue 1234`.	string	form	Yes
`topic`	Optional topic for the activity entry. Topics are essentially comment thread IDs. Examples: `reviews/1234` or `jobs/job001234`.	string	form	No
`description`	Optional description of object or activity to provide context.	string	form	No
`change`	Optional changelist ID this activity is related to. Used to filter activity related to restricted changes.	integer	form	No

Parameter	Description	Type	Parameter Type	Required
`streams[]`	Optional array of streams to display on. This can include user-initiated actions (`user-alice`), activity relating to a user's followed projects/users (`personal-alice`), review streams (`review-1234`), and project streams (`project-exampleproject`).	array (of strings)	form	No
`link`	Optional URL for `target`.	string	form	No

Examples of usage

Creating an activity entry

To create a plain activity entry:

```
curl -u "username:password"
     -X POST
     -d "type=job"
     -d "user=jira"
     -d "action=punted"
     -d "target=review 123" \
     "https://myswarm.url/api/v9/activity"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "activity":{
    "id":620,
    "action":"punted",
    "behalfOf":null,
    "change":null,
    "depotFile":null,
    "description":null,
    "details":[
```

```
    ],
    "followers":[

    ],
    "link":null,
    "preposition":"for",
    "projects":[

    ],
    "streams":[

    ],
    "target":"review 123",
    "time":1560783425,
    "topic":null,
    "type":"job",
    "user":"jira"
  }
}
```

Linking an activity entry to a review

Linking activity entries to reviews is useful. This involves providing **`link`**, **`streams`**, and **`topic`** fields in the activity data. The **`link`** field is used to make the **`review 123`** string in the activity entry clickable. The **`stream`** field is needed so that the activity entry can be attached to the review in the Swarm interface. The **`topic`** field is used to link the activity entry to the comment thread for that topic, in the event that a user wants to comment on the activity.

To create a fully linked activity entry:

```
curl -u "username:password"
     -X POST
     -d "type=job"
     -d "user=jira"
     -d "action=punted"
     -d "target=review 123" \
     -d "streams[]=review-123" \
     -d "link=reviews/123" \
```

```
     -d "topic=reviews/123" \
     "https://myswarm.url/api/v9/activity"
```

Swarm responds with an activity entity:

```
HTTP/1.1 200 OK

{
  "activity": {
    "id": 1375,
    "action": "punted",
    "behalfOf": null,
    "change": null,
    "depotFile": null,
    "description": null,
    "details": [

    ],
    "followers": [

    ],
    "link": "reviews/123",
    "preposition": "for",
    "projects": [


    ],
    "streams": [
      "review-123"
    ],
    "target": "review 123",
    "time": 1461607739,
    "topic": "reviews/123",
    "type": "job",
    "user": "jira"
  }
}
```

Cache: Swarm Cache

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Swarm config cache file delete

Tip
Alternatively, use the **Cache Info** tab of the **System Information** page. Navigate to the **User id** dropdown menu, select **System Information**, and click the **Cache Info** tab. For more information, see "Cache Info" on page 586.

Summary

Introduced for Swarm 2019.2 and later, deletes the Swarm configuration cache files and restarts the workers.

DELETE /api/v9/cache/config/

Description

Used to delete the Swarm configuration cache files after the Swarm configuration or custom modules have changed. When workers complete their current tasks they are automatically restarted so that they use the new Swarm configuration and custom modules for their next tasks.

Important
Only *admin* users can delete the Swarm configuration cache.

Example response

When cache files are not deleted and workers are not shutdown:

For example, this can happen if you make the call and then make the same call again before the cache files have rebuilt.

```
HTTP/1.1 200 OK

{
    "isValid":true,
    "messages":[
        "File <SWARM_ROOT>/data/cache/module-config-cache.php was not
```

```
deleted",
        "File <SWARM_ROOT>/data/cache/module-classmap-cache.php was not
deleted"
    ]
}
```

Example of usage

```
curl -u "username:password" -X DELETE
"https://myswarm.url/api/v9/cache/config/"
```

Assuming that the authenticated user has permission, Swarm responds with the cache deleted and workers shutdown messages:

```
HTTP/1.1 200 OK

{
    "isValid":true,
    "messages":[
        "File <SWARM_ROOT>/data/cache/module-config-cache.php deleted",
        "File <SWARM_ROOT>/data/cache/module-classmap-cache.php deleted",
        "Workers are shutdown. They will start automatically by recurring
task"
    ]
}
```

Verify Redis cache

> **Tip**
> Alternatively, use the **Cache Info** tab of the **System Information** page. Navigate to the **User id** dropdown menu, select **System Information**, and click the **Cache Info** tab. For more information, see "Cache Info" on page 586.

Summary

Introduced for Swarm 2019.2 and later, verifies the Swarm Redis cache.

POST /serverid/api/v9/cache/redis/verify?context=user,group,project,workflow

Description

The data in the Redis cache for groups, users, projects, and workflows should be the same as the data held by the Helix server. However, in some circumstances, such as when changes are made in the Helix server when Swarm is down or if errors occur during updates, the Redis cache can get out of sync with the Helix server.

Used to verify the content of the Redis cache. Use the context parameter to verify specific areas of the Redis cache, if no context is specified the entire Redis cache is verified. If a verification finds an out of sync cache file, Swarm automatically updates the data in that cache.

> **Important**
> Only *admin* users can verify the Redis cache.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`serverid`	If Swarm is connected to multiple Helix Core servers, `serverid` specifies which Helix server the cache should be verified for.	string	path	See description	

Parameter	Description	Type	Parameter Type	Required	Default Value
`context`	Add context to the call to verify a specific area of the Redis cache: ■ No context specified, verifies all of the Redis caches. ■ `user` verifies the Redis user cache. ■ `group` verifies the Redis group cache. ■ `project` verifies the Redis project cache. ■ `workflow` verifies the Redis workflow cache. To verify multiple areas of cache, separate the entries with a comma `,`	string	path	No	

Example usage

To start verification of the user and group caches:

```
curl -u "username:password" -X POST
"https://myswarm.url/api/v9/cache/redis/verify?context=user,group"
```

JSON response

```
HTTP/1.1 200 OK

{
    isValid: true,
    data: {
        user: {
            state: "Queued",
            progress: false
        },
        group: {
```

```
            state: "Queued",
            progress: false
        }
    },
    messages: {
        "Verifying Redis cache for [user,group]"
    }
}
```

Check progress of Redis cache verification

> **Tip**
> Alternatively, use the **Cache Info** tab of the **System Information** page. Navigate to the **User id** dropdown menu, select **System Information**, and click the **Cache Info** tab. For more information, see "Cache Info" on page 586.

Summary

Introduced for Swarm 2019.2 and later, check progress of the Swarm Redis cache verification.

GET /serverid/api/v9/cache/redis/verify?context=user,group,project,workflow

Description

Used to check the progress of Redis cache verification process. Use the context parameter to check verification of specific areas of the Redis cache, if no context is specified progress is checked for all areas of the entire Redis cache.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`serverid`	If Swarm is connected to multiple Helix Core servers, `serverid` specifies which Helix server cache verification is being checked for.	string	path	See description	

Parameter	Description	Type	Parameter Type	Required	Default Value
`context`	Add context to the check verification progress for a specific area of the Redis cache: ■ No context specified, checks verification progress for all of the Redis caches. ■ `user` checks verification progress for the Redis user cache. ■ `group` checks verification progress for the Redis group cache. ■ `project` checks verification progress for the Redis project cache. ■ `workflow` checks verification progress for the Redis workflow cache. To check the verification progress for multiple areas of cache, separate the entries with a comma `,`	string	path	No	

Example usage

To check verification progress for the user and group caches:

```
curl -u "username:password"
"http://myswarm.url/api/v9/cache/redis/verify?context=user,group"
```

JSON response

```
HTTP/1.1 200 OK


{
```

```
    isValid: true,
    data: {
        user: {
            state: "Running",
            progress: "Step 2 of 5; Calculating checksums for Redis keys"
        },
        group: {
            state: "Running",
            progress: "Step 1 of 5; Getting Redis cache entries"
        }
    }
}
```

Tip

- Valid states for verify are; **`Not Queued`**, **`Queued`**, **`Running`**, and **`Failed`**.
- Valid states for progress are, **`Step x of y;`** (where **`x`** is the current step and **`y`** is the total number of steps) followed by:
  - **`Getting Redis cache entries`**
  - **`Calculating checksums for Redis keys`**
  - **`Calculating checksums for Perforce models`**
  - **`Clearing any extraneous Redis cache entries`**
  - **`Indexing any missing Perforce models into the Redis cache`**
  - **`Verification complete`**

Redis cache delete

Summary

Introduced for Swarm 2019.2 and later, deletes the Swarm Redis cache.

DELETE /serverid/api/v9/cache/redis?context=user,group,project,workflow

Description

Used to delete the Redis cache. Use the context parameter to delete specific areas of the Redis cache, if no context is specified the entire Redis cache is deleted.

Important
Only *admin* users can delete the Redis cache.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`serverid`	If Swarm is connected to multiple Helix Core servers, `serverid` specifies which Helix server the cache should be deleted for.	string	path	See description	
`context`	Add context to the call to delete a specific area of the Redis cache: ■ No context specified, deletes all of the Redis caches. ■ `user` deletes the Redis user cache. ■ `group` deletes the Redis group cache. ■ `project` deletes the Redis project cache. ■ `workflow` deletes the Redis workflow cache. To delete multiple areas of cache, separate the entries with a comma `,`	string	path	No	

Example response

When an invalid Redis cache context is specified:

If an invalid context is specified in the call:

```
HTTP/1.1 200 OK

{
    "isValid":true,
    "messages":[
        "Invalid context [usr], must contain only [user, group, project,
workflow]",
    ]
}
```

Example of usage

```
curl -u "username:password" -X DELETE
"https://myswarm.url/serverA/api/
v9/cache/redis?context=user,group"
```

Assuming that the authenticated user has permission, Swarm responds with the cache deleted message:

```
HTTP/1.1 200 OK

{
    "isValid":true,
    "messages":[
        "Deleted key(s) [Swarm^serverA:user-populated-status,
Swarm^serverA:group-populated-status]",
    ]
}
```

Changes : API controller providing a service for changes

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get projects and branches affected by a given change id

Summary

Get projects, and branches, affected by a given change id.

GET /api/v9/changes/{change}/affectsprojects

Description

All authenticated users are able to use this API.

> **Tip**
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
    "change": {
        "id":"1050",
        "projects": {
            "jam": [
                "live",
                "main"
            ]
        }
    }
}
```

Get default reviewers for a given change id

Summary

Get default reviewers for a given change id.

GET /api/v9/changes/{change}/defaultreviewers

Description

All authenticated users are able to use this API.

> **Tip**
>
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
"change": {
    "id": "1050",
    "defaultReviewers": {
        "groups": {
            "group1": {"required": "1"},
            "group2": {}
        },
        "users": {
            "user1": {},
            "user2": {"required": "true"}
        }
    }
}
}
```

Perform checks on a change if enabled

Summary

Performs checks on the change if enabled

GET /api/v9/changes/{id}/check

Description

Performs checks on the change if workflow configuration requires it.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Change to check	string	form	Yes
`type`	The type of check. Must have a value of enforced, strict or shelve	string	form	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "status":   "OK",
  "isValid":  "true",
  "messages": []
}
```

Examples of usage

Carry out enforced checks on a change if configured

To check change 42:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/changes/42/check?type=enforced"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "status":   "OK",
  "isValid":  "true",
  "messages": []
}
```

Carry out enforced checks on a change if configured. Example of a failed 'requires review'

To check change 42:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/changes/42/check?type=enforced"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "status":   "NO_REVIEW",
  "isValid":  "false",
  "messages": ["Change [42] must be associated with a review"]
}
```

Carry out strict checks on a change if configured

To check change 42:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/changes/42/check?type=strict"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "status":   "OK",
```

```
  "isValid":  "true",
  "messages": []
}
```

Comments : Swarm Comments

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get List of Comments

GET /api/v9/comments/

Summary

Get List of Comments

Description

List comments.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`after`	A comment ID to seek to. Comments up to and including the specified ID are excluded from the results and do not count towards `max`. Useful for pagination. Commonly set to the `lastSeen` property from a previous query.	integer	query	No	
`max`	Maximum number of comments to return. This does not guarantee that `max` comments are returned. It does guarantee that the number of comments returned won't exceed `max`.	integer	query	No	100
`topic`	Only comments for given topic are returned. Examples: `reviews/1234`, `changes/1234` or `jobs/job001234`.	string	query	No	
`context[version]`	If a `reviews/1234` topic is provided, limit returned comments to a specific version of the provided review.	integer	query	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`ignoreArchived`	Prevents archived comments from being returned. (v5+)	boolean	query	No	
`tasksOnly`	Returns only comments that have been flagged as tasks. (v5+)	boolean	query	No	
`taskStates`	Limit the returned comments to ones that match the provided task state (one or more of `open`, `closed`, `verified`, or `comment`). (v5+)	array (of strings)	query	No	
`fields`	An optional comma-separated list (or array) of fields to show for each comment. Omitting this parameter or passing an empty value shows all fields.	string	query	No	

Examples of successful responses

Successful Response:

```
HTTP/1.1 200 OK

{
  "topic": "",
  "comments": {
    "51": {
      "id": 51,
      "attachments": [],
      "body": "Short loin ground round sin reprehensible, venison west
```

```
participle triple.",
      "context": [],
      "edited": null,
      "flags": [],
      "likes": [],
      "taskState": "comment",
      "time": 1461164347,
      "topic": "reviews/885",
      "updated": 1461164347,
      "user": "bruno"
    }
  },
  "lastSeen": 51
}
```

> **Note**
> **`lastSeen`** can often be used as an offset for pagination, by using the value in the **`after`** parameter of subsequent requests.

When no results are found, the `comments` array is empty:

```
HTTP/1.1 200 OK

{
  "topic": "jobs/job000011",
  "comments": [],
  "lastSeen": null
}
```

Examples of usage

Listing comments

To list comments:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/comments?topic=reviews/911&max=2&fields=id,body,time,user"
```

Swarm responds with a list of the first two comments for review 911 and a `lastSeen` value for pagination:

```
HTTP/1.1 200 OK

{
  "topic": "reviews/911",
  "comments": {
    "35": {
      "id": 35,
      "body": "Excitation thunder cats intelligent man braid organic
bitters.",
      "time": 1461164347,
      "user": "bruno"
    },
    "39": {
      "id": 39,
      "body": "Chamber tote bag butcher, shirk truffle mode shabby chic
single-origin coffee.",
      "time": 1461164347,
      "user": "swarm_user"
    }
  },
  "lastSeen": 39
}
```

Paginating a comment listing

To obtain the next page of a comments list (based on the previous example):

```
curl -u "username:password" "https://my-swarm-
host/api/
v9/comments?topic=reviews/911&max=2&fields=id,body,time,user&after=39"
```

Swarm responds with the second page of results, if any comments are present after the last seen comment:

```
HTTP/1.1 200 OK

{
```

```
  "topic": "reviews/911",
  "comments": {
    "260": {
      "id": 260,
      "body": "Reprehensible do lore flank ham hock.",
      "time": 1461164349,
      "user": "bruno"
    },
    "324": {
      "id": 324,
      "body": "Sinter lo-fi temporary, nihilist tote bag mustache swag
consequence interest flexible.",
      "time": 1461164349,
      "user": "bruno"
    }
  },
  "lastSeen": 324
}
```


Send notification for comments

Summary

Sends notification for comments

POST /api/v9/comments/notify

Description

Sends notification for comments.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`topic`	This is going to send a single notification for any comments that were not notified immediately for the user authenticated for a given topic they are posting for. Example: `reviews/1234`.	string	form	Yes
`preview`	Set to `true` to check if there are comments waiting for a notification to be sent, the notification is not sent.	string	form	No

Examples of successful responses

Comment notification sent.

```
HTTP/1.1 200 OK

{
  "isValid": true,
  "message": "Sending a notification for 3 comments",
  "code": 200
}
```

No comments are waiting for a notification.

```
HTTP/1.1 200 OK

{
   "isValid": true,
   "message": "No comment notifications to send",
   "code": 200
}
```

Examples of usage

Sending notification for comments

To send a notification for all delayed comments:

```
curl -u "username:password" \
     -X POST \
     -d "topic=reviews/911" \
     "https://my-swarm-host/api/v9/comments/notify"
```

Swarm responds with a message detailing a notification sent with the number of comments it applied to.

```
HTTP/1.1 200 OK

{
  "isValid": true,
  "message": "Sending a notification for 3 comments",
  "code": 200
}
```

Checking to see if any comments are waiting for a notification to be sent

To check if there are any comments awaiting a notification without sending the notification, include `"preview=true"` in your request.

```
curl -u "username:password" \
     -X POST \
     -d "topic=reviews/911" \
     -d "preview=true" \
     "https://my-swarm-host/api/v9/comments/notify"
```

Swarm responds with a message without sending a notification.

```
HTTP/1.1 200 OK

{
  "isValid": true,
  "message": "Sending a notification for 3 comments",
  "code": 200
  "preview": true
}
```

Add a Comment

Summary

Add a Comment

POST /api/v9/comments/

Description

Add a comment to a topic (such as a review or a job).

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`topic`	Topic to comment on. Examples: `reviews/1234`, `changes/1234` or `jobs/job001234`.	string	form	Yes	
`body`	Content of the comment.	string	form	Yes	
`silenceNotification`	If set to 'true' no notifications will ever be sent for this created comment	string	form	No	
`delayNotification`	If set to 'true' notifications will be delayed	string	form	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`taskState`	Optional task state of the comment. Valid values when adding a comment are `comment` and `open`. This creates a plain comment or opens a task, respectively.	string	form	No	comment
`flags[]`	Optional flags on the comment. Typically set to `closed` to archive a comment.	array (of strings)	form	No	
`context[file]`	File to comment on. Valid only for `changes` and `reviews` topics. Example: `//depot/main/README.txt`.	string	form	No	
`context[leftLine]`	Left-side diff line to attach the inline comment to. Valid only for `changes` and `reviews` topics. If this is specified, `context[file]` must also be specified.	integer	form	No	
`context[rightLine]`	Right-side diff line to attach the inline comment to. Valid only for `changes` and `reviews` topics. If this is specified, `context[file]` must also be specified.	integer	form	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`context[content]`	Optionally provide content of the specified line and its four preceding lines. This is used to specify a short excerpt of context in case the lines being commented on change during the review. When not provided, Swarm makes an effort to build the content on its own - as this involves file operations, it could become slow.	array (of strings)	form	No	
`context[version]`	With a `reviews` topic, this field specifies which version to attach the comment to.	integer	form	No	

Example of response

Successful Response contains Comment entity:

```
HTTP/1.1 200 OK

{
  "comment": {
    "id": 42,
    "attachments": [],
    "body": "Best. Comment. EVER!",
    "context": [],
    "edited": null,
    "flags": [],
    "likes": [],
    "taskState": "comment",
    "time": 123456789,
    "topic": "reviews/2",
    "updated": 123456790,
```

```
    "user": "bruno"
  }
}
```

Examples of usage

Create a comment on a review

To create a comment on a review:

```
curl -u "username:password" \
     -X POST \
     -d "topic=reviews/2" \
     -d "body=This is my comment. It is an excellent comment. It contains
a beginning, a middle, and an end." \
     "https://my-swarm-host/api/v9/comments"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "comment": {
    "id": 42,
    "attachments": [],
    "body": "This is my comment. It is an excellent comment. It contains a
beginning, a middle, and an end.",
    "context": [],
    "edited": null,
    "flags": [],
    "likes": [],
    "taskState": "comment",
    "time": 123456789,
    "topic": "reviews/2",
    "updated": 123456790,
    "user": "username"
  }
}
```

Open a task on a review

To create a comment on a review, and flag it as an open task:

```
curl -u "username:password" \
     -X POST \
     -d "topic=reviews/2" \
     -d "taskState=open" \
     -d "body=If you could go ahead and attach a cover page to your TPS
report, that would be great." \
     "https://my-swarm-host/api/v9/comments"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "comment": {
    "id": 43,
    "attachments": [],
    "body": "If you could go ahead and attach a cover page to your TPS
report, that would be great.",
    "context": [],
    "edited": null,
    "flags": [],
    "likes": [],
    "taskState": "open",
    "time": 123456789,
    "topic": "reviews/2",
    "updated": 123456790,
    "user": "username"
  }
}
```

Edit a Comment

Summary

Edit a Comment

PATCH /api/v9/comments/{id}

Description

Edit a comment.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	ID of the comment to be edited	integer	path	Yes
`topic`	Topic to comment on. Examples: `reviews/1234`, `changes/1234` or `jobs/job001234`.	string	form	No
`body`	Content of the comment.	string	form	Yes
`taskState`	Optional task state of the comment. Note that certain transitions (such as moving from `open` to `verified`) are not possible without an intermediate step (`addressed`, in this case). Examples: `comment` (not a task), `open`, `addressed`, `verified`.	string	form	No
`flags[]`	Optional flags on the comment. Typically set to `closed` to archive a comment.	array (of strings)	form	No

Parameter	Description	Type	Parameter Type	Required
`silenceNotification`	If set to 'true' no notifications will ever be sent for this edited comment	string	form	No
`delayNotification`	If set to 'true' notifications will be delayed	string	form	No

Example response

Successful Response contains Comment entity:

```
HTTP/1.1 200 OK

{
  "comment": {
    "id": 1,
    "attachments": [],
    "body": "Best. Comment. EVER!",
    "context": [],
    "edited": 123466790,
    "flags": [],
    "likes": [],
    "taskState": "comment",
    "time": 123456789,
    "topic": "reviews/42",
    "updated": 123456790,
    "user": "bruno"
  }
}
```

Examples of usage

Edit and archive a comment on a review

To edit and archive a comment on a review:

```
curl -u "username:password" \
     -X PATCH \
     -d "flags[]=closed" \
     -d "body=This comment wasn't as excellent as I may have lead you to
believe. A thousand apologies." \
     "https://my-swarm-host/api/v9/comments/42"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "comment": {
    "id": 42,
    "attachments": [],
    "body": "This comment wasn't as excellent as I may have lead you to
believe. A thousand apologies.",
    "context": [],
    "edited": 123466790,
    "flags": ["closed"],
    "likes": [],
    "taskState": "comment",
    "time": 123456789,
    "topic": "reviews/2",
    "updated": 123456790,
    "user": "username"
  }
}
```

Flag a task as addressed on a review

To flag an open task as addressed on a review:

```
curl -u "username:password" \
     -X PATCH \
     -d "taskState=addressed" \
     "https://my-swarm-host/api/v9/comments/43"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "comment": {
    "id": 43,
    "attachments": [],
    "body": "If you could go ahead and attach a cover page to your TPS
report, that would be great.",
    "context": [],
    "edited": 123466790,
    "flags": ["closed"],
    "likes": [],
    "taskState": "addressed",
    "time": 123456789,
    "topic": "reviews/2",
    "updated": 123456790,
    "user": "username"
  }
}
```

Groups : Swarm Groups

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get List of Groups

Summary

Get List of Groups

GET /api/v9/groups/

Description

Returns the complete list of groups in Swarm.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`after`	A group ID to seek to. Groups prior to and including the specified ID are excluded from the results and do not count towards `max`. Useful for pagination. Commonly set to the `lastSeen` property from a previous query.	string	query	No	
`max`	Maximum number of groups to return. This does not guarantee that `max` groups are returned. It does guarantee that the number of groups returned won't exceed `max`.	integer	query	No	100

Parameter	Description	Type	Parameter Type	Required	Default Value
`fields`	An optional comma-separated list (or array) of fields to show for each group. Omitting this parameter or passing an empty value shows all fields.	string	query	No	
`keywords`	Keywords to limit groups on. Only groups where the group ID, group name (if set), or description contain the specified keywords are returned.	string	query	No	
`ignoreExcludeList`	Determines if the list of groups has the group_exclude_list filter applied or not. Add the parameter to ignore the `group_exclude_list` filter.	boolean	query	No	

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "groups": [
    {
      "Group": "test-group",
      "MaxLockTime": null,
      "MaxResults": null,
      "MaxScanRows": null,
      "Owners": [],
      "PasswordTimeout": null,
      "Subgroups": [],
      "Timeout": 43200,
      "Users": ["bruno"],
      "config": {
        "description": "Our testing group",
        "emailAddress": "test-group3@host.domain",
        "emailFlags": [],
        "name": "Test Group",
        "useMailingList": true
      }
    }
  ]
}
```

Examples of usage

Listing groups

To list groups:

```
curl -u "username:password"
"https://myswarm.url/api/v9/groups?keywords=test-
group&fields=Group,Owners,Users,config&max=2"
```

Swarm responds with a list of groups:

```
HTTP/1.1 200 OK

{
  "groups": [
    {
      "Group": "test-group",
      "Owners": [],
      "Users": ["bruno"],
      "config": {
        "description": "Our testing group",
        "emailAddress": "test-group@host.domain",
        "emailFlags": {
          "reviews": "1",
          "commits": "0"
        },
        "name": "Test Group",
        "useMailingList: true
      }
    },
    {
      "Group": "test-group2",
      "Owners": [],
      "Users": ["bruno"],
      "config": {
        "description": "Our second testing group",
        "emailAddress": "test-group2@host.domain",
        "emailFlags": [],
        "name": "Test Group 2",
        "useMailingList: true
      }
    }
  ],
  "lastSeen": "test-group2"
}
```

Paginating the groups list

Based on the previous example, we can pass a lastSeen value of **`test-group2`** to see if there are any subsequent groups in Swarm.

```
curl -u "username:password"
"https://myswarm.url/api/v9/groups?keywords=test-
group&fields=Group,config&max=2&lastSeen=test-group2"
```

Swarm responds with a list of groups (minus the Owners and Users fields, as we haven't requested them):

```
HTTP/1.1 200 OK

{
  "groups": [
    {
      "Group": "test-group3",
      "config": {
        "description": "Our 3rd testing group",
        "emailAddress": "test-group3@host.domain",
        "emailFlags": [],
        "name": "Test Group 3",
        "useMailingList": true
      }
    }
  ],
  "lastSeen": "test-group3"
}
```

Get Group Information

Summary

Get Group Information

GET /api/v9/groups/{id}

Description

Retrieve information about a group.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Group ID	string	path	Yes
`fields`	An optional comma-separated list (or array) of fields to show for each group. Omitting this parameter or passing an empty value shows all fields.	string	query	No
`ignoreExcludeList`	Determines if the group_exclude_list filter is applied or not when getting the group information. If the filter is applied and the group specified is present in the `group_exclude_list` the group information is not included in the response. Add the parameter to ignore the `group_exclude_list` filter.	boolean	query	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "test-group",
    "MaxLockTime": null,
    "MaxResults": null,
    "MaxScanRows": null,
    "Owners": [],
```

```
    "PasswordTimeout": null,
    "Subgroups": [],
    "Timeout": 43200,
    "Users": ["bruno"],
    "config": {
      "description": "Our testing group",
      "emailFlags": [],
      "name": "Test Group"
    }
  }
}
```

Examples of usage

Fetching a group

To fetch an individual group:

```
curl -u "username:password" "https://myswarm.url/api/v9/groups/my-group"
```

Swarm responds with the group entity:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "test-group",
    "LdapConfig": null,
    "LdapSearchQuery": null,
    "LdapUserAttribute": null,
    "MaxLockTime": null,
    "MaxResults": null,
    "MaxScanRows": null,
    "Owners": [],
    "Users": ["bruno"],
    "config": {
      "description": "Our testing group",
      "emailAddress": "test-group@host.domain",
      "emailFlags": {
```

```
        "reviews": "1",
        "commits": "0"
      },
      "name": "Test Group",
      "useMailingList": true
    }
  }
}
```

Limiting returned fields

To limit the returned fields when fetching an individual group:

```
curl -u "username:password" "https://myswarm.url/api/v9/groups/my-group?fields=Group,Owners,Users,config"
```

Swarm responds with the group entity:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "test-group",
    "Owners": [],
    "Users": ["bruno"],
    "config": {
      "description": "Our testing group",
      "emailFlags": [],
      "name": "Test Group"
    }
  }
}
```

Create a new Group

Summary

Create a new Group

POST /api/v9/groups/

Description

Creates a new group in Swarm.

Parameters

Parameter	Description	Type	Parameter Type	Required
`Group`	Group identifier string.	string	form	Yes
`Users`	An optional array of group users. *At least one of Users, Owners, or Subgroups is required.	array	form	No *See Description
`Owners`	An optional array of group owners. *At least one of Users, Owners, or Subgroups is required.	array	form	No *See Description
`Subgroups`	An optional array of subgroups. *At least one of Users, Owners, or Subgroups is required.	array	form	No *See Description
`config[name]`	An optional full name for the group.	string	form	No
`config [description]`	An optional group description.	string	form	No
`config [emailAddress]`	The email address for this group.	string	form	No
`config [emailFlags] [reviews]`	Email members when a new review is requested.	boolean	form	No
`config [emailFlags] [commits]`	Email members when a change is committed.	boolean	form	No

Parameter	Description	Type	Parameter Type	Required
`config [useMailingList]`	Whether to use the configured email address or expand individual members addresses.	boolean	form	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "test-group",
    "MaxLockTime": null,
    "MaxResults": null,
    "MaxScanRows": null,
    "Owners": [],
    "PasswordTimeout": null,
    "Subgroups": [],
    "Timeout": null,
    "Users": ["alice"],
    "config": {
      "description": "Test test test",
      "emailAddress": "test-group@host.domain",
      "emailFlags": [],
      "name": "TestGroup",
      "useMailingList": true
    }
  }
}
```

Example of usage

Create a group

Important

- Only users with *super* privileges in the Helix Core server (`p4d`), or users with *admin* privileges in `p4d` versions 2012.1 or newer, can create groups.
- This API version is only capable of setting specific fields: `Group`, `Users`, `Owners`, `Subgroups`, `config`. Any other fields specified in the creation request are ignored.

To create a new group:

```
curl -u "username:password" \
     -X POST \
     -d "Group=my-group" \
     -d "Owners[]=alice" \
     -d "Owners[]=bob" \
     -d "Users[]=bruno" \
     -d "Users[]=user2" \
     -d "config[description]=This group is special to me." \
     -d "config[name]=My Group" \
     -d "config[emailFlags][reviews]=1" \
     -d "config[emailFlags][commits]=0" \
     -d "config[emailAddress]=my-group@host.domain" \
     -d "config[useMailingList]=false" \
     "https://myswarm.url/api/v9/groups"
```

Assuming that the authenticated user has permission, Swarm responds with the new group entity:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "my-group",
    "MaxLockTime": null,
    "MaxResults": null,
    "MaxScanRows": null,
    "Owners": ["username"],
    "PasswordTimeout": null,
```

```
    "Subgroups": [],
    "Timeout": null,
    "Users": [],
    "config": {
      "description": "This group is special to me.",
      "emailFlags": {
        "reviews": "1",
        "commits": "0"
      },
      "name": "My Group",
      "useMailingList": true
    }
  }
}
```

Edit a Group

Summary

Edit a Group

PATCH /api/v9/groups/{id}

Description

Change the settings of a group in Swarm. Only super users and group owners can perform this action.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Group ID	string	path	Yes
`Users`	An optional array of group users.	array	form	No
`Owners`	An optional array of group owners.	array	form	No
`Subgroups`	An optional array of group subgroups.	array	form	No

Parameter	Description	Type	Parameter Type	Required
`config[name]`	An optional full name for the group.	string	form	No
`config [description]`	An optional group description.	string	form	No
`config [emailAddress]`	The email address for this group.	string	form	No
`config [emailFlags] [commits]`	Email members when a change is committed.	boolean	form	No
`config [emailFlags] [reviews]`	Email members when a new review is requested.	boolean	form	No
`config [useMailingList]`	Whether to use the configured email address or expand individual members addresses.	boolean	form	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "test-group",
    "Users": [],
    "Owners": [],
    "Subgroups": [],
    "config": {
      "description": "New Group Description",
      "name": "TestGroup",
      "useMailingList": true
    }
```

```
  }
}
```

Example of usage

Editing a group

> **Important**
>
> - Only users with *super* privileges in the Helix Core server, or group owners, can edit groups.
> - This API version is only capable of modifying specific fields: **`Users`**, **`Owners`**, **`Subgroups`**, **`config`**. Any other fields specified in the edit request are ignored.

Here is how to update the **`name`**, **`description`**, and **`emailFlags`** configuration of the group **`my-group`**:

```
curl -u "username:password" \
     -X PATCH \
     -d "config[description]=This group is special to me." \
     -d "config[name]=My Group" \
     -d "config[emailFlags][commits]=1" \
     "https://myswarm.url/api/v9/groups/my-group"
```

Assuming that the authenticated user has permission, Swarm responds with the modified group entity:

```
HTTP/1.1 200 OK

{
  "group": {
    "Group": "my-group",
    "Users": [],
    "Owners": [],
    "Subgroups": [],
    "config": {
      "description": "This group is special to me.",
      "emailAddress": "test-group@host.domain",
      "emailFlags": {
        "reviews": "1",
        "commits": "1"
      },
```

```
      "name": "My Group",
      "useMailingList": true
    }
  }
}
```

Delete a Group

Summary

Delete a Group

DELETE /api/v9/groups/{id}

Description

Delete a group. Only super users and group owners can perform this action.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Group ID.	string	path	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "id": "test-group"
}
```

Example of usage

Deleting a group

> **Important**
> Only *super* users and group owners can delete groups.

```
curl -u "username:password" -X DELETE
"https://myswarm.url/api/v9/groups/my-group"
```

Assuming that the authenticated user has permission, Swarm responds with the `id` of the deleted group:

```
HTTP/1.1 200 OK

{
  "id": "my-group"
}
```

Index : Basic API controller providing a simple version action

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Show Version Information

Summary

Show Version Information

GET /api/v9/version

Description

This can be used to determine the currently-installed Swarm version, and also to check that Swarm's API is responding as expected.

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
    "year": "2019",
    "version": "SWARM/2018.3/1752978 (2019/01/31)"
}
```

Note
`year` refers to the year of the Swarm release, not necessarily the current year.

Login : Swarm Login API

Important
From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Checking the 2FA authentication

Summary

Checking the 2FA authentication

GET /api/v9/checkauth/

Description

Checking the 2FA authentication

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "results": {
```

```
    "trigger": "GAuth says yes!",
    "successMsg": "Second factor authentication approved."
  },
  "code": 200
}
```

Usage example

Checking the 2FA authentication

To Check User prompt input for 2FA

```
curl -u "username:password" "https://myswarm.url/api/v9/login/checkauth"
```

Assuming that the authenticated user has permission, Swarm responds with the next step in the 2FA login:

```
HTTP/1.1 200 OK

{
  "results": {
    "trigger": "GAuth says yes!",
    "successMsg": "Second factor authentication approved."
  },
  "code": 200
}
```

Get List of 2FA Methods

Summary

Get List of 2FA Methods

GET /api/v9/listmethods/

Description

Returns the complete list of methods of 2FA.

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "results": {
    "methods": {
      "1": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      },
      "2": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      },
      "3": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      },
      "4": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      }
    }
  },
  "option": {
    "persist": "option",
    "nextState": "init-auth"
  },
  "code": 200
}
```

Usage example

Listing 2FA Methods

To list the 2FA methods:

```
curl -u "username:password" "https://myswarm.url/api/v9/login/listmethods"
```

Swarm responds with a list of 2FA methods:

```
HTTP/1.1 200 OK

{
  "results": {
    "methods": {
      "1": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      },
      "2": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      },
      "3": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      },
      "4": {
        "methodName": "Method Name will be here",
        "methodDesc": "Method Description will be here"
      }
    }
  },
  "option": {
    "persist": "option",
    "nextState": "init-auth"
  },
  "code": 200
}
```

Get the current effective user details

Summary

Get the current effective user details

GET /api/v9/session

Description

Get user logged in

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
    "isValid": true,
    "messages": [],
    "user": {
        "User": "reviewer",
        "FullName": "Code Reviewer",
        "Email": "reviewer@swarm.local",
        "Type": "standard",
        "Password": "enabled"
    }
}
```

Usage example

Getting the currently effective user details

To get session details:

```
curl -u "<username>:<ticket>" "http://myswarm.url/api/v9/session"
```

Checking the 2FA authentication

POST /api/v9/checkauth/

Summary

Checking the 2FA authentication

Description

Checking the 2FA authentication

Parameters

Parameter	Description	Type	Parameter Type	Required
`token`	The token from the user for their 2FA prompt.	string	form	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "results": {
    "trigger": "otp-generated|||GAuth says yes!",
    "successMsg": "Second factor authentication approved."
  },
  "code": 200
}
```

Example usage

Checking the 2FA authentication

To Check User prompt input for 2FA

```
curl -u "username:password" \
     -X POST \
     -d "token=TOKEN" \
     "https://myswarm.url/api/v9/login/checkauth"
```

Assuming that the authenticated user has permission, Swarm responds with the next step in the 2FA login:

```
HTTP/1.1 200 OK

{
  "results": {
    "trigger": "GAuth says yes!",
    "successMsg": "Second factor authentication approved."
  },
  "code": 200
}
```

Initiating the 2FA authentication

Summary

Initiating the 2FA authentication

POST /api/v9/initauth/

Description

Initiating the 2FA authentication

Parameters

Parameter	Description	Type	Parameter Type	Required
method	The Method in which you want to use.	string	form	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "results": {
     "trigger": "TriggerName",
```

```
    "successMsg": "Message from Authentication method"
  },
  "option": {
    "prompt": true,
    "nextState": "check-auth"
  },
  "code": 200
}
```

Example usage

Initiating the 2FA authentication

To Initiate the user 2FA login:

```
curl -u "username:password" \
     -X POST
     -d "method=METHOD" \
     "https://myswarm.url/api/v9/login/initauth"
```

Assuming that the authenticated user has permission, Swarm responds with the next step in the 2FA login:

```
HTTP/1.1 200 OK

{
  "results": {
    "trigger": "TriggerName",
    "successMsg": "Message from Authentication method"
  },
  "option": {
    "prompt": true,
    "nextState": "check-auth"
  },
  "code": 200
}
```

Login to Swarm

Summary

Login to Swarm

POST /api/v9/login/

Description

Login to Swarm

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
     "isValid": true,
     "messages": [],
      "user": {
          "User": "swarm.user",
          "FullName": "Swarm User",
          "Email": "swarm.user@mydomain.com",
          "Type": "standard",
          "Password": "enabled",
          "isAdmin": false,
          "isSuper": false
     }
}
```

> **Note**
> In the event of a failed login attempt Swarm responds with:

Example usage

Logging in to swarm

To login:

```
curl -H "Content-Type: application/json" \
     -X POST \
     -u "super:<ticket>" \
     -d '{"username":"swarm.user","password":"1234"}'
"http://myswarm.url/api/v9/login"
```

Login to Swarm with SAML

Summary

Login to Swarm with SAML

POST /api/v9/login/saml

Description

Login to Swarm with SAML

Parameters

Parameter	Description	Type	Parameter Type	Required
`redirect`	Options are: ■ `redirect=true` or not specified: Swarm redirects the user to the `HTTP_REFERER` url or to the specified custom "logout_url" on page 498 if it has been set. ■ `redirect=false`: Swarm does not redirect the user.	string	query	No

Example usage

Logging in to swarm with SAML

```
curl -u "super:<ticket>" \
     -X POST \
     "http://myswarm.url/api/v9/login/saml"
```

JSON response:

```
HTTP/1.1 302 OK

{
   "isValid": "true"
   "url": "<url to redirect to>"
}
```

Logging in to swarm with SAML and redirect=false

```
curl -u "super:<ticket>" \
     -X POST \
     "http://myswarm.url/api/v9/login/saml?redirect=false"
```

JSON response:

```
HTTP/1.1 200 OK

{
   "isValid": "true",
}
```

Logout of Swarm with optional redirect

Summary

Logout of Swarm with optional redirect

POST /api/v9/logout/

Description

Logout of Swarm

Examples responses

Successful Response:

```
HTTP/1.1 302 OK

{
     "isValid": true,
```

```
     "messages": []
}
```

Successful Response:

```
HTTP/1.1 200 OK

{
     "isValid": true,
     "messages": []
}
```

> **Note**
> In the event of a failed login attempt Swarm responds with:

```
HTTP/1.1 200 OK

{
"isValid": false,
"messages": ["Error message."]
}
```

Example usage

Logout of Swarm

To logout:

```
curl -X POST \
     -u "super:<ticket>" \
     "http://myswarm.url/api/v9/logout"
```

Logout of Swarm without redirect

To logout without any redirect:

```
curl -X POST \
     -u "super:<ticket>" \
     "http://myswarm.url/api/v9/logout?stay=true"
```

Create a new Swarm session using the given credentials

Summary

Create a new Swarm session using the given credentials

POST /api/v9/session

Description

Login to swarm

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
    "isValid": true,
    "messages": [],
    "user": {
        "User": "reviewer",
        "FullName": "Code Reviewer",
        "Email": "reviewer@swarm.local",
        "Type": "standard",
        "Password": "enabled"
    }
}
```

Example usage

Logging in to Swarm

To login:

```
curl -H "Content-Type: application/json" \
     -X POST \
     -d '{"username": "<username>","password": "<password>","remember":
"false"}' \
     -X POST http://localhost/api/v9/session
```

Destroy the current session, for instance logout

Summary

Destroy the current session, for instance logout

DELETE /api/v9/session

Description

Logout of Swarm

Example response

Successful Response:

```
HTTP/1.1 200 OK


{
    "isValid": true,
    "messages": []
}
```

Example usage

Logging out of swarm

To login:

```
curl -u "<username>:<ticket>" -X DELETE
"http://myswarm.url/api/v9/session"
```

Projects : Swarm Projects

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get List of Projects

Summary

Get List of Projects

GET /api/v9/projects/

Description

Returns a list of projects in Swarm that are visible to the current user. Administrators will see all projects, including private ones.

> **Tip**
>
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each project. Omitting this parameter or passing an empty value shows all fields.	string	query	No
`workflow`	An optional parameter to only list projects using a workflow.	string	query	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "jplugin",
```

```
  "branches": [
    {
      "id": "main",
      "name": "MAIN",
      "workflow": null,
      "paths": [
        "//projects/jplugin/main/..."
      ],
      "defaults": {
        "reviewers": {
          "users": {
            "steve.russell": [ ]
          }
        }
      },
      "minimumUpVotes": null,
      "retainDefaultReviewers": false,
      "moderators": [
        "claire.brevia"
      ],
      "moderators-groups": [ ]
    },
    {
      "id": "candidate",
      "name": "CANDIDATE",
      "workflow": null,
      "paths": [
        "//projects/jplugin/candidate/..."
      ],
      "defaults": {
        "reviewers": [ ]
      },
      "minimumUpVotes": null,
      "retainDefaultReviewers": false,
      "moderators": [ ],
```

```
      "moderators-groups": [ ]
    }
  ],
  "defaults": {
    "reviewers": {
      "users": {
        "allison.clayborne": {
          "required": true
        },
        "jack.boone": [ ],
        "steve.russell": [ ]
      }
    }
  },
  "deleted": false,
  "deploy": {
    "enabled": false,
    "url": ""
  },
  "description": "A Java plugin for continuous integration.",
  "emailFlags": {
    "change_email_project_users": "1",
    "review_email_project_members": "1"
  },
  "jobview": "",
  "members": [
    "allison.clayborne",
    "jack.boone",
    "steve.russell"
  ],
  "minimumUpVotes": 1,
  "name": "JPlugin",
  "owners": [
    "allison.clayborne"
  ],
```

```
    "private": false,
    "retainDefaultReviewers": false,
    "subgroups": [ ],
    "tests": {
      "enabled": false,
      "url": "",
      "postBody": "",
      "postFormat": "URL"
    },
    "workflow": null
  },
  "readme":<h1>P4 Plugin</h1>\n<p>Jenkins plugin for a Perforce Helix
Versioning Engine (P4D).</p>\n<h2>Contents</h2>\n<ul>\n<li><a
href=\"https://github.com/jenkinsci/p4-
plugin/blob/master/RELEASE.md\">Release notes</a></li>\n\">
}
```

Example usage

Listing projects

To list all projects:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/projects?fields=id,description,members,name"
```

Pagination is not currently supported by this endpoint. Swarm responds with a list of all projects:

```
HTTP/1.1 200 OK

{
  "projects": [
    {
      "id": "blue-book",
      "description": "This is a private project for use only by users with
sufficient clearance.",
      "members": [
        "alex.randolph",
        "allison.clayborne"
```

```
      ],
      "name": "Blue Book"
    },
    {
      "id": "jplugin",
      "description": "A Java plugin for continuous integration.",
      "members": [
        "allison.clayborne",
        "jack.boone",
        "steve.russell"
      ],
      "name": "JPlugin"
    },
    {
      "id": "mercury",
      "description": "Mercury project for code management.",
      "members": [
        "alex.randolph",
        "claire.brevia",
        "jack.boone"
      ],
      "name": "Mercury"
    },
    {
      "id": "test-data",
      "description": "",
      "members": [
        "steve.russell"
      ],
      "name": "Test Data"
    },
    {
      "id": "test-project-for-workflow",
      "description": "This project has been created to test Swarm
workflow.",
```

```
      "members": [
        "alex.randolph",
        "allison.clayborne",
        "claire.brevia",
        "jack.boone",
        "paula.boyle",
        "steve.russell"
      ],
      "name": "Test project for workflow"
    }
  ]
}
```

Project administrators wishing to see the `tests` and `deploy` fields must fetch projects individually.

Get Project Information

Summary

Get Project Information

GET /api/v9/projects/{id}

Description

Retrieve information about a project.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Project ID	string	path	Yes

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each project. Omitting this parameter or passing an empty value shows all fields.	string	query	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "jplugin",
    "branches": [
      {
        "id": "main",
        "name": "MAIN",
        "workflow": null,
        "paths": [
          "//projects/jplugin/main/..."
        ],
        "defaults": {
          "reviewers": {
            "users": {
              "steve.russell": [ ]
            }
          }
        },
        "minimumUpVotes": null,
        "retainDefaultReviewers": false,
        "moderators": [
          "claire.brevia"
        ],
        "moderators-groups": [ ]
```

```
    },
    {
      "id": "candidate",
      "name": "CANDIDATE",
      "workflow": null,
      "paths": [
        "//projects/jplugin/candidate/..."
      ],
      "defaults": {
        "reviewers": [ ]
      },
      "minimumUpVotes": null,
      "retainDefaultReviewers": false,
      "moderators": [ ],
      "moderators-groups": [ ]
    }
  ],
  "defaults": {
    "reviewers": {
      "users": {
        "allison.clayborne": {
          "required": true
        },
        "jack.boone": [ ],
        "steve.russell": [ ]
      }
    }
  },
  "deleted": false,
  "deploy": {
    "enabled": false,
    "url": ""
  },
  "description": "A Java plugin for continuous integration.",
  "emailFlags": {
```

```
      "change_email_project_users": "1",
      "review_email_project_members": "1"
    },
    "jobview": "",
    "members": [
      "allison.clayborne",
      "jack.boone",
      "steve.russell"
    ],
    "minimumUpVotes": 1,
    "name": "JPlugin",
    "owners": [
      "allison.clayborne"
    ],
    "private": false,
    "retainDefaultReviewers": false,
    "subgroups": [ ],
    "tests": {
      "enabled": false,
      "url": "",
      "postBody": "",
      "postFormat": "URL"
    },
    "workflow": null
  },
  "readme":<h1>P4 Plugin</h1>\n<p>Jenkins plugin for a Perforce Helix
Versioning Engine (P4D).</p>\n<h2>Contents</h2>\n<ul>\n<li><a
href=\"https://github.com/jenkinsci/p4-
plugin/blob/master/RELEASE.md\">Release notes</a></li>\n
}
```


Example usage

Fetching a project

To fetch an individual project:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/projects/testproject2?fields=id,description,members,name"
```

Swarm responds with a project entity:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "jplugin",
    "description": "A Java plugin for continuous integration.",
    "members": [
      "allison.clayborne",
      "jack.boone",
      "steve.russell"
    ],
    "name": "JPlugin"
  }
}
```

Project administrators have access to additional fields (**`tests`** and **`deploy`**) when fetching individual projects using this endpoint.

Create a new Project

Summary

Create a new Project

POST /api/v9/projects/

Description

Creates a new project in Swarm.

Parameters

Parameter	Description	Type	Parameter Type	Required
`name`	Project Name (is also used to generate the Project ID)	string	form	Yes
`members`	An array of project members.	array	form	Yes
`subgroups`	An optional array of project subgroups.	array	form	No
`owners`	An optional array of project owners.	array	form	No
`description`	An optional project description.	string	form	No
`private`	Private projects are visible only to Members, Moderators, Owners, and Administrators. (Default: false)	boolean	form	No
`deploy`	Configuration for automated deployment. Example: {"enabled": true, "url": "http://localhost/?change={change}"}	array	form	No
`tests`	Configuration for testing/continuous integration.	array	form	No
`branches`	Optional branch definitions for this project.	array	form	No
`jobview`	An optional jobview for associating certain jobs with this project.	string	form	No

Parameter	Description	Type	Parameter Type	Required
`emailFlags[change_email_project_users]`	Email members, moderators and followers when a change is committed.	boolean	form	No
`emailFlags[review_email_project_members]`	Email members and moderators when a new review is requested.	boolean	form	No
`defaults`	An optional array of defaults at a project level (for example default reviewers).	array	form	No
`retainDefaultReviewers`	An optional to retain the default reviewers.	boolean	form	No
`minimumUpVotes`	An optional to set the minimum number of up votes required.	string	form	No

Example responses

Successful Response:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "testproject",
    "branches": [ ],
    "defaults": {
      "reviewers": [ ]
    },
    "deleted": false,
    "deploy": {
      "enabled": false,
      "url": null
    },
    "description": "My project is great",
```

```
    "emailFlags": [ ],
    "jobview": null,
    "members": [
      "bruno"
    ],
    "minimumUpVotes": null,
    "name": "testProject",
    "owners": [ ],
    "private": false,
    "retainDefaultReviewers": false,
    "subgroups": [ ],
    "tests": {
      "enabled": false,
      "url": null
    },
    "workflow": null
  },
  "readme": "",
  "mode": "add"
}
```

Successful Response:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "project-x",
    "branches": [ ],
    "defaults": {
      "reviewers": {
        "users": {
          "bruno": {
            "required": "1"
          }
        }
```

```
      }
    },
    "deleted": false,
    "deploy": {
      "enabled": false,
      "url": null
    },
    "description": "This project is temporary until project Z replaces
it",
    "emailFlags": [ ],
    "jobview": null,
    "members": [
      "swarm"
    ],
    "minimumUpVotes": null,
    "name": "project x",
    "owners": [ ],
    "private": false,
    "retainDefaultReviewers": false,
    "subgroups": [ ],
    "tests": {
      "enabled": false,
      "url": null
    },
    "workflow": null
  },
  "readme": "",
  "mode": "add"
}
```


Example usage

Creating a new project

To create a project:

```
curl -u "username:password" \
     -X POST \
     -d "name=TestProject 3" \
     -d "description=The third iteration of our test project." \
     -d "members[]=alice" \
     -d "members[]=bob" \
     "https://my-swarm-host/api/v9/projects/"
```

Swarm responds with the new project entity:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "testproject-3",
    "branches": [ ],
    "defaults": {
      "reviewers": [ ]
    },
    "deleted": false,
    "deploy": {
      "enabled": false,
      "url": null
    },
    "description": "The third iteration of our test project.",
    "emailFlags": [ ],
    "jobview": null,
    "members": [
      "bruno",
      "swarm"
    ],
    "minimumUpVotes": null,
    "name": "TestProject 3",
    "owners": [ ],
    "private": false,
    "retainDefaultReviewers": false,
    "subgroups": [ ],
```

```
    "tests": {
      "enabled": false,
      "url": null
    },
    "workflow": null
  },
  "readme": "",
  "mode": "add"
}
```

Creating a private project with branches

Specifying a branch requires using array notation and providing at least two fields (`name` and `paths`) for each branch you wish to create. Creating more than one branch involves incrementing the `branches[0]` specifier for each branch - an example of this accompanies the PATCH endpoint documentation.

Projects are public by default. To mark a project as private, set the `private` parameter to `true`.

```
curl -u "username:password" \
     -X POST \
     -d "name=TestProject 4" \
     -d "description=The 4th iteration of our test project." \
     -d "private=true" \
     -d "members[]=bob" \
     -d "branches[0][name]=Branch One" \
     -d "branches[0][paths][]=//depot/main/TestProject/..." \
     "https://my-swarm-host/api/v9/projects"
```

Swarm responds with the new project entity:

```
HTTP/1.1 200 OK

{
  "project": {
    "id": "testproject-4",
    "branches": [ ],
    "defaults": {
      "reviewers": [ ]
    },
    "deleted": false,
```

```
    "deploy": {
      "enabled": false,
      "url": null
    },
    "description": "The 4th iteration of our test project.",
    "emailFlags": [ ],
    "jobview": null,
    "members": [
      "bruno"
    ],
    "minimumUpVotes": null,
    "name": "TestProject 4",
    "owners": [ ],
    "private": true,
    "retainDefaultReviewers": false,
    "subgroups": [ ],
    "tests": {
      "enabled": false,
      "url": null
    },
    "workflow": null
  },
  "readme": "",
  "mode": "add"
}
```


Edit a Project

Summary

Edit a Project

PATCH /api/v9/projects/{id}

Description

Change the settings of a project in Swarm. If a project has owners set, only the owners can perform this action.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Project ID	string	path	Yes
`name`	Project Name (changing the project name does not change the project ID)	string	form	No
`members`	An array of project members.	array	form	No
`subgroups`	An optional array of project subgroups.	array	form	No
`owners`	An optional array of project owners.	array	form	No
`description`	Your project description.	string	form	No
`private`	Private projects are visible only to Members, Moderators, Owners, and Administrators. (Default: false)	boolean	form	No
`deploy`	Configuration for automated deployment. Example: {"enabled": true, "url": "http://localhost/?change={change}"}	array	form	No

Parameter	Description	Type	Parameter Type	Required
`tests`	Configuration for testing/continuous integration.	array	form	No
`branches`	Optional branch definitions for this project.	array	form	No
`jobview`	A jobview for associating certain jobs with this project.	string	form	No
`emailFlags[change_email_project_users]`	Email members, moderators and followers when a change is committed.	boolean	form	No
`emailFlags[review_email_project_members]`	Email members and moderators when a new review is requested.	boolean	form	No
`defaults`	An optional array of defaults at a project level (for example default reviewers).	array	form	No
`retainDefaultReviewers`	An optional to retain the default reviewers.	boolean	form	No
`minimumUpVotes`	An optional to set the minimum number of up votes required.	string	form	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "project":{
    "id":"project1",
    "branches":[
```

```
    {
      "name":"main",
      "paths":[
        "//depot/dev/APIProject/..."
      ],
      "id":"main",
      "moderators":[ ],
      "moderators-groups":[ ],
      "defaults":{
        "reviewers":[ ]
      },
      "workflow":null,
      "retainDefaultReviewers":false,
      "minimumUpVotes":null
    }
  ],
  "defaults":{
    "reviewers":[ ]
  },
  "deleted":false,
  "deploy":{
    "enabled":false,
    "url":""
  },
  "description":"Updated description",
  "emailFlags":[ ],
  "jobview":"",
  "members":[
    "swarm-admin"
  ],
  "minimumUpVotes":null,
  "name":"API Project",
  "owners":[ ],
  "private":false,
  "retainDefaultReviewers":false,
```

```
    "subgroups":[ ],
    "tests":{
      "enabled":false,
      "url":""
    },
    "workflow":null
  },
  "readme":"",
  "mode":"edit"
}
```

Example usage

Editing a project

To edit a project:

> **Note**
> It is safe to edit a project without specifying branches, but the instructions for adding branches contain important information for modifying branch configuration.

```
curl -u "username:password" \
     -X PATCH
     -d "description=Witness the power of a fully operational Swarm
project." \
     "https://my-swarm-host/api/v9/projects/jam"
```

Swarm responds with the updated project entity:

```
HTTP/1.1 200 OK

{
  "project":{
    "id":"jam",
    "branches":[
      {
        "id":"main",
        "name":"Main",
        "workflow":null,
```

```
        "paths":[
          "\/\/depot\/Jam\/MAIN\/..."
        ],
        "defaults":{
          "reviewers":[ ]
        },
        "minimumUpVotes":null,
        "retainDefaultReviewers":false,
        "moderators":[ ],
        "moderators-groups":[ ]
      },
      {
        "id":"release-2-1",
        "name":"Release 2.1",
        "workflow":null,
        "paths":[
          "\/\/depot\/Jam\/REL2.1\/..."
        ],
        "defaults":{
          "reviewers":[ ]
        },
        "minimumUpVotes":null,
        "retainDefaultReviewers":false,
        "moderators":[ ],
        "moderators-groups":[ ]
      },
      {
        "id":"release-2-2",
        "name":"Release 2.2",
        "workflow":null,
        "paths":[
          "\/\/depot\/Jam\/REL2.2\/..."
        ],
        "defaults":{
          "reviewers":[ ]
```

```
        },
        "minimumUpVotes":null,
        "retainDefaultReviewers":false,
        "moderators":[ ],
        "moderators-groups":[ ]
      }
    ],
    "defaults":{
      "reviewers":[ ]
    },
    "deleted":false,
    "deploy":{
      "enabled":false,
      "url":""
    },
    "description":"Witness the power of a fully operational Swarm
project.",
    "emailFlags":{
      "change_email_project_users":"1",
      "review_email_project_members":"1"
    },
    "jobview":"",
    "members":[
      "alex_qc",
      "pubuser"
    ],
    "minimumUpVotes":null,
    "name":"Jam",
    "owners":[
      "bruno"
    ],
    "private":false,
    "retainDefaultReviewers":false,
    "subgroups":[
      "Administrators"
```

```
    ],
    "tests":{
      "enabled":false,
      "url":"",
      "postBody":"",
      "postFormat":"URL"
    },
    "workflow":"1"
  },
  "readme":"",
  "mode":"edit"
}
```

Editing a project to add a moderated branch and make the project public

Specifying a branch requires using array notation and providing at least two fields (**`name`** and **`paths`**) for each branch you wish to create. Creating more than one branch involves incrementing the **`branches[0]`** specifier for each branch.

> **Important**
> If you have existing branches, you must specify all of them in the query to avoid data loss. This operation sets the value of the entire **`branches`** property to match the provided input.

To change a private project to public, set the **`private`** parameter to **`false`**.

```
curl -u "username:password" \
     -X PATCH \
     -d "private=false" \
     -d "branches[0][name]=Branch One" \
     -d "branches[0][paths][]=//depot/main/TestProject/..." \
     -d "branches[1][name]=Branch Two" \
     -d "branches[1][paths][]=//depot/main/SecondBranch/..." \
     -d "branches[1][moderators][]=bob" \
     -d "branches[1][moderators-groups][]=group1" \
     "https://my-swarm-host/api/v9/projects/testproject-4"
```

Swarm responds with the new project entity:

```
HTTP/1.1 200 OK
```

```
{
  "project":{
    "id":"testproject-4",
    "branches":[
      {
        "name":"Branch One",
        "paths":[
          "//depot/main/TestProject/..."
        ],
        "id":"branch-one",
        "moderators":[ ],
        "moderators-groups":[ ],
        "defaults":{
          "reviewers":[ ]
        },
        "workflow":null,
        "retainDefaultReviewers":false,
        "minimumUpVotes":null
      },
      {
        "name":"Branch Two",
        "paths":[
          "//depot/main/SecondBranch/..."
        ],
        "moderators":[
          "non-admin"
        ],
        "moderators-groups":[
          "group1"
        ],
        "id":"branch-two",
        "defaults":{
          "reviewers":[ ]
        },
        "workflow":null,
```

```
        "retainDefaultReviewers":false,
        "minimumUpVotes":null
      }
    ],
    "defaults":{
      "reviewers":[ ]
    },
    "deleted":false,
    "deploy":{
      "enabled":false,
      "url":""
    },
    "description":"Updated description",
    "emailFlags":[ ],
    "jobview":null,
    "members":[
      "non-admin"
    ],
    "minimumUpVotes":null,
    "name":"testproject-4",
    "owners":[ ],
    "private":false,
    "retainDefaultReviewers":false,
    "subgroups":[ ],
    "tests":{
      "enabled":false,
      "url":""
    },
    "workflow":null
  },
  "readme":"",
  "mode":"edit"
}
```

Editing a project to add default reviewers

```
curl -u "<user>:<password>" \
     -H "Content-Type: application/json" \
     -X PATCH \
     -d '"defaults":{"reviewers":{"swarm_admin":{"required":true}} \
         "branches":[{ \
           "name":"dev", \
           "paths":{"//depot/dev/API Project/..."} \
           "defaults": { \
               "reviewers":{ \
                   "groups":{"group1":{"required":"1"}}, \
                   "users":{"non_admin":{"required":true}} \
               } \
           }]' \
     "https://my-swarm-host/api/v9/projects/API Project"
```

Or without JSON content:

```
curl -u "username:password" \
     -X PATCH \
     -d "name=Udated description" \
     -d "defaults[reviewers][swarm_admin][required]=true" \
     -d "branches[0][name]=dev" \
     -d "branches[0][paths][]=//depot/dev/API Project/..." \
     -d "branches[0][defaults][reviewers][groups][group1][required]=1" \
     -d "branches[0][defaults][reviewers][users][non_admin]
[required]=true" \
     "https://my-swarm-host/api/v9/projects/API Project"
```

Swarm responds with project entity similar to:

```
HTTP/1.1 200 OK

{
  "project":{
    "id":"api-project",
    "branches":[
      {
        "name":"dev",
```

```
        "paths":[
          "//depot/dev/APIProject/..."
        ],
        "defaults":{
          "reviewers":{
            "users":{
              "non-admin":{
                "required":true
              }
            },
            "groups":{
              "group1":{
                "required":"1"
              }
            }
          }
        },
        "id":"dev",
        "moderators":[ ],
        "moderators-groups":[ ],
        "workflow":null,
        "retainDefaultReviewers":false,
        "minimumUpVotes":null
      }
    ],
    "defaults":{
      "reviewers":{
        "users":{
          "swarm-admin":{
            "required":true
          }
        }
      }
    },
    "deleted":false,
```

```
    "deploy":{
      "enabled":false,
      "url":""
    },
    "description":"Updated description",
    "emailFlags":[ ],
    "jobview":null,
    "members":[
      "swarm-admin"
    ],
    "minimumUpVotes":null,
    "name":"API Project",
    "owners":[ ],
    "private":false,
    "retainDefaultReviewers":false,
    "subgroups":[ ],
    "tests":{
      "enabled":false,
      "url":""
    },
    "workflow":null
  },
  "readme":"",
  "mode":"edit"
}
```


Delete a Project

Summary

Delete a Project

DELETE /api/v9/projects/{id}

Description

Mark a Swarm project as deleted. The project ID and name cannot be reused. If a project has owners set, only the owners can perform this action.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Project ID	string	path	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "id": "testproject"
}
```

Example usage

Deleting a project:

Super users, administrators, and owners can delete projects. Members can delete projects that have no owners set.

```
curl -u "username:password" -X DELETE "https://my-swarm-
host/api/v9/projects/testproject3"
```

Assuming that the authenticated user has permission, Swarm responds with the id of the deleted project:

```
HTTP/1.1 200 OK

{
```

```
  "id": "testproject3"
}
```

Reviews : Swarm Reviews

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get reviews for action dashboard

Summary

Get reviews for action dashboard

GET /api/v9/dashboards/action

Description

Gets reviews for the action dashboard for the authenticated user.

> **Tip**
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Example response

Successful Commit contains Review and Commit Entities:

```
HTTP/1.1 200 OK

{
  "lastSeen": 120,
  "reviews": [
    {
      "id": 7,
```

```
      "author": "swarm_admin",
      "changes": [6],
      "comments": [0,0],
      "commits": [6],
      "commitStatus": [],
      "created": 1485793976,
      "deployDetails": [],
      "deployStatus": null,
      "description": "test\n",
      "groups": ["swarm-project-test"],
      "participants": {"swarm_admin":[]},
      "pending": false,
      "projects": {"test":["test"]},
      "roles": ["moderator|reviewer|required_reviewer|author"],
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1485958875,
      "updateDate": "2017-02-01T06:21:15-08:00"
    }
  ],
  "totalCount": null
}
```


Example usage

Getting reviews for the action dashboard

To list reviews:

```
curl -u "username:password" "http://my-swarm-
host/api/v9/dashboards/action"
```

Swarm responds with a list of the latest reviews, a **`totalCount`** field, and a **`lastSeen`** value for pagination:

```
HTTP/1.1 200 OK

{
  "lastSeen": 120,
  "reviews": [
    {
      "id": 7,
      "author": "swarm_admin",
      "changes": [6],
      "comments": [0,0],
      "commits": [6],
      "commitStatus": [],
      "created": 1485793976,
      "deployDetails": [],
      "deployStatus": null,
      "description": "test\n",
      "groups": ["swarm-project-test"],
      "participants": {"swarm_admin":[]},
      "pending": false,
      "projects": {"test":["test"]},
      "roles": ["moderator|reviewer|required_reviewer|author"],
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1485958875,
      "updateDate": "2017-02-01T06:21:15-08:00"
    }
  ],
  "totalCount": null
}
```

Get List of Reviews

Summary

Get List of Reviews

GET /api/v9/reviews/

Description

List and optionally filter reviews.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`after`	A review ID to seek to. Reviews up to and including the specified `id` are excluded from the results and do not count towards `max`. Useful for pagination. Commonly set to the `lastSeen` property from a previous query.	integer	query	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`max`	Maximum number of reviews to return. This does not guarantee that `max` reviews are returned. It does guarantee that the number of reviews returned won't exceed `max`. Server-side filtering may exclude some reviews for permissions reasons.	integer	query	No	1000
`fields`	An optional comma-separated list (or array) of fields to show. Omitting this parameter or passing an empty value shows all fields.	string	query	No	
`author[]`	One or more authors to limit reviews by. Reviews with any of the specified authors are returned. (v1.2+)	array (of strings)	query	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`change[]`	One or more change IDs to limit reviews by. Reviews associated with any of the specified changes are returned.	array (of integers)	query	No	
`hasReviewers`	Boolean option to limit to reviews to those with or without reviewers. Use `true` or `false` for JSON-encoded data, `1` for true and or `0` for false for form-encoded data. The presence of the parameter without a value is evaluated as true.	boolean	query	No	
`ids[]`	One or more review IDs to fetch. Only the specified reviews are returned. This filter cannot be combined with the `max` parameter.	array (of integers)	query	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`keywords`	Keywords to limit reviews by. Only reviews where the description, participants list or project list contain the specified keywords are returned.	string	query	No	
`participants[]`	One or more participants to limit reviews by. Reviews with any of the specified participants are returned.	array (of strings)	query	No	
`project[]`	One or more projects to limit reviews by. Reviews affecting any of the specified projects are returned.	array (of strings)	query	No	
`state[]`	One or more states to limit reviews by. Reviews in any of the specified states are returned.	array (of strings)	query	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`passesTests`	Boolean option to limit reviews by tests passing or failing. Use `true` or `false` for JSON-encoded data, `1` for true and `0` for false for form-encoded data. The presence of the parameter without a value is evaluated as true.	string	query	No	
`notUpdatedSince`	Option to fetch unchanged reviews. Requires the date to be in the format YYYY-mm-dd, for example 2017-01-01. Reviews to be returned are determined by looking at the last updated date of the review.	string	query	No	
`hasVoted`	Should have the value 'up' or 'down' to filter reviews that have been voted up or down by the current authenticated user.	string	query	No	

Parameter	Description	Type	Parameter Type	Required	Default Value
`myComments`	True or false to support filtering reviews that include comments by the current authenticated user.	boolean	query	No	

Examples responses

Successful Response:

```
HTTP/1.1 200 OK

{
  "lastSeen": 12209,
  "reviews": [
    {
      "id": 12206,
      "author": "swarm",
      "changes": [12205],
      "comments": 0,
      "commits": [],
      "commitStatus": [],
      "created": 1402507043,
      "deployDetails": [],
      "deployStatus": null,
      "description": "Review Description\n",
      "participants": {
        "swarm": []
      },
      "pending": true,
      "projects": [],
      "state": "needsReview",
      "stateLabel": "Needs Review",
```

```
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1402518492
    }
  ],
  "totalCount": 1
}
```

Note
Swarm returns `null` for `totalCount` if no search filters were provided. `lastSeen` can often be used as an offset for pagination, by using the value in the `after` parameter of subsequent requests.

When no results are found, the `reviews` array is empty:

```
HTTP/1.1 200 OK

{
  "lastSeen": null,
  "reviews": [],
  "totalCount": 0
}
```

Example usage

Listing reviews

To list reviews:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/reviews?max=2&fields=id,description,author,state"
```

Swarm responds with a list of the latest reviews, a `totalCount` field, and a `lastSeen` value for pagination:

```
HTTP/1.1 200 OK

{
  "lastSeen": 120,
  "reviews": [
```

```
    {
      "id": 123,
      "author": "bruno",
      "description": "Adding .jar that should have been included in
r110\n",
      "state": "needsReview"
    },
    {
      "id": 120,
      "author": "bruno",
      "description": "Fixing a typo.\n",
      "state": "needsReview"
    }
  ],
  "totalCount": null
}
```

The **totalCount** field is populated when keywords are supplied. It indicates how many total matches there are. If keywords are not supplied the **totalCount** field remains **null**, indicating that the list of all reviews is being queried.

Paginating a review listing

To obtain the next page of a reviews list (based on the previous example):

```
curl -u "username:password" "https://my-swarm-
host/api/v9/reviews?max=2&fields=id,description,author,state&after=120"
```

Swarm responds with the second page of results, if any reviews are present after the last seen review:

```
HTTP/1.1 200 OK

{
  "lastSeen": 100,
  "reviews": [
    {
      "id": 110,
      "author": "bruno",
      "description": "Updating Java files\n",
      "state": "needsReview"
```

```
    },
    {
      "id": 100,
      "author": "bruno",
      "description": "Marketing materials for our new cutting-edge
product\n",
      "state": "needsReview"
    }
  ],
  "totalCount": null
}
```

Finding reviews for a change or a list of changes

Given a list of change IDs (5, 6, 7), here is how to check if any of them have reviews attached:

```
curl -u "username:password" "https://my-swarm-
host/api/
v9/reviews?max=2&fields=id,changes,description,author,state&change\
[\]=5&change\[\]=6&change\[\]=7"
```

Swarm responds with a list of reviews that include these changes:

```
HTTP/1.1 200 OK

{
  "lastSeen": 100,
  "reviews": [
    {
      "id": 110,
      "author": "bruno",
      "changes": [5],
      "description": "Updating Java files\n",
      "state": "needsReview"
    },
    {
      "id": 100,
      "author": "bruno",
      "changes": [6,7],
```

```
      "description": "Marketing materials for our new cutting-edge
product\n",
      "state": "needsReview"
    }
  ],
  "totalCount": 2
}
```

If no corresponding reviews are found, Swarm responds with an empty reviews list:

```
HTTP/1.1 200 OK

{
  "lastSeen": null,
  "reviews": [],
  "totalCount": 0
}
```

Finding inactive reviews (by checking the last updated date)

```
curl -u "username:password" "https://my-swarm-host/api/v9
/reviews?max=2&fields=id,changes,description,author,state&notUpdatedSince=
2017-01-01"
```

Swarm responds with a list of reviews that have not been updated since the notUpdatedSince date:

```
HTTP/1.1 200 OK

{
  "lastSeen": 100,
  "reviews": [
    {
      "id": 110,
      "author": "bruno",
      "changes": [5],
      "description": "Updating Java files\n",
      "state": "needsReview"
    },
    {
      "id": 100,
```

```
      "author": "bruno",
      "changes": [6,7],
      "description": "Marketing materials for our new cutting-edge
product\n",
      "state": "needsReview"
    }
  ],
  "totalCount": 2
}
```

Finding reviews I have voted up

```
curl -u "username:password" "https://my-swarm-
host/api/
v9/reviews?max=2&fields=id,changes,description,author,state&hasVoted=up"
```

Swarm responds with a list of reviews that include these changes:

```
HTTP/1.1 200 OK

{
  "lastSeen": 100,
  "reviews": [
    {
      "id": 110,
      "author": "bruno",
      "changes": [5],
      "description": "Updating Java files\n",
      "state": "needsReview"
    },
    {
      "id": 100,
      "author": "bruno",
      "changes": [6,7],
      "description": "Marketing materials for our new cutting-edge
product\n",
      "state": "needsReview"
    }
```

```
  ],
  "totalCount": 2
}
```

If no corresponding reviews are found, Swarm responds with an empty reviews list:

```
{
  "lastSeen": null,
  "reviews": [],
  "totalCount": 0
}
```

Finding reviews I have commented on (current authenticated user)

```
curl -u "username:password" "https://my-swarm-
host/api/
v9
/reviews?max=2&fields=id,changes,description,author,state&myComments=true"
```

Swarm responds with a list of reviews that include these changes:

```
HTTP/1.1 200 OK

{
  "lastSeen": 100,
  "reviews": [
    {
      "id": 110,
      "author": "bruno",
      "changes": [5],
      "description": "Updating Java files\n",
      "state": "needsReview"
    },
    {
      "id": 100,
      "author": "bruno",
      "changes": [6,7],
      "description": "Marketing materials for our new cutting-edge
product\n",
      "state": "needsReview"
```

```
    }
  ],
  "totalCount": 2
}
```

If no corresponding reviews are found, Swarm responds with an empty reviews list:

```
HTTP/1.1 200 OK

{
  "lastSeen": null,
  "reviews": [],
  "totalCount": 0
}
```

Get Review Information

Summary

Get Review Information

GET /api/v9/reviews/{id}

Description

Retrieve information about a review.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show. Omitting this parameter or passing an empty value shows all fields.	string	query	No

Example usage

Fetching a review

To fetch a review:

```
curl -u "username:password" "https://my-swarm-host/api/v9/reviews/123"
```

Swarm responds with a review entity:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 123,
    "author": "bruno",
    "changes": [122,124],
    "commits": [124],
    "commitStatus": [],
    "created": 1399325913,
    "deployDetails": [],
    "deployStatus": null,
    "description": "Adding .jar that should have been included in r110\n",
    "groups": [],
    "participants": {
      "alex_qc": [],
      "bruno": {
        "vote": 1,
        "required": true
      },
      "vera": []
    },
```

```
    "reviewerGroups": {
      "group1" : [],
      "group2" : {
        "required" : true
      },
      "group3" : {
        "required" : true,
        "quorum": "1"
      }
    },
    "pending": false,
    "projects": {
      "swarm": ["main"]
    },
    "state": "archived",
    "stateLabel": "Archived",
    "testDetails": {
      "url": "http://jenkins.example.com/job/project_ci/123/"
    },
    "testStatus": null,
    "type": "default",
    "updated": 1399325913,
    "versions": [
      {
        "difference": 1,
        "stream": "//jam/main",
        "streamSpecDifference": 0,
        "change": 124,
        "user": "bruno",
        "time": 1568115902,
        "pending": true,
        "addChangeMode": "replace",
        "testRuns": [
          8
        ]
```

```
      },
      {
        "difference": 1,
        "stream": "//jam/main",
        "streamSpecDifference": 0,
        "change": 12157,
        "user": "bruno",
        "time": 1568115918,
        "pending": true,
        "addChangeMode": "replace",
        "archiveChange": 122,
        "testRuns": [
          1,
          2,
          3,
          4,
          5,
          6,
          7
        ]
      }
    ]
  }
}
```

Fetching a review that does not exist

If you try to fetch a review that does not exist:

```
curl -u "username:password" "https://my-swarm-host/api/v9/reviews/999999"
```

Swarm responds with a 404 resonse:

```
HTTP/1.1 404 Not Found

{
  "error": "Not Found"
}
```

Get transitions for a review

Summary

Get transitions for a review

GET /api/v9/reviews/{id}/transitions

Description

Get the allowed transitions for a review.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`upVoters`	A list of users whose vote up will be assumed when determining the transitions. For example if a user has not yet voted but would be the last required vote and asked for possible transitions we would want to include 'approve'	string	query	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "isValid": "true",
  "transitions": {
      "needsRevision": "Needs Revision",
      "approved": "Approve",
```

```
      "approved:commit": "Approve and Commit",
      "rejected": "Reject",
      "archived": "Archive"
  }
}
```

Example usage

Getting transitions that a review with id 1 can transition to:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/reviews/1/transitions?upVoters=bruno"
```

The transitions that are returned depend on the current review state, Swarm setup and the configuration of associated projects and branches

```
HTTP/1.1 200 OK

{
  "isValid": "true"
  "transitions": {
      "needsRevision": "Needs Revision",
      "approved": "Approve",
      "approved:commit": "Approve and Commit",
      "rejected": "Reject",
      "archived": "Archive"
  }
}
```

Create a Review

Summary

Create a Review

POST /api/v9/reviews/

Description

Pass in a changelist ID to create a review. Optionally, you can also provide a description and a list of reviewers.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`change`	Change ID to create a review from	integer	form	Yes
`description`	Description for the new review (defaults to change description)	string	form	No
`reviewers`	A list of reviewers for the new review	array (of strings)	form	No
`requiredReviewers`	A list of required reviewers for the new review (v1.1+)	array (of strings)	form	No
`reviewerGroups`	A list of required reviewers for the new review (v7+)	array	form	No

Example response

Successful Response contains Review Entity:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 12205,
    "author": "bruno",
    "changes": [10667],
    "commits": [10667],
```

```
    "commitStatus": [],
    "created": 1399325913,
    "deployDetails": [],
    "deployStatus": null,
    "description": "Adding .jar that should have been included in
r10145\n",
    "participants": {
      "bruno": []
    },
    "reviewerGroups": {
      "group1" : [],
      "group2" : {
        "required" : true
      },
      "group3" : {
        "required" : true,
        "quorum": "1"
      }
    },
    "pending": false,
    "projects": [],
    "state": "archived",
    "stateLabel": "Archived",
    "testDetails": [],
    "testStatus": null,
    "type": "default",
    "updated": 1399325913
  }
}
```


Example usage

Starting a review

To start a review for a committed change or a non-empty shelved changelist specifying reviewer groups:

```
curl -u "username:password" \
     -X POST \
     -d "change=122" \
     -d "reviewerGroups[0][name]=group1" \
     -d "reviewerGroups[1][name]=group2" \
     -d "reviewerGroups[1][required]=true" \
     -d "reviewerGroups[2][name]=group3" \
     -d "reviewerGroups[2][required]=true" \
     -d "reviewerGroups[2][quorum]=1" \
     "https://my-swarm-host/api/v9/reviews/"
```

Swarm responds with the new review entity:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 123,
    "author": "bruno",
    "changes": [122],
    "commits": [],
    "commitStatus": [],
    "created": 1399325913,
    "deployDetails": [],
    "deployStatus": null,
    "description": "Adding .jar that should have been included in r110\n",
    "groups": [],
    "participants": {
      "bruno": []
    },
    "reviewerGroups": {
      "group1" : [],
      "group2" : {
        "required" : true
      },
      "group3" : {
        "required" : true,
```

```
        "quorum": "1"
      }
    },
    "pending": true,
    "projects": [],
    "state": "needsReview",
    "stateLabel": "Needs Review",
    "testDetails": [],
    "testStatus": null,
    "type": "default",
    "updated": 1399325913,
    "versions": []
  }
}
```


Archiving the inactive reviews

Summary

Archiving the inactive reviews (v6+)

POST /api/v9/reviews/archive/

Description

Archiving reviews not updated since the date (v6+).

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`notUpdatedSince`	Updated since date. Requires the date to be in the format YYYY-mm-dd, for example 2017-01-01	string	form	Yes
`description`	A description that is posted as a comment for archiving.	string	form	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "archivedReviews": [
    {
      "id": 836,
      "author": "swarm",
      "changes": [789],
      "commits": [],
      "commitStatus": [],
      "created": 1461164339,
      "deployDetails": [],
      "deployStatus": null,
      "description": "Review description\n",
      "groups": [],
      "participants": {
        "swarm": []
      },
      "pending": false,
      "projects": [],
      "state": "archived",
      "stateLabel": "Archived",
```

```
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1478191607
    }
  ],
  "failedReviews": []
}
```

Example usage

Archiving reviews inactive since 2016/06/30

To archive reviews not updated since 2016/06/30 inclusive:

```
curl -u "username:password" \
     -X POST \
     -d "notUpdatedSince=2016-06-30" \
     "https://my-swarm-host/api/v9/reviews/archive/"
```

Swarm responds with the list of archived reviews and failed reviews if there are any:

```
HTTP/1.1 200 OK

{
  "archivedReviews":[
    {
      "id": 911,
      "author": "swarm",
      "changes": [601],
      "commits": [],
      "commitStatus": [],
      "created": 1461164344,
      "deployDetails": [],
      "deployStatus": null,
      "description": "Touch up references on html pages.\n",
      "groups": [],
      "participants": {
        "swarm":[]
```

```
      },
      "pending": false,
      "projects": [],
      "state": "archived",
      "stateLabel": "Archived",
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1478191605
    },
    {
      "id": 908,
      "author": "earl",
      "changes": [605],
      "commits": [],
      "commitStatus": [],
      "created": 1461947794,
      "deployDetails": [],
      "deployStatus": null,
      "description": "Remove (attempted) installation of now deleted man
pages.\n",
      "groups": [],
      "participants": {
        "swarm": []
      },
      "pending": false,
      "projects": [],
      "state": "archived",
      "stateLabel": "Archived",
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1478191605
    }
  ],
```

```
  "failedReviews":[
    {
    }
  ]
}
```

If no reviews are archived, Swarm responds with an empty reviews list:

```
HTTP/1.1 200 OK

{
  "archivedReviews": [],
  "failedReviews": []
}
```

Add Change to Review

Summary

Add Change to Review

POST /api/v9/reviews/{id}/changes/

Description

Links the given change to the review and schedules an update.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required	Default Value
`id`	Review ID	integer	path	Yes	
`change`	Change ID	integer	form	Yes	

Parameter	Description	Type	Parameter Type	Required	Default Value
`mode`	The mode of operation, currently 'replace' or 'append'	string	form	No	replace

Example response

Successful Response contains Review Entity:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 12206,
    "author": "bruno",
    "changes": [10667, 12000],
    "commits": [10667, 12000],
    "commitStatus": [],
    "created": 1399325913,
    "deployDetails": [],
    "deployStatus": null,
    "description": "Adding .jar that should have been included in
r10145\n",
    "participants": {
      "bruno": []
    },
    "pending": false,
    "projects": [],
    "state": "archived",
    "stateLabel": "Archived",
    "testDetails": [],
    "testStatus": null,
    "type": "default",
    "updated": 1399325913
  }
}
```

Example usage

Adding a change to a review

You may want to update a review from a shelved or committed change that is different from the initiating change. This is done by adding a change to the review.

To replace a changelist:

```
curl -u "username:password" \
     -X POST \
     -d "change=124" \
     "https://my-swarm-host/api/v9/reviews/123/changes/"
```

To append a changelist:

```
curl -u "username:password" -X POST -d "change=124" -d "mode=append"
https://my-swarm-host/api/v9/reviews/123/changes/"
```

Swarm responds with the updated review entity:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 123,
    "author": "bruno",
    "changes": [122, 124],
    "commits": [],
    "commitStatus": [],
    "created": 1399325913,
    "deployDetails": [],
    "deployStatus": null,
    "description": "Adding .jar that should have been included in r110\n",
    "groups": [],
    "participants": {
      "bruno": []
    },
    "pending": true,
    "projects": [],
    "state": "needsReview",
    "stateLabel": "Needs Review",
```

```
    "testDetails": [],
    "testStatus": null,
    "type": "default",
    "updated": 1399325913,
    "versions": [
      {
        "difference": 1,
        "stream": null,
        "change": 124,
        "user": "bruno",
        "time": 1399330003,
        "pending": true,
        "archiveChange": 124
      }
    ]
  }
}
```

Clean up a review

Summary

Clean up a review (v6+)

POST /api/v9/reviews/{id}/cleanup

Description

Clean up a review for the given id.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`reopen`	Expected to be a boolean (defaulting to false). If true then an attempt will be made to reopen files into a default changelist	boolean	form	No

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "complete": [
    {
      "1": ["2"]
    }
  ],
  "incomplete": []
}
```

Example usage

Cleaning up a review with id 1

Cleanup review number 1, reopening any files into the default changelist.

```
curl -u "username:password" \
     -X POST \
     -d "reopen=true" \
     "https://my-swarm-host/api/v9/reviews/1/cleanup"
```

Swarm responds with the review and the changelists cleaned. Depending on the completion they will be either detailed in 'complete' or 'incomplete'. Incomplete changelists will have messages indicating why it was not possible to complete:

```
HTTP/1.1 200 OK
```

```
{
  "complete": [
    {
      "1": ["2"]
    }
  ],
  "incomplete": []
}
```

Obliterate a review

Summary

Obliterate a review

POST /api/v9/reviews/{id}/obliterate

Description

Obliterate a review for the given id.

> **Tip**
>
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "isValid": true,
  "message": "The review with id [1] has been obliterated.",
  "code": 200
}
```

Example usage

Obliterating a review with id 1

Obliterating review number 1.

```
curl -u "username:password" \
     -X POST \
     "https://my-swarm-host/api/v9/reviews/1/obliterate"
```

Swarm responds with a message informing you that is has successfully Obliterated the review:

```
HTTP/1.1 200 OK

{
  "isValid": true,
  "message": "The review with id [1] has been obliterated.",
  "code": 200
}
```

Set vote for the authenticated user to up, down, or cleared

Summary

Set the vote for the authenticated user to be up, down or cleared

POST /api/v9/reviews/{id}/vote/

Description

Set the vote for the authenticated user to be up, down or clear.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`version`	Expected to be a valid review revision to vote on if supplied, ignored if the revision does not exist and the vote will apply to the latest revision	string	form	No
`vote`	Expected to be a valid vote. Valid votes are 'up', 'down' and 'clear	string	form	Yes

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "isValid": "true",
  "messages": ["User username set vote to up on review 1"]
}
```

Example usage

Voting up for review with id 1

```
curl -u "username:password" \
     -X POST \
     -d "vote[value]=up" -d "vote[version]=1" \
     "https://my-swarm-host/api/v9/reviews/1/vote/"
```

Swarm responds with

```
HTTP/1.1 200 OK

{
  "isValid": "true",
  "messages": ["User username set vote to up on review 1"]
}
```

Transition the Review State

Summary

Transition the Review State (v2+)

PATCH /api/v9/reviews/{id}/state/

Description

Transition the review to a new state. When transitioning to approved, you can optionally commit the review. (v2+).

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes
`state`	Review State. Valid options: needsReview, needsRevision, approved, archived, rejected	string	form	Yes
`description`	An optional description that is posted as a comment for non-commit transitions. Commits that do not include a description default to using the Review description in the resulting change description.	string	form	No
`commit`	Set this flag to true and provide a state of `approved` in order to trigger the **Approve and Commit** action in Swarm.	boolean	form	No

Parameter	Description	Type	Parameter Type	Required
`wait`	Instruct Swarm to wait for a commit to finish before returning.	boolean	form	No
`jobs[]`	When performing an 'Approve and Commit', one or more jobs can be attached to the review as part of the commit process.	stringArray	form	No
`fixStatus`	Provide a fix status for the attached job(s) when performing an 'Approve and Commit'. Possible status values vary by job specification, but often include: open, suspended, closed, review, fixed.	string	form	No

Example responses

Successful Response contains Review Entity:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 12207,
    "author": "bruno",
    "changes": [10667, 12000],
    "commits": [],
    "commitStatus": [],
    "created": 1399325913,
    "deployDetails": [],
    "deployStatus": null,
    "description": "Adding .jar that should have been included in
r10145\n",
    "participants": {
      "bruno": []
    },
    "pending": false,
```

```
    "projects": [],
    "state": "needsRevision",
    "stateLabel": "Needs Revision",
    "testDetails": [],
    "testStatus": null,
    "type": "default",
    "updated": 1399325913
  },
  "transitions": {
    "needsReview": "Needs Review",
    "approved": "Approve",
    "rejected": "Reject",
    "archived": "Archive"
  }
}
```

Successful Commit contains Review and Commit Entities:

```
HTTP/1.1 200 OK

{
  "review": {
    "id": 12208,
    "author": "bruno",
    "changes": [10667, 12000, 12006],
    "commits": [12006],
    "commitStatus": {
      "start": 1399326910,
      "change": 12006,
      "status": "Committed",
      "committer": "bruno",
      "end": 1399326911
    },
    "created": 1399325900,
    "deployDetails": [],
    "deployStatus": null,
```

```
    "description": "Adding .jar that should have been included in
r10145\n",
    "participants": {
      "bruno": []
    },
    "pending": false,
    "projects": [],
    "state": "needsRevision",
    "stateLabel": "Needs Revision",
    "testDetails": [],
    "testStatus": null,
    "type": "default",
    "updated": 1399325905
  },
  "transitions": {
    "needsReview": "Needs Review",
    "needsRevision": "Needs Revision",
    "rejected": "Reject",
    "archived": "Archive"
  },
  "commit": 12006
}
```

Example usage

Committing a review

To commit a review:

```
curl -u "username:password" \
     -X PATCH \
     -d "state=approved" -d "commit=1" \
     "https://my-swarm-host/api/v9/reviews/123/state/"
```

Swarm responds with the updated review entity, as well as a list of possible transitions for the review:

```
HTTP/1.1 200 OK

{
```

```
"review": {
  "id": 123,
  "author": "bruno",
  "changes": [122, 124],
  "commits": [124],
  "commitStatus": {
      "start": 1399326910,
      "change": 124,
      "status": "Committed",
      "committer": "bruno",
      "end": 1399326911
    },
  "created": 1399325913,
  "deployDetails": [],
  "deployStatus": null,
  "description": "Adding .jar that should have been included in r110\n",
  "groups": [],
  "participants": {
    "bruno": []
  },
  "pending": false,
  "projects": [],
  "state": "approved",
  "stateLabel": "Approved",
  "testDetails": [],
  "testStatus": null,
  "type": "default",
  "updated": 1399325913,
  "versions": []
},
  "transitions": {
    "needsReview": "Needs Review",
    "approved": "Approve",
    "rejected": "Reject",
    "archived": "Archive"
```

```
    }
}
```

Update Review Description

Summary

Update Review Description

PATCH /api/v9/reviews/{review_id}

Description

Update the description field of a review.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`review_id`	Review ID	integer	path	Yes
`author`	The new author for the specified review. (At least one of Author or Description are required.)	string	form	No
`description`	The new description for the specified review. (At least one of Description or Author are required.)	string	form	No
`_method`	Method Override. If your client cannot submit HTTP PATCH, use an HTTP POST with the parameter ?_method=PATCH to override.	string	query	No

Example responses

Successful Response:

```
HTTP/1.1 200 OK

{
  "review": {
      "id": 12306,
      "author": "swarm",
      "changes": [12205],
      "comments": 0,
      "commits": [],
      "commitStatus": [],
      "created": 1402507043,
      "deployDetails": [],
      "deployStatus": null,
      "description": "Updated Review Description\n",
      "participants": {
        "swarm": []
      },
      "pending": true,
      "projects": [],
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testDetails": [],
      "testStatus": null,
      "type": "default",
      "updated": 1402518492
  },
  "transitions": {
      "needsRevision": "Needs Revision",
      "approved": "Approve",
      "rejected": "Reject",
      "archived": "Archive"
  },
```

```
  "canEditAuthor": true
}
```

> **Note**
> Swarm returns `null` for `totalCount` if no search filters were provided. `lastSeen` can often be used as an offset for pagination, by using the value in the `after` parameter of subsequent requests.

When no results are found, the `reviews` array is empty:

```
HTTP/1.1 200 OK

{
  "lastSeen": null,
  "reviews": [],
  "totalCount": 0
}
```

Servers : Swarm Servers API

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get a list of servers

Summary

Gets a list of servers

GET /api/v9/servers/

Description

Gets a list of servers

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
  "servers": {
    "Master": {
     "port": "ssl:10.33.44.55:1666"
    },
    "Artifacts": {
     "port": "10.55.66.77:1666"
    }
  }
}
```

Example usage

Get a list of servers

Get a list of servers that Swarm is aware of

```
curl -u "username:password" "https://myswarm.url/api/v9/servers/"
```

If multiple p4d are configured it will give an output like

```
HTTP/1.1 200 OK

{
  "servers": {
    "Master": {
       "port": "ssl:10.33.44.55:1666"
    },
    "Artifacts": {
       "port": "10.55.66.77:1666"
    }
  }
}
```

If a single p4d is configured it will give an output like

```
HTTP/1.1 200 OK

{
  "servers": {
    "p4": {
       "port": "ssl:10.33.44.55:1666"
    }
  }
}
```

Users : Swarm Users

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get List of Users

Summary

Get List of Users

GET /api/v9/users/

Description

Returns a list of users in Swarm.

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each user. Omitting this parameter or passing an empty value shows all fields. Be aware, the fields are case sensitive for users. You can use one of the following: `User`, `Type`, `Email`, `Updated`, `Access`, `FullName`, `JobView`, `Password`, `AuthMethod`, `Reviews`.	string	query	No
`users`	An optional comma-separated list (or array) of users to display. Omitting this parameter or passing an empty value shows all users.	string	query	No
`group`	An optional to get users from a group. Cannot be used with users parameter	string	query	No
`ignoreExcludeList`	Determines if the list of users has the user_exclude_list filter applied or not. Add the parameter to ignore the `user_exclude_list` filter.	boolean	query	No

Example response

Successful Response:

```
HTTP/1.1 200 OK
```

```
{
    "User": "bruno",
    "Type": "standard",
    "Email": "bruno@perforce.com",
    "Updated": "2005/10/11 21:05:15",
    "Access": "2018/04/12 14:55:10",
    "FullName": "Bruno First",
    "JobView": null,
    "Password": null,
    "AuthMethod": "perforce",
    "Reviews": []
}
```

Example usage

Listing users

To list all users:

```
curl -u "username:password" "https://my-swarm-
host/api/v9/users?fields=User,FullName&users=super,bruno"
```

Pagination is not currently supported by this endpoint. Swarm responds with a list of all users:

```
HTTP/1.1 200 OK

{
    {
      "User": "bruno",
      "FullName": "Bruno First"
    },
    {
      "User": "super",
      "FullName": "Super Second"
    }
}
```

Unfollow all Users and Projects

Summary

Unfollow all Users and Projects

POST /api/v9/users/{user}/unfollowall

Description

Admin and super users are permitted to execute unfollow all against any target user. Other users are only permitted to execute the call if they themselves are the target user

Example response

Successful Response:

```
HTTP/1.1 200 OK

{
    "isValid": true,
    "messages": "User {user} is no longer following any Projects or
Users."
}
```

Workflows : Controller for querying, creating and updating workflows

> **Important**
> From Swarm 2019.3, APIs older than v9 are being deprecated. Support for them will be removed in a future release.

Get workflows

Summary

Gets workflows

GET /api/v9/workflows/

Description

Get all workflows

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each workflow. Omitting this parameter or passing an empty value shows all fields.	string	query	No
`noCache`	If provided and has a value of 'true' a query will always be performed and the cache of workflows is ignored. Otherwise the cache will be used if it exists.	boolean	query	No

Usage example

Get a list of workflows

```
curl -u "username:password" "https://my-swarm-host/api/v9/workflows"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflows": [
        {
            "on_submit":{
                "without_review":{
                    "rule": "no_checking"
                },
                "with_review":{
                    "rule": "no_checking"
                }
            },
            "name": "myWorkflow",
            "description": "A description",
```

```
            "shared": "true",
            "owners": [
                "user1",
                "user2"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_revision"
                }
            },
            "auto_approve":{
                "rule": "never"
            },
            "counted_votes":{
                "rule": "anyone"
            },
            "group_exclusions":{
                "rule": [ ]
            },
            "user_exclusions":{
                "rule": [ ]
            },
            "id": "1"
        },
        {
            "on_submit":{
                "without_review":{
                    "rule": "no_checking"
                },
                "with_review":{
                    "rule": "no_checking"
                }
            },
            "name": "myWorkflow 2",
            "description": "Another description",
```

```
            "shared": "true",
            "owners": [
                "user3",
                "user4"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_revision"
                }
            },
            "auto_approve":{
                "rule": "votes"
            },
            "counted_votes": {
                "rule": "members"
            },
            "group_exclusions":{
                "rule": [ ]
            },
            "user_exclusions":{
                "rule": [ ]
            },
            "id": "2"
        },
    ]
}
```


Get a workflow by id

Summary

Gets a workflow by id

GET /api/v9/workflows/{id}

Description

Gets a workflow by id

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each workflow. Omitting this parameter or passing an empty value shows all fields.	string	query	No

Example usage

Get a workflows by id

```
curl -u "username:password" "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflows": [
        "on_submit":{
            "without_review":{
                "rule": "no_checking"
            },
            "with_review":{
                "rule": "no_checking"
            }
        },
        "name": "myWorkflow",
        "description": "A description",
        "shared": "true",
        "owners": [
            "user1",
            "user2"
        ],
        "end_rules":{
            "update":{
                "rule":"no_revision"
```

```
            }
        },
        "auto_approve":{
            "rule": "never"
        },
        "counted_votes":{
            "rule": "anyone"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id": "1"
    }
}
```

Create a workflow

Summary

Create a workflow

POST /api/v9/workflows/

Description

Create a new workflow.

Parameters

Parameter	Description	Type	Parameter Type	Required
`name`	The workflow name. Will be compared against other workflows and rejected if not unique	string	form	Yes
`description`	Description for the new workflow	string	form	No

Parameter	Description	Type	Parameter Type	Required
`shared`	Whether this workflow is shared for other users that do not own it. Defaults to not shared	boolean	form	No
`owners`	A list owners for the workflow. Can be users or group names (prefixed with swarm-group-). Users and group names must exist or the workflow will be rejected	array (of strings)	form	No
`on_submit`	Data for rules when changes are submitted. Valid values for with_ review are no_checking, approved, strict. Valid values for without review are no_checking, auto_ create, reject	array	form	No
`end_rules`	Data for rules when changes are submitted. Valid values are no_ checking, no_revision.	array	form	No
`auto_ approve`	Data for rules when changes are submitted. Valid values are votes, never.	array	form	No
`counted_ votes`	Data for rules when counting votes up. Valid values are anyone, members.	array	form	No

Example usage

Create a workflow

```
curl -u "username:password" \
     -X POST \
     -d "on_submit[with_review][rule]=no_checking" \
     -d "on_submit[without_review][rule]=no_checking" \
     -d "name=myWorkflow" \
     -d "description=A description" \
     -d "shared=true" \
     -d "owners[]=Francois_Piccard" \
     -d "owners[]=Anna_Schmidt" \
```

```
     -d "end_rules[update][rule]=no_checking" \
     -d "auto_approve:[rule]never" \
     -d "counted_votes:[rule]members" \
     "https://my-swarm-host/api/v9/workflows"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"myWorkflow",
        "description":"A description",
        "shared":true,
        "owners":[
            "Anna_Schmidt",
            "Francois_Piccard"
        ],
        "end_rules":{
            "update":{
                "rule":"no_checking"
            }
        },
        "auto_approve":{
            "rule":"never"
        },
        "counted_votes":{
            "rule":"members"
        },
```

```
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```


Patch a workflow

Summary

Patch a workflow

PATCH /api/v9/workflows/{id}

Description

Patch a workflow.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	The id of the workflow being patched	string	form	Yes
`name`	The workflow name. Will be compared against other workflows and rejected if not unique	string	form	No
`description`	Description for the new workflow	string	form	No
`shared`	Whether this workflow is shared for other users that do not own it. Defaults to not shared	boolean	form	No

Parameter	Description	Type	Parameter Type	Required
`owners`	A list owners for the workflow. Can be users or group names (prefixed with swarm-group-). Users and group names must exist or the workflow will be rejected	array (of strings)	form	No
`on_submit`	Data for rules when changes are submitted. Valid values for with_ review are no_checking, approved, strict. Valid values for without review are no_checking, auto_ create, reject	array	form	No
`end_rules`	Data for rules when changes are submitted. Valid values are no_ checking, no_revision.	array	form	No
`auto_ approve`	Data for rules when changes are submitted. Valid values are votes, never.	array	form	No
`counted_ votes`	Data for rules when counting votes up. Valid values are anyone, members.	array	form	No

Example usage

Patch workflow name and description

```
curl -u "username:password" \
     -X PATCH \
     -d "name=myWorkflow" \
     -d "description=A description" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
```

```
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"myWorkflow",
        "description":"A description",
        "shared":false,
        "owners":[
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_revision"
            }
        },
        "auto_approve":{
            "rule":"votes"
        },
        "counted_votes":{
            "rule":"members"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```

Patch on_submit with_review

```
curl -u "username:password" \
     -X PATCH \
     -d "on_submit[with_review][rule]=no_checking" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":false,
        "owners":[
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_revision"
            }
        },
        "auto_approve":{
            "rule":"votes"
        },
        "counted_votes":{
            "rule":"members"
        },
```

```
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```

Patch on_submit without_review

```
curl -u "username:password" \
     -X PATCH \
     -d "on_submit[without_review][rule]=no_checking" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":false,
        "owners":[
            "bruno"
        ],
        "end_rules":{
```

```
            "update":{
                "rule":"no_revision"
            }
        },
        "auto_approve":{
            "rule":"votes"
        },
        "counted_votes":{
            "rule":"members"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```

Patch Auto_approve

```
curl -u "username:password" \
     -X PATCH \
     -d "auto_approve[rule]=never" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
```

```
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":false,
        "owners":[
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_revision"
            }
        },
        "auto_approve":{
            "rule":"never"
        },
        "counted_votes":{
            "rule":"members"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```

Patch counted_votes

```
curl -u "username:password" \
     -X PATCH \
     -d "counted_votes[rule]=anyone" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":false,
        "owners":[
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_revision"
            }
        },
        "auto_approve":{
            "rule":"never"
        },
        "counted_votes":{
            "rule":"anyone"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
```

```
        "id":1
    }
}
```

Patch end_rules

```
curl -u "username:password" \
     -X PATCH \
     -d "end_rules[update][rule]=no_checking" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":false,
        "owners":[
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_checking"
            }
        },
        "auto_approve":{
            "rule":"never"
```

```
        },
        "counted_votes":{
            "rule":"members"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```

Patch shared

```
curl -u "username:password" \
     -X PATCH \
     -d "shared=true" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":true,
```

```
        "owners":[
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_checking"
            }
        },
        "auto_approve":{
            "rule":"never"
        },
        "counted_votes":{
            "rule":"anyone"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```

Patch Owners

```
curl -u "username:password" \
     -X PATCH \
     -d "owners[]=Francois_Piccard" \
     -d "owners[]=Anna_Schmidt" \
     -d "owners[]=bruno" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
```

```
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
            }
        },
        "name":"test",
        "description":"test line 2",
        "shared":true,
        "owners":[
            "Anna_Schmidt",
            "Francois_Piccard",
            "bruno"
        ],
        "end_rules":{
            "update":{
                "rule":"no_checking"
            }
        },
        "auto_approve":{
            "rule":"never"
        },
        "counted_votes":{
            "rule":"anyone"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
```

```
    }
}
```

Update a workflow

Summary

Update a workflow

PUT /api/v9/workflows/{id}

Description

Update a workflow. All values should be provided in the request. If not provided any missing values are reverted to default.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	The id of the workflow being patched	string	form	Yes
`name`	The workflow name. Will be compared against other workflows and rejected if not unique	string	form	Yes
`owners`	A list owners for the workflow. Can be users or group names (prefixed with swarm-group-). Users and group names must exist or the workflow will be rejected	array (of strings)	form	No
`description`	Description for the new workflow	string	form	No
`shared`	Whether this workflow is shared for other users that do not own it. Defaults to not shared	boolean	form	No
`on_submit`	Data for rules when changes are submitted. Valid values for with_ review are no_checking, approved, strict. Valid values for without review are no_checking, auto_ create, reject	array	form	No

Parameter	Description	Type	Parameter Type	Required
`end_rules`	Data for rules when changes are submitted. Valid values are no_checking, no_revision.	array	form	No
`auto_approve`	Data for rules when changes are submitted. Valid values are votes, never.	array	form	No
`counted_votes`	Data for rules when counting votes up. Valid values are anyone, members.	array	form	No

Example usage

Update a workflow

```
curl -u "username:password" \
     -X PUT \
     -d "on_submit[with_review][rule]=no_checking" \
     -d "on_submit[without_review][rule]=no_checking" \
     -d "name=myWorkflow" \
     -d "description=A description" \
     -d "shared=true" \
     -d "owners[]=Francois_Piccard" \
     -d "owners[]=Anna_Schmidt" \
     "https://my-swarm-host/api/v9/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
    "workflow":{
        "on_submit":{
            "with_review":{
                "rule":"no_checking"
            },
            "without_review":{
                "rule":"no_checking"
```

```
            }
        },
        "name":"myWorkflow",
        "description":"A description",
        "shared":true,
        "owners":[
            "Anna_Schmidt",
            "Francois_Piccard"
        ],
        "end_rules":{
            "update":{
                "rule":"no_checking"
            }
        },
        "auto_approve":{
            "rule":"never"
        },
        "counted_votes":{
            "rule":"anyone"
        },
        "group_exclusions":[ ],
            "rule": [ ]
        },
        "user_exclusions":[ ],
            "rule": [ ]
        },
        "id":1
    }
}
```


Delete a workflow

Summary

Delete a workflow

DELETE /api/v9/workflows/{id}

Description

Delete a workflow for the provided id. This call must be authenticated and the user must have permission to edit the workflow. If the workflow is in use it cannot be deleted and an error message will be returned.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	The id of the workflow being deleted	string	form	Yes

Example usage

Delete a workflow not in use

```
curl -u "username:password" -X DELETE "https://my-swarm-
host/api/v9/workflows/1"
```

JSON Response

```
HTTP/1.1 200 OK

{
    "isValid": true,
    "messages": [
        "Workflow [1] was deleted"
    ]
}
```

Delete a workflow that is in use on public projects

```
curl -u "username:password" -X DELETE "https://my-swarm-
host/api/v9/workflows/1"
```

JSON Response

```
HTTP/1.1 200 OK

{
    "isValid": false,
    "messages": [
        "Cannot delete workflow [1], it is in use on project [project1,
project2]"
    ]
}
```

Delete a workflow that is in use on a public and some private projects

```
curl -u "username:password" -X DELETE "https://my-swarm-
host/api/v9/workflows/1"
```

JSON Response

```
HTTP/1.1 200 OK

{
    "isValid": false,
    "messages": [
        "Cannot delete workflow [1], it is in use on project [project1]
and others"
    ]
}
```

Delete a workflow that is in use on only private projects

```
curl -u "username:password" -X DELETE "https://my-swarm-
host/api/v9/workflows/1"
```

JSON Response

```
HTTP/1.1 200 OK

{
    "isValid": false,
    "messages": [
```

```
        "Cannot delete workflow [1], it is in use"
    ]
}
```

API (v9) examples

This section contains some API (v9) examples.

Extended API example

This section contains an extended API example, involving multiple API calls to answer a more complicated kind of question than any single API endpoint can provide: which reviews does a specific *userid* need to attend to?

The code

```
<?php
/**
 * vim:set ai si et ts=4 sw=4 syntax=php:
 *
 * reviews.php
 *
 * Queries the Swarm API and reports which reviews a specified user
 * needs to attend to.
 *
 * Required attention is determined by the following criteria:
 * - the user is a participant in a review
 * - and the user has not voted on the review
 * - and the user has not commented on the review
 * - or the user's comment on the review is a
 *   task that has been addressed and needs verification
 */

if (ini_set('track_errors', 1) === false) {
    echo "Warning: unable to track errors.\n";
}
```

```
# process command-line arguments
$options = getopt(
    'hs:r:v',
    array('help', 'swarm:', 'reviewer', 'verbose')
);

$swarm = '';
if (isset($options['s'])) {
    $swarm = $options['s'];
}
if (isset($options['swarm'])) {
    $swarm = $options['swarm'];
}
if (!$swarm) {
    usage('Swarm API URL not provided.');
}

$reviewer = '';
if (isset($options['r'])) {
    $reviewer = $options['r'];
}
if (isset($options['reviewer'])) {
    $reviewer = $options['reviewer'];
}
if (!$reviewer) {
    usage('Swarm reviewer not provided.');
}

$verbose = false;
if (isset($options['v']) || isset($options['verbose'])) {
    $verbose = true;
}

if (isset($options['h']) || isset($options['help'])) {
    usage();
```

```
}

function usage($message = null)
{
    if ($message) {
        echo "$message\n\n";
    }

    $script = basename(__FILE__);
    echo <<<EOU
$script: -s <Swarm URL> -u <API userid> -p <API user's password> \
  -r <reviewer userid to report on> -h

-s|--swarm     Swarm's URL (e.g. https://user@password:myswarm.url/)
-r|--reviewer  The reviewer to report on.
-h|--help      This help text.
-v|--verbose   Verbose output.

This script queries the Swarm API and reports on reviews that the
specified user needs to attend to.

Note: If your Helix Core Server (p4d) has security level 3 set, you
cannot use a password to authenticate; you must acquire a host-unlocked
ticket from p4d, and use the ticket in place of a password when
communicating with the Swarm API connected to p4d.

EOU;
    exit;
}

function msg($message)
{
    global $verbose;

    if ($verbose) {
```

```
        echo $message;
    }
}

function call_api($url, $params)
{
    global $php_errormsg;

    $query    = http_build_query($params);
    $request  = $url . '?' . $query;
    $response = @file_get_contents($request);
    if ($php_errormsg) {
        echo "Unable to call api: $php_errormsg\n";
        exit;
    }

    $json = @json_decode($response, true);
    if ($php_errormsg) {
        echo "Unable to decode api response: $php_errormsg\n";
        exit;
    }

    return $json;
}

# remove trailing / from URL, if it exists
$swarm = rtrim(trim($swarm), '/');

# fetch the list of reviews
$reviews = call_api(
    "$swarm/api/v9/reviews",
    array(
        'hasReviewers' => 1, # only reviews with participants
        'participants' => array($reviewer), # only review for this
reviewer
```

```
        'max'          => 9, # get plenty of reviews, if available
        'fields'       => array('id', 'description', 'commits'), # get
these fields
    )
);

$report = array();
foreach ($reviews['reviews'] as $review) {
    if (is_null($review)) {
        continue;
    }

    $flag = false;
    msg('Review: ' . $review['id'] . ' ');

    # if the review is already committed, it likely does not need
attention
    if (array_key_exists('commits', $review)
        && count($review['commits'])
    ) {
        msg("is committed, skipping...\n");
        continue;
    }

    # if the review has a vote from the reviewer, they are already aware
    if (array_key_exists('participants', $review)
        && array_key_exists('vote', $review['participants'][$reviewer])
    ) {
        msg("has vote from reviewer, skipping...\n");
        continue;
    }

    # if there are no open comments on the review, the reviewer's
    # attention is required
    if (array_key_exists('comments', $review)
```

```
        && $review['comments'][0] == 0
    ) {
        msg("has no open comments, skipping...\n");
        continue;
    }

    # fetch the comments for this review
    $comments = call_api(
        "$swarm/api/v9/comments",
        array(
            'topic' => 'reviews/' . $review['id'], # comments for this
review
            'max'   => 9, # get plenty of comments, if available
        )
    );

    foreach ($comments['comments'] as $comment) {
        msg("\n  Comment: " . $comment['id'] . ' ');

        // skip over comments from other reviewers
        if (array_key_exists('user', $comment) && $reviewer != $comment
['user']) {
            msg("is by another user, carry on...\n");
            continue;
        }

        # skip archived comments
        if (array_key_exists('flags', $comment)
            && count($comment['flags']) > 0
            && $comment['flags'][0] == 'closed'
        ) {
            msg("is archived, carry on...\n");
            continue;
        }
```

```
        # skip marked tasks
        if (array_key_exists('taskState', $comment)
            && ($comment['taskState'] == 'comment'
                || $comment['taskState'] == 'verified'
                || $comment['taskState'] == 'open'
            )
        ) {
            msg("reviewer's comment needs attention, carry on...\n");
            continue;
        }

        // anything else means that the reviewer's comment needs attention
        // by the reviewer
        $flag = true;
        msg("needs attention!\n");
        break;
    }

    // evaluation is complete. Does this review need attention?
    if ($flag) {
        $report[] = $review;
    }
}

if (count($report)) {
    echo "User '$reviewer' needs to attend to these reviews:\n";
    foreach ($report as $review) {
        $description = trim($review['description']);
        if (strlen($description) > 60) {
            $description = substr($description, 0, 60) . ' ...';
        }
        echo $review['id'] . ": $description\n";
    }
} else {
    echo "User '$reviewer' has no reviews to attend to.\n";
```

```
}
```

Executing the example

The example is written in PHP. To use it copy and paste it into a file called `reviews.php`. Then, execute it like this:

```
$ php reviews.php -s https://myswarm.host:port/ -r bob
```

Replace ***http://myswarm.host/*** with the URL to your Swarm installation. Replace ***bob*** with the userid you'd like to report on.

To authenticate, insert `username:password@` before the hostname. If your Helix Core server security counter is set to `3` or higher, you need to acquire a ticket and use the ticket in place of the password (see Authentication for details). If your Swarm is installed on a custom port, or is installed in a sub-folder, include those elements in the URL as well. For example:

```
$ php reviews.php -s
https://me:F0FC33068BA244B1BBD8196CC9166F34@my.host:8080/swarm/ -
r bob
```

If you do not specify the URL correctly, you might see an error like:

```
Unable to call api: file_get_contents
(http://...@my.host:8080/swarm/api/v9
/reviews?hasReviewers=1&participants%5B0%5D=bob&max=9&fields%5B0%5
D=id&fields%5B1%5D=description&fields%5B2%5D=commits): failed to
open stream: HTTP request failed! HTTP/1.1 404 Not Found
```

If there are no errors, and the specified userid does have reviews to attend to, the output might look like:

```
1234: Added grapple-grommit support to woozlewobble class. @bob sh
...
```

`1234` is the id of a review that `bob` should attend to, followed by the first 60 characters of the review's description.

Pending review cleanup API example

This section contains an example script that cleans up pending changelists which are no longer needed. See the review cleanup options for how this can be done automatically when a review is committed.

For pending changelists which were present before this option was available, or for reviews which have been contributed to by multiple authors and so require super user access to tidy up, there is an API which allows the super user to bulk remove such changelists.

The script demonstrates how this API could be used. It isn't meant as a complete solution, just a guide to demonstrate what is possible.

The code

```
<?php
/**
 * Perforce Swarm
 *
 * @copyright   2017 Perforce Software. All rights reserved.
 * @license     Please see LICENSE.txt in top-level folder of this
distribution.
 * @version     <release>/<patch>
 */
/**
 * This example script can be used to clean up (delete) Perforce Server
pending changelists automatically when run
 * as a super user. It is able to query for reviews based on parameters to
establish which changelists are eligible
 * for clean up. In this way, it can be tailored to run against reviews of
a user's choice.

 * Requirements for this script:
 *   - MUST be a super user
 *   - MUST populate the parameters below
 *   - MUST be using Swarm 2017.1 or later
 *
 * Usage of script
 *   php superUserReviewCleanUp.php max=10 notUpdatedSince=2017-04-01
state=approved
 *   php superUserReviewCleanUp.php max=10 author=bruno state=approved
 *
 * Each of the parameters that you can use are documented at the following
URL:
 *
https://www.perforce.com/perforce/doc.current/manuals/swarm/Content/Swarm/
swarm-apidoc_endpoint_reviews.html
 *
 *
```

```
 * This returns a JSON object that contains four main objects.
 *
 * {
 *   "error": "",
 *   "help": "",
 *   "results": "[]",
 *   "search_criteria": {
 *     "fields": "id",
 *     "max": "10",
 *     "notUpdatedSince": "2017-04-01"
 *   }
 * }
 *
 * Referencing the example above:
 *
 * The error section will contain any errors that have been encountered
trying to execute the script.
 *
 * The help section will populated if you have run the command php
superUserReviewCleanUp.php help
 *
 * The search_criteria section indicates which parameters have been used
to fetch the reviews list.
 *
 * The results section returns a JSON object of each of the reviews it has
processed, which may
 * include actions that were incomplete.
 *
 * Below is an example of results being processed:
 * "814": {
 *    "complete": [],
 *    "incomplete": {
 *       "814": {
 *           "813": [
 *               "0": "Command failed: No shelved files in changelist to
```

```
delete.",
 *           ]
 *       }
 *    }
 * },
 * "818": {
 *    "complete": [],
 *    "incomplete": []
 * }
 * "820": {
 *    "complete": [],
 *    "incomplete": {
 *       "820": {
 *           "821": [
 *              "0": "Command failed: No shelved files in changelist to
delete.",
 *              "1": "Command failed: Usage: fix [ -d ] [ -s status ] -c
changelist# jobName ...\nMissing/wrong
 *                    number of arguments."
 *           ]
 *       }
 *    }
 * },
 *
 * Some reviews may have no actions or incomplete actions. Incomplete
actions indicate that additional work is required
 * and the review could not be entirely cleaned up. In the example above,
the first message indicates a changelist was
 * not found. This could be because the end user has already deleted it.
 *
 * An error with the fix command can indicate that the pending changelist
doesn't have any jobs linked or that the jobs
 * have already been removed.
 *
 * NOTES:
```

```
 * To make the output print nicely, you can use the python command like
this:
 *
 * php superUserReviewCleanUp.php max=10 notUpdatedSince=2018-04-01
state=approved | python -m json.tool
 *
 */


/* ************************************************** */
/* These values MUST be set before running the script.  */
/* ************************************************** */

// URL to access swarm. Must not include trailing slash.
$swarmURL = 'http://my.swarm.com';
// Username of super user
$username = '';
// Ticket for user, can be created by running "p4 -u $username login -pa"
$ticket = '';

/* ************************************************** */


// If the super user wants to reopen the files of the end user.
$reopen = true;
// Prebuild the the return message helper array.
$help = array( "help" => "", "error" => "", "results" => "", "search_
criteria" => "");
// @codingStandardsIgnoreEnd
/**
 * function that make the GET or POST requests
 *
 * @param $url         Url in which we want to make our request to
 * @param bool $fetch  Set to true for a GET request, otherwise will do a
POST.
```

```
 * @param array $args  These are the parameter that we pass the the GET or
POST request to filter the reviews
 * @return JSON        We return a JSON object back to be worked on.
 */
function request($url, $fetch = false, $args = array())
{
    // Fetch the settings to allow this function to access them.
    global $username, $ticket, $reopen, $help;
    $curl = '';

    // If GET request fetch should be true and args shouldn't be empty
    if ($fetch === true) {
        // If is args is empty just give the url, otherwise build a http
query with args elements
        $curl = empty($args) ? curl_init($url) : curl_init($url."?".http_
build_query($args));
    } else {
        // Assume fetch is false and build a POST request.
        $curl = curl_init($url."/".$args['id'].'/cleanup');
        curl_setopt($curl, CURLOPT_POSTFIELDS, http_build_query(array
('reopened'=>$reopen)));
        curl_setopt($curl, CURLOPT_POST, count($reopen));
    }
    curl_setopt($curl, CURLOPT_USERPWD, "$username:$ticket");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);

    $result = curl_exec($curl);

    // Catch the error code in case Ticket expired or url doesn't return.
    $statusCode = curl_getinfo($curl, CURLINFO_HTTP_CODE);
    if ($statusCode != 200) {
        $help['error'] = array("HTTP code" => $statusCode);
    }
```

```
    curl_close($curl);
    return json_decode($result, true);
}

/**
 * Fetch a list of Reviews based on parameters
 *
 * @param $elements  Each of the args passed from command line are treated
as elements
* @return JSON      We return a JSON object back to be worked on.
 */
function fetchReviews($elements)
{
    global $swarmURL, $help;
    $parameters = array();

    if ($elements !== null) {
        // Loop though each of the args we want to fetch reviews based on
        foreach ($elements as $key => $value) {
            // Break each of the element into key and value to allow use
to build a array to pass to url
            $brokenDown = explode("=", $value);
            if ($key !== 0 && sizeof($brokenDown) === 2) {
                $parameters[$brokenDown[0]] = $brokenDown[1];
            }
        }
        // We only require the field id to limit the amount of data.
        $parameters['fields'] = 'id';
        // Helpful for debugging which parameters we have passed in the
queue.
        $help['search_criteria'] = $parameters;
    }
    // Now make the request to the Swarm server with your field options.
    $result = request(
        "$swarmURL/api/v9/reviews",
```

```
        true,
        $parameters
    );

    // Return the JSON object back to be worked on.
    return $result;
}

/**
 * Loop though each of the Reviews passed in and run clean up for them.
 *
 * @param $reviews  JSON object of all the reviews we want to run cleanup
on
 * $reviews => array(
 *      'reviews' => array(
 *          0 => array (
 *                  'id' => 134,
 *          ),
 *          1 => array (
 *                  'id' => 136,
 *          ),
 *          2 => array (
 *                  'id' => 143,
 *          ),
 *          3 => array (
 *                  'id' => 158,
 *          )
 *      )
 * )
 *
 * @return $array   return the array of work that has been carried out.
 */
function runCleanUp($reviews)
{
    global $swarmURL;
```

```
    $results = array();
    // Check if there is an reviews element of the array
    if (isset($reviews['reviews'])) {
        foreach ($reviews['reviews'] as $review) {
            // Now make the request to the Swarm for each review.
            $results[$review['id']] = request(
                "$swarmURL/api/v9/reviews",
                false,
                $review
            );
        }
    }
    return $results;
}

/**
 * The help function in case a user doesn't set the basic settings.
 *
 * @param $help    Pass the help array from main script to append the
helpful message.
 * @return $array  Return the array that will be presented to the users.
 */
function helpMessage($help)
{
    $help["help"]      = array( "1" => "", "2" => "", "3" => "" );
    $help["help"]["1"] = "Please ensure you have set the Username, Ticket
and Swarm URL before using this script";
    $help["help"]["2"] = "Running the script can be done by using any of
the standard Swarm API fields for reviews";
    $help["help"]["3"] = "Visit
https://www.perforce.com/perforce/doc.current/manuals/swarm/index.html";
    return $help;
}

// check the first argument is not help.
```

```
$helpSet = isset($argv[1]) && $argv[1] == "help" ? "help" : null;

// Check if the user has given help as a command to this script.
if (isset($argv[1]) && $helpSet == "help") {
    // Set the $help array with the help message.
    $help = helpMessage($help);
}

// Check if the basic user ticket and swarmurl are set.
if (!empty($username) && !empty($ticket) && !empty($swarmURL) && $helpSet
!= "help") {
    try {
        $help["results"] = runCleanUp(fetchReviews($argv));
    } catch (Exception $e) {
        $help = helpMessage($help);
    }
} else {
    $message    = "Please ensure you have set the below before using this
script";
    $errorArray = array("message" => $message, "parameter" => "");

    $missingParameter = array();

    // Now check if the basic settings are empty and show the end user.
    empty($username) ? $missingParameter[] = "Username":'';
    empty($ticket) ? $missingParameter[] = "Ticket":'';
    empty($swarmURL) ? $missingParameter[] = "SwarmURL":'';

    $errorArray['parameter'] = $missingParameter;

    $help["error"] = $errorArray ;
}
// Output the end result of the what the script does.
echo json_encode($help, JSON_FORCE_OBJECT);
// @codingStandardsIgnoreEnd
```

Executing the script

The example is written in PHP, and demonstrates how to make use of the APIs which remove unneeded pending changelists. It **must** be run as a super user.

For a full set of instructions on how to use the example script, see the comments in the script itself.

Abbreviated instructions:

1. Set the value of the `$swarmURL`, `$username` and `$ticket` variables.
2. Run the script by using a command similar to the following:

```
$ php pendingReviewCleanUp.php max=10 author=bruno
state=approved
```

Swarm API endpoints (v10)

This section includes coverage for the endpoints provided by the API (v10).

Changes: Swarm changes

Get job information for jobs associated with a change

Summary

Get job information for jobs associated with a change

GET /api/v10/changes/{id}/jobs

Description

Get job information for jobs associated with a change.

Important
You must be authenticated to view the job information for a change.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.

- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Change ID	integer	path	Yes

Example usage

Get job information for change 12345

Get job information for all of the jobs associated with change 12345.

```
curl -u "username:<ticket>" "https://myswarm-url/api/v10/changes/12345/jobs"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "jobs": [{
        "job": "job000020",
        "link": "/jobs/job000998",
        "fixStatus": "open",
        "description": "Fix final bugs for beta release.\n",
        "description-markdown": "<span class=\"first-line\">Fix final bugs for beta release.</span>"
      },
      {
        "job": "job001001",
        "link": "/jobs/job001001",
        "fixStatus": "fixed",
        "description": "Write release notes for beta release.\n",
```

```
        "description-markdown": "<span class=\"first-line\">Write release
notes for beta release.</span>"
      }
    ]
  }
}
```

If a request fails

<error code>:

- **403** Insufficient permissions to access the change
- **404** Change does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Files: Swarm files

Get file data

Summary

Get file data for the specified file

GET /api/v10/files/{id}

Description

Get file data for the specified file.

Important
You must be authenticated to view the file information.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	File ID, this is the full filepath including the filename encoded to **URL safe Base64**. Various tools are available online to encode the `id` to **URL safe Base64**, for example: https://www.base64encode.org	string	path	Yes
`fileRevision`	Specify the revision of the file using one of the following options: ■ Specific file revision number: `fileRevision=#nnn` ■ File revision in a pending or submitted changelist: `fileRevision=@=nnn` ■ File revision in a submitted changelist only: `fileRevision=@nnn` Where `nnn` is the revision or changelist number you want.	string	query	No

Example usage

Get file data for a specific revision of a file

Get file data for `//depot/main/myfile.txt` revision 3. The full filepath and filename must be encoded in **URL safe Base64**.

```
curl -u "username:ticket" "https://myswarm-url/api/v10/files/Ly9kZXBvdC9tYWluL215ZmlsZS50eHQ?fileRevision=#3"
```

Swarm responds with:

```
HTTP/1.1 200 OK
```

```
{
  "error": null,
  "messages": [],
  "data" : {
    "filename":"//depot/main/myfile.txt"
    "contentLink": "http://myswarm-url/view/depot/main/myfile.txt",
    "fileRevision": "#3",
    "contentType": "text/plain",
    "changeId": "12871"
  }
}
```

If a request fails

<error code>:

- **400** Invalid file revision specified. Must be one of the following: `@` or `@=` followed by the changelist number, or **#** followed by the file revision number.
- **403** Insufficient permissions to access the file
- **404** File does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

File edit

> **Important**
> File edit is a Technology preview feature.
>
> **Technology preview features:**

Technology Preview features are currently unsupported, might not be functionally complete, and are not suitable for deployment in production. These features are provided to the customer to solicit interest and feedback, with the goal of full support in future releases. Customers are encouraged to provide feedback and functionality suggestions for Technology Preview features before they become fully supported.

Summary

Edit the file content and submit or shelve the file

PUT /api/v10/files/{id}

Description

Edit the file content and submit or shelve the file.

Note
Requirements:

- You must have permissions to edit the file.
- If the file version changes while you are editing it or if you are not editing the Head revision of the file, committing your file changes will overwrite the current head revision of the file.

  If you are unsure, shelving your change is a safer option because it will not overwrite the head revision of the file and you can do a manual resolve when you commit the shelf.
- File edit is enabled by default but can be disabled by your Swarm administrator, see "allow_edits" on page 500.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	File ID, this is the full filepath including the filename encoded to **URL safe Base64**. Various tools are available online to encode the `id` to **URL safe Base64**, for example: https://www.base64encode.org	string	path	Yes

Parameter	Description	Type	Parameter Type	Required
`content`	Specify the entire file content to be shelved or submitted. Lines in the content must be separated with backslash and n characters `\n`.	string	body	Yes
`description`	Specify a changelist description. **Optional:** you can create a review for your change or add it to an existing review by including a review keyword in the **Change description**. For instruction on creating a review and adding a changelist to a review using review keywords, see "Create a review" on page 548 and "Add a changelist to a review" on page 548.	string	body	No
`action`	Specify the file action as either `shelve` or `submit`.	string	body	Yes

Example usage

Shelve file with edited content

Shelve `//depot/main/myfile.txt` file with edited content. The full filepath and filename must be encoded in **URL safe Base64**.

```
curl -X PUT -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/files/Ly9kZXBvdC9tYWluL215ZmlsZS50eHQ"
```

The "mybodyfilename.txt" file contains the file content, description and the shelve action:

```
{
  "content": "My original line\nMy nice new line\nAnother original line"
  "description": "Added a new line."
  "action": "shelve"
}
```

Swarm responds with:

```
HTTP/1.1 200 OK
```

```
{
  "error": null,
  "messages": [],
  "data" : {
    "filename":"//depot/main/myfile.txt"
    "contentLink": "http://myswarm-url/view/depot/main/myfile.txt",
    "fileRevision": "@=12896",
    "contentType": "text/plain",
    "changeId": "12896"
  }
}
```

If a request fails

<error code>:

- **400** Invalid file revision specified. Must be one of the following: `@` or `@=` followed by the changelist number, or **#** followed by the file revision number.
- **403** Insufficient permissions to submit or shelve the file
- **404** File does not exist
- **500** Submit command failed because a newer revision of the file exists, you must resolve and submit your changes or revert your changes.

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Login: Swarm login

Login to Swarm with SAML

Summary

Login to Swarm with SAML

POST /api/v10/saml/login

Description

Login to Swarm with SAML.

Parameters

Parameter	Description	Type	Parameter Type	Required
`redirect`	Options are: ■ `true` or not specified: Swarm redirects the user to the `HTTP_REFERER` url or to the specified custom "logout_url" on page 498 if it has been set. ■ `false`: Swarm does not redirect the user.	string	query	No

Example usage

Login in to Swarm with SAML

Login in to Swarm with SAML.

```
curl -X POST -u "super:<ticket>" "https://myswarm-url/api/v10/saml/login"
```

On successful login Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
```

```
  "data" : {
    "isValid": "true",
    "url": "<url to redirect to>"
  }
}
```

Login in to Swarm with SAML and redirect=false

Login in to Swarm with SAML and `redirect=false`

```
curl -X POST -u "super:<ticket>" "https://myswarm-url/api/v10/saml/login?redirect=false"
```

On successful login Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "isValid": "true"
  }
}
```

If a request fails

<error code>:

400 Error occurred with the SAML login

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Projects : Swarm Projects

Get List of Projects

Summary

Get List of Projects and metadata

GET /api/v10/projects

Description

Returns a list of projects in Swarm that are visible to the current user.

Tip

- `tests` and `deploy` fields:
  - **If a project has an owner:** only project owners and users with *super* user permissions can view the `tests` and `deploy` fields when fetching projects.
  - **If a project does not have an owner:** only project members and users with *super* user permissions can view the `tests` and `deploy` fields when fetching projects.
- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list or array of top level fields to return for the projects. Omitting this parameter or passing an empty value shows all fields.	string	query	No

Example usage

Fetch a list of projects

To fetch all projects:

```
curl -u "username:password" "https://my-swarm-host/api/v10/projects"
```

Pagination is not currently supported by this endpoint. Swarm responds with a list of projects:


```
{
  "error": null,
  "messages": [],
  "data": {
    "projects": [
      {
        "id": "jam",
        "branches": [
          {
            "id": "main",
            "name": "Main",
            "workflow": null,
            "paths": [
              "//depot/Jam/MAIN/...",
              "//jam/main/..."
            ],
            "defaults": {
              "reviewers": []
            },
            "minimumUpVotes": null,
            "retainDefaultReviewers": false,
            "moderators": [],
            "moderators-groups": []
          },
          ...
          ...
        ],
        "defaults": {
          "reviewers": []
        },
        "deleted": false,
        "deploy": {
          "enabled": false,
```

```
          "url": ""
        },
        "description": "This the project Jam description",
        "emailFlags": {
          "change_email_project_users": "1",
          "review_email_project_members": "1"
        },
        "jobview": "",
        "members": [
          "Bruno",
          "raj"
        ],
        "minimumUpVotes": null,
        "name": "Jam",
        "owners": [],
        "private": false,
        "retainDefaultReviewers": false,
        "subgroups": [],
        "tests": {
          "enabled": false,
          "url": "",
          "postBody": "",
          "postFormat": "url"
        },
        "workflow": null
      },
      {
        ...
        <other projects formatted as above>
        ...
      },
    ],
  }
}
```

Fetch specified fields for all projects

To fetch the **id**, **description**, and **members** fields for all projects, use one of the following methods:

- **Using a comma-separated list of top level fields:**

  ```
  curl -u "username:password" "https://my-swarm-host/api/v10/projects?fields=id,description,members"
  ```

- **Using an array of top level fields:**

  ```
  curl -u "username:password" "https://my-swarm-host/api/v10/projects?fields[]=id&fields[]=description&fields[]=members"
  ```

Pagination is not currently supported by this endpoint. Swarm responds with a list of all projects:


```
{
  "error": null,
  "messages": [],
  "data": {
    "projects": [
      {
        "id": "blue-book",
        "description": "This is a private project for use only by users with sufficient clearance.",
        "members": [
          "alex.randolph",
          "allison.clayborne"
        ],
      },
      {
        "id": "jam",
        "description": "This the project Jam description.",
        "members": [
          "Bruno",
          "raj"
        ],
      },
      {
        "id": "jplugin",
```

```
        "description": "A Java plugin for continuous integration.",
        "members": [
          "allison.clayborne",
          "jack.boone",
          "steve.russell"
        ],
      },
    ],
  }
}
```

If a request fails

<error code>:

400 metadata request invalid, boolean required

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Get Project Information

Summary

Get Project Information

GET /api/v10/projects/{id}

Description

Retrieve all information about a project or specify which fields you want to return for the project.

Tip

- `tests` and `deploy` fields:
  - **If a project has an owner:** only project owners and users with *super* user permissions can view the `tests` and `deploy` fields when fetching projects.
  - **If a project does not have an owner:** only project members and users with *super* user permissions can view the `tests` and `deploy` fields when fetching projects.
- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Project ID	string	path	Yes
`fields`	An optional comma-separated list or array of top level fields to return for the project. Omitting this parameter or passing an empty value shows all fields.	string	query	No

Example usage

Fetch all fields for a project

To fetch all fields for project jam:

```
curl -u "username:password" "https://my-swarm-host/api/v10/projects/jam"
```

Swarm responds with a project entity:

```
{
  "error": null,
  "messages": [],
  "data": {
    "projects": [
      {
        "id": "jam",
```

```
        "branches": [
          {
            "id": "main",
            "name": "Main",
            "workflow": null,
            "paths": [
              "//depot/Jam/MAIN/...",
              "//jam/main/..."
            ],
            "defaults": {
              "reviewers": []
            },
            "minimumUpVotes": null,
            "retainDefaultReviewers": false,
            "moderators": [],
            "moderators-groups": []
          },
          ...
          ...
        ],
        "defaults": {
          "reviewers": []
        },
        "deleted": false,
        "deploy": {
          "enabled": false,
          "url": ""
        },
        "description": "This the project Jam description",
        "emailFlags": {
          "change_email_project_users": "1",
          "review_email_project_members": "1"
        },
        "jobview": "",
        "members": [
```

```
          "Bruno",
          "raj"
        ],
        "minimumUpVotes": null,
        "name": "Jam",
        "owners": [],
        "private": false,
        "retainDefaultReviewers": false,
        "subgroups": [],
        "tests": {
          "enabled": false,
          "url": "",
          "postBody": "",
          "postFormat": "url"
        },
        "workflow": null
      },
    ],
  }
}
```

Fetch specified fields for a project

To fetch the **id**, **description**, and **members** fields for project Jam, use one of the following methods:

- **Using a comma-separated list of top level fields:**

  ```
  curl -u "username:password" "https://my-swarm-host/api/v10/projects/jam?fields=id,description,members"
  ```

- **Using an array of top level fields:**

  ```
  curl -u "username:password" "https://my-swarm-host/api/v10/projects/jam?fields[]=id&fields[]=description&fields[]=members"
  ```

Swarm responds with a project entity containing the requested fields:

```
{
  "error": null,
  "messages": [],
```

```
  "data": {
    "projects": [
      {
        "id": "jam",
        "description": "This the project Jam description",
        "members": [
          "Bruno",
          "raj"
        ],
      },
    ],
  }
}
```

If a request fails

<error code>:

404 Project does not exist or you do not have permission to view it

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Reviews: Swarm reviews

Get a list of reviews

Summary

Get a list of reviews.

GET /api/v10/reviews

Description

List the reviews the user has permission to view.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`max`	Specify the maximum number of results to return as an integer greater than `0`. If the `max` parameter is not specified, the 50 most recent reviews are returned.	integer	query	No
`project`	To limit the results to a list of reviews in a specific project or projects, specify the projects you want reviews returned for.	string	query	No
`after`	Return reviews created after the *reviewid* specified in the `after` parameter. Reviews are returned in the order they were created in. **Note** You cannot use the `after` and `afterUpdated` parameters in the same request, they are mutually exclusive.	string	query	No

Parameter	Description	Type	Parameter Type	Required
`afterUpdated`	Return reviews updated on the day before the date/time specified in the `afterUpdated` parameter. The parameter value must be set in seconds since epoch. The seconds since epoch value for the day reviews are returned for is included at the end of the response. Reviews are returned in the order they were updated in, most recent first. Used by the Swarm team to step backwards one day at a time, loading reviews in an iterative process by using the value returned in a response as the `afterUpdated` parameter value for the next request. **For example:** ■ **Requested:** `afterUpdated=1606233362` (Tuesday November 24th, 2020, 15:56:02) ■ **Response fetched for:** any reviews updated on Monday November 23rd, 2020. `1606089600` (Monday, 23 November 2020 00:00:00) is added to the end of the response **Note** You cannot use the `after` and `afterUpdated` parameters in the same request, they are mutually exclusive.	string	query	No

Parameter	Description	Type	Parameter Type	Required
`state`	To limit the results to a list of reviews in a specific state or states, specify the states you want reviews returned for. **Valid review states are:** ■ `needsRevision` review needs revision ■ `needsReview` review needs review ■ `approved` review is approved ■ `approved:isPending` review is approved but not committed ■ `committed` review is committed ■ `approved:commit` review is approved and committed ■ `approved:notPending` review is approved and committed ■ `rejected` review is rejected ■ `archived` review is archived	string	query	No

Example usage

Get a list of reviews

```
curl -u "username:ticket" "https://myswarm-url/api/v10/reviews"
```

Swarm responds with the review entities:

> **Tip**
> By default, this is limited to the 50 most recent reviews. This can be changed by adding a **`max`** parameter value to the request.

```
HTTP/1.1 200 OK


{
```

```
"error": null,
"messages": [],
"data": {
  "reviews": [{
      "id": 12345,
      "type": "default",
      "changes": [
        12344,
      ],
      "commits": [],
      "author": "bruno",
      "approvals": null,
      "participants": ["bruno"],
      "participantsData": {
        "bruno": []
      },
      "hasReviewer": 0,
      "description": "Review for my code change",
      "created": 1594265070,
      "updated": 1594266228,
      "projects": {
        "myproject": ["main"]
      },
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testStatus": "fail",
      "testDetails": [],
      "deployStatus": null,
      "deployDetails": [],
      "pending": true,
      "commitStatus": [],
      "groups": [],
      "complexity": {
        "files_modified": 1,
        "lines_added": 0,
```

```
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
      ...
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 1,
    "lastSeen": 12345
  }
}
```

Get a list of the 25 most recent reviews

```
curl -u "username:ticket" "https://myswarm-url/api/v10/reviews?max=25"
```

Swarm responds with the 25 most recent review entities:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
        "type": "default",
        "changes": [
          12344,
        ],
        "commits": [],
        "author": "bruno",
        "approvals": null,
        "participants": ["bruno"],
        "participantsData": {
          "bruno": []
```

```
        },
        "hasReviewer": 0,
        "description": "Review for my code change",
        "created": 1594265070,
        "updated": 1594266228,
        "projects": {
          "myproject": ["main"]
        },
        "state": "needsReview",
        "stateLabel": "Needs Review",
        "testStatus": "fail",
        "testDetails": [],
        "deployStatus": null,
        "deployDetails": [],
        "pending": true,
        "commitStatus": [],
        "groups": [],
        "complexity": {
          "files_modified": 1,
          "lines_added": 0,
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
      ...
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 1,
    "lastSeen": 12345
  }
}
```

Get a list of reviews for the myproject project

```
curl -u "username:ticket" "https://myswarm-
url/api/v10/reviews?project=myproject"
```

Swarm responds with the review entities for the **`myproject`** project:

> **Tip**
> By default, this is limited to the 50 most recent reviews. This can be changed by adding a **`max`** parameter value to the request.

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
        "type": "default",
        "changes": [
          12344,
        ],
        "commits": [],
        "author": "bruno",
        "approvals": null,
        "participants": ["bruno"],
        "participantsData": {
          "bruno": []
        },
        "hasReviewer": 0,
        "description": "Review for my code change",
        "created": 1594265070,
        "updated": 1594266228,
        "projects": {
          "myproject": ["main"]
        },
        "state": "needsReview",
        "stateLabel": "Needs Review",
        "testStatus": "fail",
        "testDetails": [],
        "deployStatus": null,
```

```
        "deployDetails": [],
        "pending": true,
        "commitStatus": [],
        "groups": [],
        "complexity": {
          "files_modified": 1,
          "lines_added": 0,
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
      ...
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 1,
    "lastSeen": 12345
  }
}
```

Get a list of reviews for the myproject and gemini projects

```
curl -u "username:ticket" "https://myswarm-url/api/v10/reviews?project[]=myproject&project[]=gemini"
```

Swarm responds with the review entities for the **myproject** and **gemini** projects:

> **Tip**
> By default, this is limited to the 50 most recent reviews. This can be changed by adding a **max** parameter value to the request.

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
```

```
        "type": "default",
        "changes": [
          12344,
        ],
        "commits": [],
        "author": "bruno",
        "approvals": null,
        "participants": ["bruno"],
        "participantsData": {
          "bruno": []
        },
        "hasReviewer": 0,
        "description": "Review for my code change",
        "created": 1594265070,
        "updated": 1594266228,
        "projects": {
          "myproject": ["main"]
        },
        "state": "needsReview",
        "stateLabel": "Needs Review",
        "testStatus": "fail",
        "testDetails": [],
        "deployStatus": null,
        "deployDetails": [],
        "pending": true,
        "commitStatus": [],
        "groups": [],
        "complexity": {
          "files_modified": 1,
          "lines_added": 0,
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
        "id": 12356,
```

```
      "type": "default",
      "changes": [
        12350,
      ],
      "commits": [],
      "author": "jsmith",
      "approvals": null,
      "participants": ["jsmith"],
      "participantsData": {
        "jsmith": []
      },
      "hasReviewer": 0,
      "description": "Another one of my reviews",
      "created": 1594265070,
      "updated": 1594266228,
      "projects": {
        "gemini": ["main"]
      },
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testStatus": "pass",
      "testDetails": [],
      "deployStatus": null,
      "deployDetails": [],
      "pending": true,
      "commitStatus": [],
      "groups": [],
      "complexity": {
        "files_modified": 1,
        "lines_added": 0,
        "lines_edited": 1,
        "lines_deleted": 0
      },
    },
    ...
```

```
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 2,
    "lastSeen": 12356
  }
}
```

Get a list of reviews created after review 12344

`curl -u "username:ticket" "https://myswarm-url/api/v10/reviews?after=12344"`

Swarm responds with the review entities:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
        "type": "default",
        "changes": [
          12345,
        ],
        "commits": [],
        "author": "bruno",
        "approvals": null,
        "participants": ["bruno"],
        "participantsData": {
          "bruno": []
        },
        "hasReviewer": 0,
        "description": "Review for my code change",
        "created": 1594265070,
        "updated": 1594266228,
```

```
        "projects": {
          "myproject": ["main"]
        },
        "state": "needsReview",
        "stateLabel": "Needs Review",
        "testStatus": "fail",
        "testDetails": [],
        "deployStatus": null,
        "deployDetails": [],
        "pending": true,
        "commitStatus": [],
        "groups": [],
        "complexity": {
          "files_modified": 1,
          "lines_added": 0,
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
      ...
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 7,
    "lastSeen": 12352
  }
}
```

Get a list of reviews updated during the day before the set date

Request reviews updated on the day before `1606233362` (Tuesday November 24th, 2020, 15:56:02)

```
curl -u "username:ticket" "https://myswarm-url/api/v10/reviews?afterUpdated=1606233362"
```

Swarm return all reviews that were updated on the day before the date/time specified in your request. In this example, Swarm responds with any reviews that were updated on Monday November 23rd 2020. The seconds since epoch value Swarm used is added to the end of the response:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
        "type": "default",
        "changes": [
          12345,
        ],
        "commits": [],
        "author": "bruno",
        "approvals": null,
        "participants": ["bruno"],
        "participantsData": {
          "bruno": []
        },
        "hasReviewer": 0,
        "description": "Review for my code change",
        "created": 1594265070,
        "updated": 1606119044,
        "projects": {
          "myproject": ["main"]
        },
        "state": "needsReview",
        "stateLabel": "Needs Review",
        "testStatus": "fail",
        "testDetails": [],
        "deployStatus": null,
        "deployDetails": [],
        "pending": true,
        "commitStatus": [],
        "groups": [],
```

```
        "complexity": {
          "files_modified": 1,
          "lines_added": 0,
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
      ...
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 25,
    "lastSeen": "0",
    "afterUpdated": "1606089600"
  }
}
```

Get a list of reviews that need review

`curl -u "username:ticket" "https://myswarm-url/api/v10/reviews?state=needsReview"`

Swarm responds with the review entities in the **`needsReview`** state:

> **Tip**
> By default, this is limited to the 50 most recent reviews. This can be changed by adding a **`max`** parameter value to the request.

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
        "type": "default",
        "changes": [
          12344,
```

```
      ],
      "commits": [],
      "author": "bruno",
      "approvals": null,
      "participants": ["bruno"],
      "participantsData": {
        "bruno": []
      },
      "hasReviewer": 0,
      "description": "Review for my code change",
      "created": 1594265070,
      "updated": 1594266228,
      "projects": {
        "myproject": ["main"]
      },
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testStatus": "fail",
      "testDetails": [],
      "deployStatus": null,
      "deployDetails": [],
      "pending": true,
      "commitStatus": [],
      "groups": [],
      "complexity": {
        "files_modified": 1,
        "lines_added": 0,
        "lines_edited": 1,
        "lines_deleted": 0
      },
    },
    ...
    <other review ids formatted as above>
    ...
  ],
```

```
    "totalCount": 1,
    "lastSeen": 12345
  }
}
```

Get a list of reviews that are open

Reviews are considered open when they are in any one of three states:

- `needsReview`
- `needsRevision`
- `approved:isPending`

```
curl -u "username:ticket" "https://myswarm-
url/api/v10/reviews?state[]=approved:isPending&state
[]=needsReview&state[]=needsRevision"
```

Swarm responds with the review entities for open reviews:

> **Tip**
> By default, this is limited to the 50 most recent reviews. This can be changed by adding a **`max`** parameter value to the request.

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
        "id": 12345,
        "type": "default",
        "changes": [
          12344,
        ],
        "commits": [],
        "author": "bruno",
        "approvals": null,
        "participants": ["bruno"],
        "participantsData": {
```

```
        "bruno": []
      },
      "hasReviewer": 0,
      "description": "Review for my code change",
      "created": 1594265070,
      "updated": 1594266228,
      "projects": {
        "myproject": ["main"]
      },
      "state": "needsReview",
      "stateLabel": "Needs Review",
      "testStatus": "fail",
      "testDetails": [],
      "deployStatus": null,
      "deployDetails": [],
      "pending": true,
      "commitStatus": [],
      "groups": [],
      "complexity": {
        "files_modified": 1,
        "lines_added": 0,
        "lines_edited": 1,
        "lines_deleted": 0
      },
    },
      "id": 12356,
      "type": "default",
      "changes": [
        12350,
      ],
      "commits": [],
      "author": "jsmith",
      "approvals": null,
      "participants": ["jsmith"],
      "participantsData": {
```

```
          "jsmith": []
        },
        "hasReviewer": 0,
        "description": "Another one of my reviews",
        "created": 1594265070,
        "updated": 1594266228,
        "projects": {
          "gemini": ["main"]
        },
        "state": "needsRevision",
        "stateLabel": "Needs Revision",
        "testStatus": "pass",
        "testDetails": [],
        "deployStatus": null,
        "deployDetails": [],
        "pending": true,
        "commitStatus": [],
        "groups": [],
        "complexity": {
          "files_modified": 1,
          "lines_added": 0,
          "lines_edited": 1,
          "lines_deleted": 0
        },
      },
      ...
      <other review ids formatted as above>
      ...
    ],
    "totalCount": 2,
    "lastSeen": 12356
  }
}
```


If a request fails

<error code>:

- **400** the input specified must be an integer greater than 0
- **401** you are not authorized to view reviews

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Get review information

Summary

Get review information and metadata.

GET /api/v10/reviews/{id}

Description

Retrieve all of the information and metadata for a review.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Parameter	Description	Type	Parameter Type	Required
`transitions`	To limit the results to a list of the state transitions that are allowed for the review, add the `transitions` parameter to the request.	query	path	No

Example usage

Fetch information for review number 12345

```
curl -u "username:ticket" "https://myswarm-url/api/v10/reviews/12345"
```

Swarm responds with a review entity:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "reviews": [{
      "id": 12345,
      "type": "default",
      "changes": [1,3],
      "commits": [],
      "author": "normal-user",
      "approvals": null,
      "participants": ["normal-user"],
      "participantsData": {
        "normal-user": []
      },
      "hasReviewer": 0,
      "description": "Added a file\n",
      "created": 1576595749,
      "updated": 1576595754,
      "projects": [],
```

```
    "state": "needsReview",
    "stateLabel": "Needs Review",
    "transitions": {
      "needsRevision": "Needs Revision",
      "approved": "Approve",
      "approved:commit": "Approve and Commit",
      "rejected": "Reject",
      "archived": "Archive"
    },
    "testStatus": null,
    "testDetails": [],
    "deployStatus": null,
    "deployDetails": [],
    "pending": true,
    "commitStatus": [],
    "groups": [],
    "versions": [
      {
        "difference": 1,
        "stream": "//jam/main",
        "streamSpecDifference": 0,
        "change": 2,
        "user": "normal-user",
        "time": 1576595752,
        "pending": true,
        "addChangeMode": "replace",
        "archiveChange": 3,
        "testRuns": []
      },
      {
        ...
        <other "versions" formatted as above>
        ...
      }
    ],
```

```
      "description-markdown": "<span class=\"first-line\">Added a
file</span>"
    }]
  }
}
```

Fetch a list of the state transitions that are allowed for review 12345

```
curl -u "username:ticket" "https://myswarm-
url/api/v10/reviews/12345/transitions"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "transitions": {
      "needsRevision": "Needs Revision",
      "approved": "Approve",
      "approved:commit": "Approve and Commit",
      "rejected": "Reject",
      "archived": "Archive"
    }
  }
}
```

If a request fails

<error code>:

404 Review does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
```

```
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Get list of files that changed in a review

Summary

Get a list of files that changed between specified versions of a review.

GET /api/v10/reviews/{id}/files?from={x}&to={y}

Description

Get a list of files that changed between specified versions of a review.

The GET request returns the following:

- `root`: the common root depot path of the files that changed between the specified versions of the review.
- `limited`: Swarm limits the number of files presented for changelists based on the max_changelist_files setting for the Helix server connection. By default, a maximum of `1000` files are returned:
  - `true`: the number of files in the review is ≥ `max_changelist_files`. Some of the files changed in the specified review versions might not have been returned by this request.
  - `false`: the number of files in the review is < `max_changelist_files`. All of the files changed in the specified review versions have been returned by this request.
- **For each file:**
  - `depotFile`: the filename and path of the file.
  - `action`: how the file changed, **Add**, **Edit**, or **Delete**.
  - `type`: the filetype of the file, **text**, **binary**, **symlink**, **unicode**, **utf8**, **utf16**, **apple**, or **resource**.

    For more information on filetypes, see the Base filetypes section of the *Helix Core P4 Command Reference*.
  - `rev`: the file revision number.
  - `fileSize`: the file size in bytes.
  - `digest`: the file digest.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes
`from`	Specify a value for `from`: ■ `0` specifies the base version of the review. ■ `-1` specifies the current `#head` version of the review files in the Helix server. ■ If omitted, `from` defaults to `0` which is the base version of the review. ■ Must not be greater that the latest version of the review.	integer	query	No
`to`	Specify a value for `to`: ■ If omitted, `to` defaults to the latest version of the review. ■ Cannot be `0` or `-1` ■ Must not be greater that the latest version of the review.	integer	query	No

Example usage

Get a list of files that changed between version 1 and 3 of review 12345

```
curl -u "username:ticket" "https://myswarm-
url/api/v10/reviews/12345/files?from=1&to=3"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "root": "//jam/main/src",
    "limited": false,
    "files": [{
        "depotFile": "//jam/main/src/execvms.c",
        "action": "edit",
        "type": "text",
        "rev": "3",
        "fileSize": "3343",
        "digest": "82D5161601D12DB46F184D3F0778A16D"
      },
      {
        "depotFile": "//jam/main/src/jam.h",
        "action": "add",
        "type": "text",
        "rev": "2",
        "fileSize": "7364",
        "digest": "2E94B379F201AD02CF7E8EE33FB7DA99"
      }
    ]
  }
}
```

If a request fails

<error code>:

- **400** invalid review version number requested
- **403** Insufficient permissions to access the review
- **404** Review does not exist

```
HTTP/1.1 <response error code>
```

```
{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Set vote for the authenticated user

Summary

Set the vote for the authenticated user to be up, down, or cleared

POST /api/v10/reviews/{id}/vote

Description

Set the vote for the authenticated user to be up, down, or cleared.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes
`vote`	Specify the `vote` value you want to set on the review. Valid vote values are `up`, `down`, and `clear`. If the requested vote value already exists on the latest version of the review, the vote is not changed.	string	body	Yes

Parameter	Description	Type	Parameter Type	Required
`version`	Votes can only be set on the latest version of a review. The `version` parameter is used to check that you are voting on the latest version of the review. If the specified `version` is not the latest version of the review, the vote is not applied to the review and a `400` error is returned. If the `version` parameter is not included, the vote is applied to the latest version of the review.	integer	body	No

Example usage

Vote up version 1 of review 12345

```
curl -X POST -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/reviews/12345/vote"
```

The "mybodyfilename.txt" file contains the authenticated user's `vote` and the `version` of the review they are voting on:

```
{
  "vote": "up"
  "version": "1"
}
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "vote": {
      "value": 1,
      "version": 1,
      "isStale": false
```

```
    }
  }
}
```

If a request fails

<error code>:

- **400** Version specified is not the latest version of the review
- **401** User must be authenticated
- **403** Insufficient permissions to access the review
- **404** Review does not exist
- **409** Invalid vote specified, the response error message lists valid votes.

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Change review state

Summary

Change the state of a review.

POST /api/v10/reviews/{id}/transition

Description

Change the state of a review. The state that a review can be changed to depends on a number of factors including its current state. You can check that the state change you want to make is allowed for the review by making a "Get review information" on page 897 request first. For more information on review states and restrictions on changing them, see "States" on page 387

Review state transitions:

- `needsRevision`
- `needsReview`
- `approved` only available if the voting requirements for the review are satisfied
- `committed` only available for "Pre-commit model" on page 338 reviews that have been approved
- `approved:commit` only available for unapproved "Pre-commit model" on page 338 reviews when the voting requirements for the review are satisfied
- `rejected`
- `archived`

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Change the state of review 12345 to approved

```
curl -X POST -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/reviews/12345/transition"
```

The "mybodyfilename.txt" file contains the state you want to change the review to:

```
{
  "transition": "approved"
}
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "state": "approved",
    "stateLabel": "Approved"
  }
}
```

If a request fails

<error code>:

- **400** Specified review state does not exist
- **401** Insufficient permissions to change review state
- **403** Insufficient permissions to access the review
- **404** Review does not exist
- **409** Specified state is not a valid transition for this review. API response message lists all valid transition states for the review

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Change review author

Summary

Change review author.

PUT /api/v10/reviews/{id}/author

Description

Change the review author.

> **Important**
> By default you cannot change the author of a review, this option must be enabled by your Swarm administrator. See "Allow author change" on page 556 for details.

> **Tip**
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Change the author of review 12345

```
curl -X PUT -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/reviews/12345/author"
```

The "mybodyfilename.txt" file contains the new author username:

```
{
  "author": "asmith"
}
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
```

```
  "data": {
    "author": "asmith"
  }
}
```

If a request fails

<error code>:

- **400** Author change not allowed, set **allow_author_change** to **true**
- **401** Insufficient permissions to change author
- **403** Insufficient permissions to access the review
- **404** Review does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Replace review description

Summary

Replace review description.

PUT /api/v10/reviews/{id}/description

Description

Replace the review description with a new description.

Optional: also replace pending changelist description

Only available if you are editing the description of a pre-commit review and you are the original author of the changelist that created the review.

To also apply your review description changes to the original changelist description, specify `'updateOriginalChangelist' : true`.

> **Tip**
>
> - **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
> - **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Replace the description for review 12345

```
curl -X PUT -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/reviews/12345/description"
```

The "mybodyfilename.txt" file contains the new description:

```
{
  "description": "This is my replacement description text and it is better
than the original description."
}
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "description": "This is my replacement description text and it is
better than the original description.",
```

```
    "description-markdown": "<span class=\"first-line\">This is my
replacement description text and it is better than the original
description.</span>"
  }
}
```

If a request fails

<error code>:

- **401** Insufficient permissions to change description
- **404** Review does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Replace review participants

Summary

Replace review participants.

PUT /api/v10/reviews/{id}/participants

Description

Replace the review participants with a new set of participants.

> **Note**
> - The text file used for your request must contain a complete list of all of the participants you want on the review (**users** and **groups**). The existing review participants are completely replaced by the participants in the text file. For example, if you already have some group

participants on the review and you do not specify any **group** roles in the text file, all of the participant groups will be removed from the review.

- Default reviewers can be removed from a review but retained default reviewers cannot be removed from a review, see "Default reviewers" on page 415 and "Retain default reviewers" on page 416.

  If your request tries to remove a retained default reviewer, your request will succeed but with a message in the response indicating that the retained default reviewer has not been removed.
- You cannot reduce the voting requirement of a retained default reviewer but you can increase it, see "Retain default reviewers" on page 416.

  If your request tries to reduce the voting requirement of a retained default reviewer, your request will succeed but with a message in the response indicating that the retained default reviewer has not been changed.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Replace participants for review 12345

```
curl -X PUT -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/reviews/12345/participants"
```

The "mybodyfilename.txt" file contains the new review participants and their voting requirements:

```
{
  "participants": {
    "users": {
```

```
      "asmith": {
        "required": "yes"
      },
      "jbloggs": {
        "required": "no"
      },
      "molsen": {
        "required": "no"
      }
    },
    "groups": {
      "docs": {
        "required": "none"
      },
      "qa": {
        "required": "all"
      },
      "dev": {
        "required": "one"
      },
      "testers": {
        "required": "one"
      }
    }
  }
}
```

Swarm responds with a list of all of the participants and their voting requirements:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": {
    "combinedReviewers": [
      "'molsen' must be a reviewer with a minimum requirement of
'required'",
```

```
      "'testers' must be a reviewer with a minimum requirement of 'require
all'"
    ]
  },
  "data": {
    "participants": {
      "users": {
        "asmith": {
          "required": "yes"
        },
        "jbloggs": {
          "required": "no"
        },
        "molsen": {
          "required": "yes",
          "minimumRequired": "yes"
        }
      },
      "groups": {
        "dev": {
          "required": "one"
        },
        "docs": {
          "required": "none"
        },
        "qa": {
          "required": "all"
        },
        "testers": {
          "required": "all",
          "minimumRequired": "all"
        }
      }
    }
  }
```

```
}
```

If a request fails

<error code>:

- **400** Invalid participant data, for example: unknown participant id or voting requirement
- **401** Insufficient permissions to replace participants
- **403** Insufficient permissions to edit reviewers
- **404** Review does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Append review participants

Summary

Append review participants.

POST /api/v10/reviews/{id}/participants

Description

Append more participants to the existing review participants.

> **Note**
> Only the users and groups specified in the request are appended to the review. For example, if you don't want to append any group participants to the review there is no need to include the **groups** role in the text file.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Append participants to review 12345

```
curl -X POST -H "Content-Type: application/json" -u "username:ticket" -d "@mybodyfilename.txt" "https://myswarm-url/api/v10/reviews/12345/participants"
```

The "mybodyfilename.txt" file contains the new review participants and their voting requirements. In this example, `bjones` is an optional participant:

```
{
  "participants": {
    "users": {
      "bjones": []
    }
  }
}
```

Swarm responds with a list of all of the participants and their voting requirements:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
```

```
    "participants": {
      "users": {
        "asmith": {
          "required": "yes"
        },
        "jbloggs": {
          "required": "no"
        },
        "molsen": {
          "required": "yes",
          "minimumRequired": "yes"
        },
        "bjones": []
      },
      "groups": {
        "dev": {
          "required": "one"
        },
        "docs": {
          "required": "none"
        },
        "qa": {
          "required": "all"
        },
        "testers": {
          "required": "all",
          "minimumRequired": "all"
        }
      }
    }
  }
}
```

If a request fails

<error code>:

- **400** Invalid participant data, for example: unknown participant id or voting requirement
- **401** Insufficient permissions to append participants
- **403** Insufficient permissions to edit reviewers
- **404** Review does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Delete review participants

Summary

Delete review participants.

DELETE /api/v10/reviews/{id}/participants

Description

Delete review participants from a review.

Note

- Only the users and groups specified in the request are deleted from the review. For example, if you don't want to delete any group participants from the review there is no need to include the **groups** role in the text file.
- Default reviewers can be removed from a review but retained default reviewers cannot be removed from a review, see "Default reviewers" on page 415 and "Retain default reviewers" on page 416.

  If your request tries to remove a retained default reviewer, your request will succeed but with a message in the response indicating that the retained default reviewer has not been removed.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Delete participants from review 12345

```
curl -X DELETE -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/reviews/12345/participants"
```

The "mybodyfilename.txt" file contains the participants you want to delete from the review:

```
{
  "participants": {
    "users": {
      "jbloggs": [],
      "molsen": []
    },
    "groups": {
      "docs": []
    }
  }
}
```

Swarm responds with a list of all of the participants and their voting requirements:

```
HTTP/1.1 200 OK


{
```

```
  "error": null,
  "messages": {
    "combinedReviewers": [
      "'molsen' must be a reviewer with a minimum requirement of
'required'"
    ]
  },
  "data": {
    "participants": {
      "users": {
        "asmith": {
          "required": "yes"
        },
        "molsen": {
          "required": "yes",
          "minimumRequired": "yes"
        },
        "bjones": []
      },
      "groups": {
        "dev": {
          "required": "one"
        },
        "qa": {
          "required": "all"
        },
        "testers": {
          "required": "all",
          "minimumRequired": "all"
        }
      }
    }
  }
}
```

If a request fails

<error code>:

- **400** Unknown participant id
- **401** Insufficient permissions to delete participants
- **403** Insufficient permissions to edit reviewers
- **404** Review does not exist

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Obliterate a review

Summary

Obliterate a review.

DELETE /api/v10/reviews/{id}

Description

Important
Obliterate must be used with care, the review and all of its associated metadata are permanently deleted. An obliterated review cannot be reinstated, not even by Perforce Support.

Note

- By default, you must be a user with *admin* or *super* user rights to obliterate a review.
- **Optional:** Swarm can be configured to allow users to obliterate reviews that they have authored. Configured by your Swarm administrator, see "Allow author obliterate review" on page 556.

Obliterate is used to permanently delete reviews that have been created by mistake. For instance, if a review is associated with the wrong changelist, or a review contains sensitive information that should not be openly available.

For information on what happens to a review when it is obliterated, see "When you obliterate a review" on page 533.

Tip

- **Private projects:** when a project is made private, only the project owners, moderators, and members, plus users with *super*-level privileges in the Helix server, can see the project, its activity streams, and ongoing reviews. API responses honor these private project restrictions. For more information about private project restrictions, see "Private projects" on page 306.
- **Permissions:** if a public review contains files that the user does not have permission to view, the review is returned but the restricted files are not displayed.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	Review ID	integer	path	Yes

Example usage

Obliterate review number 12345

```
curl -X DELETE -u "username:ticket" "https://myswarm-url/api/v10/reviews/12345"
```

Swarm responds with a message informing you that it has successfully Obliterated the review:

```
HTTP/1.1 200 OK


{
  "error": null,
  "messages": [{
    "code": "review-obliterated",
    "text": "The review with id [12345] has been obliterated."
  }],
  "data": {
    "reviews": [{
      "id": 12345,
```

```
    "type": "default",
    "changes": [1, 3],
    "commits": [],
    "author": "normal-user",
    "approvals": null,
    "participants": ["normal-user"],
    "participantsData": {
      "normal-user": []
    },
    "hasReviewer": 0,
    "description": "Hello\n",
    "created": 1576579928,
    "updated": 1576579932,
    "projects": [],
    "state": "needsReview",
    "stateLabel": "Needs Review",
    "testStatus": null,
    "testDetails": [],
    "deployStatus": null,
    "deployDetails": [],
    "pending": true,
    "commitStatus": [],
    "groups": []
    "versions": [
      {
        "difference": 1,
        "stream": "//jam/main",
        "streamSpecDifference": 0
        "change": 2,
        "user": "normal-user",
        "time": 1576595752,
        "pending": true,
        "addChangeMode": "replace",
        "archiveChange": 3
        "testRuns": []
```

```
        },
        {
          ...
          <other "versions" formatted as above>
          ...
        }
      }
    }
  }
}
```

Obliterate review number 56789 (with insufficient permission)

`curl -X DELETE -u "username:ticket" "https://myswarm-url/api/v10/reviews/56789"`

Swarm responds with a message informing you that you do not have permission to obliterate the review:

```
HTTP/1.1 200 OK

{
  "error":403,
  "messages":[{
    "code": "forbidden-exception",
    "text": "Failed to Obliterate the review, you need to have admin privileges or be the review author with 'allow_author_obliterate' set to true."
  }]
  "data": null
}
```

If a request fails

`<error code>`":

- **403** Insufficient permissions to obliterate review
- **404** Review does not exist

```
HTTP/1.1 <response error code>
```

```
{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Search: Swarm search

Search Redis cache for users and groups

Summary

Search the Redis cache for users and groups.

GET /api/v10/search?term=<term>&context=user,group

Description

Search the Redis cache for *userid*s, full names, *groupid*s, and group names. Results are returned in alphabetical order and the search is case insensitive.

Parameter	Description	Type	Parameter Type	Required
`term`	Specifies the characters to search for. Searches are case insensitive.	string	path	Yes
`context`	Limit the search results, valid context values are `user` and `group`. Separate context values with a comma (`,`).	string	path	Yes

Parameter	Description	Type	Parameter Type	Required
`starts_with_only`	Options are: ■ `true`: searches for users/groups that start with the search `term`. This results in a faster search than when this parameter is set to `false`. ■ `false`: searches for the `term` anywhere in users/groups. This is the search used if `starts_with_only` is not included in the request.	boolean	path	No

Parameter	Description	Type	Parameter Type	Required
`limit`	Limit the combined maximum number of search results returned for users and groups. Search results are balanced across the contexts requested and will never return more results than the specified `limit`. **Examples:** based on a search that results in 8 user matches and 30 group matches before the `limit` is applied: ■ `limit=7` and `context=user`: search result is limited to 7 users. ■ `limit=6` and `context=user,group`: search result is limited to 3 users and 3 groups. An even limit value splits the search result limit evenly between the specified contexts. ■ `limit=7` and `context=user,group`: search result is limited to 4 users and 3 groups. An odd limit value splits the search result limit unevenly between the specified contexts with the higher number applied to the first context in the list. ■ `limit=20` and `context=user,gro`	integer	path	No

Parameter	Description	Type	Parameter Type	Required
	`up`, search result is limited to 8 users and 12 groups. Search results are not limited when the `limit` parameter is excluded from the request.			
`ignoreExcludeList`	Options are: ■ `true`: the `user_exclude_list` and `group_exclude_list` filters are ignored and are not applied to the search results. ■ `false`: applies the user_exclude_list and group_exclude_list filters to the search results. This is the search used if `ignoreExcludeList` is not included in the request.	boolean	path	No

Example usage

Search users and groups starting with "jo" with a limit of 3 results returned

Search users and groups starting with "jo" with a limit of 3 results returned.

```
curl -u "username:ticket" "https://myswarm-url/api/v10/search?starts_with_only=true&term=jo&context=user,group&limit=3"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
```

```
    "user": [{
        "User": "joakly",
        "FullName": "Jane Oakly"
      },
      {
        "User": "jsmith",
        "FullName": "John Smith"
      }
    ],
    "group": [{
      "Group": "jogd",
      "name": "JOGD"
    }],
    "limit": 3
  }
}
```

Search groups for "admin" anywhere in group names and *groupid*s

Search for "admin" anywhere in group names and *groupid*s with no limit on the number of results returned by the search.

```
curl -u "username:ticket" "https://myswarm-url/api/v10/search?context=group&term=admin"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "group": [
      {
        "Group": "swarm-admin",
        "name": "Swarm-admin"
      },
      {
        "Group": "jplugin-admin",
```

```
        "name": "JPlugin-Admin"
      },
      {
        "Group": "administrators",
        "name": "Administrators"
      },
      {
        "Group": "sysadmins",
        "name": "Sysadmins"
      }
    ]
  }
}
```

If a request fails

<error code>:

> **422** Parameter not specified or invalid, the error message in the response contains the error details.

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Specs: Spec fields

Get spec information

Summary

Get spec file fields.

GET /api/v10/specs/{spec}/ fields

Description

Get fields for a spec type.

Parameter	Description	Type	Parameter Type	Required
`spec`	Spec type. Valid spec types and location in the Helix server spec depot: ■ `branch`: `//spec/branch/` ■ `change`: the change spec is not stored in the Helix server spec depot ■ `client`: `//spec/client/` ■ `depot`: `//spec/depot/` ■ `group`: `//spec/group/` ■ `job`: `//spec/job/` ■ `label`: `//spec/label/` ■ `license`: `//spec/license.p4s` ■ `protect`: `//spec/protect.p4s` ■ `spec`: `//spec/spec/` ■ `stream`: `//spec/stream/` ■ `triggers`: `//spec/triggers.p4s` ■ `typemap`: `//spec/typemap.p4s` ■ `user`: `//spec/user/`	string	path	Yes

Parameter	Description	Type	Parameter Type	Required
`fields`	Omitting the `fields` parameter or including the `fields` parameter with no fields specified, returns all of the fields in the spec including any custom fields. Limit the response to specific fields by including a list of comma-separated fields in the request. The fields available depend on the spec type requested. Invalid fields do not cause an error and are ignored.	array of strings	path	No

Example usage

Fetch all fields for the job spec

Fetching all fields for the job spec

```
curl -u "username:ticket" "https://myswarm-url/api/v10/specs/job/fields"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "job": {
      "Job": {
        "code": "101",
        "dataType": "word",
        "displayLength": "32",
        "fieldType": "required"
      },
      "Status": {
        "code": "102",
        "dataType": "select",
```

```
        "displayLength": "10",
        "fieldType": "required",
        "options": [
          "open",
          "suspended",
          "fixed",
          "closed"
        ],
        "default": "open"
      },
      "Created_by": {
        "code": "103",
        "dataType": "word",
        "displayLength": "32",
        "fieldType": "once",
        "default": "$user"
      },
      "Description": {
        "code": "105",
        "dataType": "text",
        "displayLength": "0",
        "fieldType": "required",
        "default": "$blank"
      },
      "Date_modified": {
        "code": "130",
        "dataType": "date",
        "displayLength": "20",
        "fieldType": "always",
        "default": "$now"
      }
    }
  }
}
```

Limit results to job and status fields for the job spec

Limit the results to the job and status fields for the job spec.

```
curl -u "username:ticket" "https://myswarm-url/api/v10/specs/job/fields?fields=status,job"
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "job": {
      "Job": {
        "code": "101",
        "dataType": "word",
        "displayLength": "32",
        "fieldType": "required"
      },
      "Status": {
        "code": "102",
        "dataType": "select",
        "displayLength": "10",
        "fieldType": "required",
        "options": [
          "open",
          "suspended",
          "fixed",
          "closed"
        ],
        "default": "open"
      }
    }
  }
}
```

Limit results to depot and type fields for the depot spec

Limit the results to the depot and type fields for the depot spec.

```
curl -u "username:ticket" "https://myswarm-url/api/v10/specs/depot/fields?fields=depot,type"
```

Swarm responds with:

```
HTTP/1.1 200 OK


{
  "error": null,
  "messages": [],
  "data": {
    "depot": {
      "Depot": {
        "code": "251",
        "dataType": "word",
        "displayLength": "32",
        "fieldType": "key"
      },
      "Type": {
        "code": "255",
        "dataType": "word",
        "displayLength": "10",
        "fieldType": "required"
      }
    }
  }
}
```

Limit results to status and type fields for the change spec

Limit the results to the status and type fields for the change spec.

```
curl -u "username:ticket" "https://myswarm-url/api/v10/specs/change/fields?fields=status,type"
```

Swarm responds with:

```
HTTP/1.1 200 OK
```

```
{
  "error": null,
  "messages": [],
  "data": {
    "change": {
      "Status": {
        "code": "205",
        "dataType": "word",
        "displayLength": "10",
        "fieldType": "always",
        "order": "5",
        "position": "R"
      },
      "Type": {
        "code": "211",
        "dataType": "select",
        "displayLength": "10",
        "fieldType": "optional",
        "order": "6",
        "position": "L",
        "options": [
          "public",
          "restricted"
        ]
      }
    }
  }
}
```


If a request fails

<error code>:

- **400** Unknown spec type requested
- **401** Insufficient permissions to fetch spec fields

```
HTTP/1.1 <response error code>
```

```
{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Testdefinitions: Swarm Test definitions

Get a list of test definitions

Summary

Get a list of test definitions and test fields

GET /api/v10/testdefinitions

Description

Returns a list of test definitions in Swarm that are visible to the current user.

> **Tip**
> Shared test definitions only display the test **`url`**, **`body`**, **`headers`**, and associated fields if you are the test definition owner. By default, this private data is visible in key data for users with list-level access. To hide key data, see "Hiding Swarm storage from regular users" on page 173.

Example usage

Get a list of Swarm test definitions:

```
curl -u "username:password" "https://my-swarm-host/api/v10/testdefinitions"
```

Swarm responds with a list of all of its test definitions:

In the example response below, you own testdefinition **`id1`** but you do not own testdefinition **`id2`**. The test **`url`**, **`body`**, **`headers`**, and associated fields are not returned for **`id2`** because you do not own that testdefinition.

```
HTTP/1.1 200 OK
```

```
{
  "error": null,
  "messages": [],
  "data" : {
    testdefinitions: [
      {
        "id": 1,
        "name": "Main code test",
        "headers": {
          "BasicAuth": "YWxpY2U6QUJDRDEyMzQ1Njc4Cg=="
        },
        "encoding": "url",
        "body": "status={status}&change={change}&update={update}",
        "url": "http://jenkins_host:8080/job/{branches}-review-
test/review/build",
        "timeout": 20,
        "owners": [
          "allison"
        ],
        "shared": false,
        "description": "Test for main code"
      },
      {
        "id": 2,
        "name": "Doc tests",
        "owners": [
          "bruno"
        ],
        "shared": true,
        "description": "Tests to check that the docs build."
      }
    ]
  }
}
```

If the request fails

<error code>:

501 Workflows must be enabled for testdefinitions. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Get test definition information

Summary

Get the details of a test definition

GET /api/v10/testdefinitions/{id}

Description

Returns the specified test definition if it is visible to the current user.

> **Tip**
> Shared test definitions only display the test **url**, **body**, **headers**, and associated fields if you are the test definition owner. By default, this private data is visible in key data for users with list-level access. To hide key data, see "Hiding Swarm storage from regular users" on page 173.

Parameters

Parameter	Description	Type	Parameter Type	Required
id	Testdefinition ID	integer	path	Yes

Example usage

Get test definition 1:

```
curl -u "username:password" "https://my-swarm-host/api/v10/testdefinitions/1"
```

Swarm responds with details for the test definition:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    testdefinitions: [
      {
        "id": 1,
        "name": "Main code test",
        "headers": {
          "BasicAuth": "YWxpY2U6QUJDRDEyMzQ1Njc4Cg=="
        },
        "encoding": "url",
        "body": "status={status}&change={change}&update={update}",
        "url": "http://jenkins_host:8080/job/{branches}-review-test/review/build",
        "timeout": 20,
        "owners": [
          "allison"
        ],
        "shared": false,
        "description": "Test for main code"
      }
    ]
  }
}
```

If the request fails

<error code>:

- **404** Test definition does not exist or you do not have permission to view it.
- **501** Workflows must be enabled for testdefinitions. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Create a test definition

Summary

Create a test definition

POST /api/v10/testdefinitions

Description

Create a test definition.

Parameters

Parameter	Description	Type	Parameter Type	Required
name	Provide a name for your test. This should be a unique name that is 1 to 32 characters in length.	string	body	Yes
description	Provide a description for the test.	string	body	No
headers	Specify name=value pairs to pass to the test suite.	array of strings	body	No

Parameter	Description	Type	Parameter Type	Required
`encoding`	Specify the encoding method for the request `body` parameter: ■ `url` for URL Encoded: POST parameters are parsed into name=value pairs. This is the default. ■ `json` for JSON Encoded: parameters are passed raw in the POST body. ■ `xml` for XML Encoded: parameters are passed in XML format in the POST body.	string	body	No (required if `body` parameter used)
`body`	Specify parameters that your test suite requires in the request `body` parameter. Encoding is set with the `encoding` parameter.	string	body	No
`url`	Specify the test request URL that will trigger your test suite using the `url` parameter. 1 to 1024 characters. Special arguments are also available to pass details from Swarm to your test suite, see "Pass special arguments to your test suite" on page 443. **Note** **Helix Plugin for Jenkins 1.10.11 and later:** Swarm must send the parameters for the build to Jenkins as a POST request. To do this, specify the parameters in the `body` and specify `url` for `encoding`.	string	body	Yes

Parameter	Description	Type	Parameter Type	Required
`timeout`	Specify a timeout in seconds. The timeout is used when Swarm connects to your test suite. If a timeout is not set, the http_client_options `timeout` setting is used. By default, this is 10 seconds.	integer	body	No
`owners`	Specify at least one owner for the test.	array of strings	body	Yes
`shared`	Specify if want other Swarm users to be able to view or use this test definition: ■ `true`: other Swarm users can use this test. ■ `false` or not or not specified: other Swarm users will not be able to see or use this test.	string	body	No

Example usage

Create a test definition

```
curl -X POST -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt"
"https://myswarm.url/api/v10/testdefinitions"
```

The "mybodyfilename.txt" file contains the test definition details:

```
{
  "name": "My test def",
  "description": "My test definition description",
  "headers": {
    "BasicAuth": "user:password",
    "OtherHeader": "OtherValue"
  },
  "encoding": "json",
  "body": "{\"test\":\"{test}\"}",
  "url": "http://my-test-host/api/v10/testruns/test?test={test}",
```

```
  "timeout": 10,
  "owners" : ["bruno"],
  "shared": true
}
```

Swarm responds with the testdefinition details:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "testdefinitions": [
      {
        "id": 3,
        "name": "My test def",
        "headers": {
          "BasicAuth": "user:password",
          "OtherHeader": "OtherValue"
        },
        "encoding": "json",
        "body": "{\"test\":\"{test}\"}",
        "url": "http://my-test-host/api/v10/testruns/test?test={test}",
        "timeout": 10,
        "owners": [
          "bruno"
        ],
        "shared": true,
        "description": "My test definition description"
      }
    ]
  }
}
```

If the request fails

<error code>:

400 one of the following:

- The test definition **name** parameter is required and must be unique.
- The test definition **url** and **owners** parameters are required and cannot be empty.
- The **encoding**, **shared**, and **description** parameters require a value and cannot be empty.

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Testruns: Swarm Test Integration

Get testrun details of a review version

Summary

Get the testrun details of a review version.

GET /api/v10/reviews/{reviewId}/testruns

Description

Used to get the testrun details for a review version.

> **Note**
> Any users can get test run details for a review version, the user does not need to be authenticated. The test run **uuid** is not included in the request response.

Parameters

Parameter	Description	Type	Parameter Type	Required
`reviewId`	Review ID	integer	path	Yes
`version`	An optional parameter that enables you to specify the review version you want to return the testrun details for. Omitting this parameter or passing an empty value shows testrun details for the latest version of the review.	string	query	No

Example of usage

Get details for all of the test runs for the latest version of review 12345

```
curl "https://myswarm.url/api/v10/reviews/12345/testruns"
```

Swarm responds with the testrun details:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "testruns" : [
      {
        "id": 706,
        "change": 12345,
        "version": 2,
        "test": "global1",
        "startTime": 1567895432,
        "completedTime": 1567895562,
        "status": "pass",
        "messages": [
          "Test completed successfully",
          "another message"
        ],
```

```
        "url": "http://my.jenkins.com/projectx/main/1224"
        "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
      },
      {
        <ids for other testruns of the review, formatted as above>
      }
    ]
  }
}
```

Get details for all of the test runs for version 2 of review 12345

```
curl "https://myswarm.url/api/v10/reviews/12345/testruns?version=2"
```

Swarm responds with the testrun details:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "testruns" : [
      {
        "id": 706,
        "change": 12345,
        "version": 2,
        "test": "global1",
        "startTime": 1567895432,
        "completedTime": 1567895562,
        "status": "pass",
        "messages": [
          "Test completed successfully",
          "another message"
        ],
        "url": "http://my.jenkins.com/projectx/main/1224"
        "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
```

```
      },
      {
        <ids for other testruns for version 2 of the review, formatted as
above>
      }
    ]
    "status" : "pass"
  }
}
```

Tip
When you make a GET request for all of the testruns in a specific review version, Swarm includes a **`"status"`** for the review version as a whole in the response. This is shown after the individual testruns. This status is calculated from the **`"status"`** values of all of the individual testruns for that review version:

- If the test **`"status"`** for any of the testruns is **`"running"`** then the overall result for that version is **`"running"`**.
- If no testruns are **`"running"`** and the test **`"status"`** of any of the testruns is **`"fail"`** the overall result for that version is **`"fail"`**.
- If the test **`"status"`** of all of the testruns is **`"pass"`** then the overall result for that version is **`"pass"`**.

If a request fails

```
HTTP/1.1 <response error code>

{
  "error": <high level description>,
  "messages": [{"code" : "<code string>", "msg" : "<error message>"}],
  "data" : null
}
```

Create a testrun for a review version

Summary

Create a testrun for a review

POST /api/v10/reviews/{reviewid}/testruns

Description

Create a testrun for a review version. After you have created a testrun for a version of a review, you can update the testrun details as the test progresses using the PATCH and PUT API endpoints or the `{pass}`, `{fail}` and `{update}` callback urls.

> **Important**
> Only *admin* users can create a test run for a review.

Parameters

Parameter	Description	Type	Parameter Type	Required	Restriction
`reviewid`	Review ID.	integer	path	Yes	
`version`	The version of the review the testrun is being run against.	integer	body	Yes	
`test`	Specifies the test used for the testrun and checks that it is a valid name.	string	body	Yes	1 to 32 characters
`startTime`	The time the testrun started, expressed as an epoch value.	integer	body	Yes	> 0
`completedTime`	The time the testrun completed, expressed as an epoch value.	integer	body	No	> 0
`status`	The status of the testrun, options are: pass, fail, and running	string	body	No	Valid options are pass, fail, and running

Parameter	Description	Type	Parameter Type	Required	Restriction
`messages`	An array of one or more messages for the testrun. They should be formatted in JSON in the body of the POST request. You can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated.	array	body	No	
`url`	The CI system url for the testrun, this is the link to the CI that enables you to view the current testrun results.	string	body	No	1 to 1024 characters

Parameter	Description	Type	Parameter Type	Required	Restriction
`uuid`	The uuid for the testrun. Used to enable non-*admin* users to update the testrun. If the requested uuid does not match the review an error message is returned.	string	body	No	Minimum 32 max 64 characters

Example usage

Append a new testrun to version 2 of review 12345

```
curl -X POST -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt"
"https://myswarm.url/api/v10/reviews/12345/testruns"
```

The "mybodyfilename.txt" file contains the testrun details:

```
{
  "change": 12345,
  "version": 2,
  "test": "mytest",
  "startTime": 1567895432,
  "status": "running",
  "messages": [
    "Mytest running",
    "another message"
  ]
  "url": "http://my.jenkins.com/projectx/main/1224",
  "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
}
```

Swarm responds with the testrun details:

```
HTTP/1.1 200 OK
```

```
{
  "error": null,
  "messages": [],
  "data" : {
    "testruns" : [
      {
        "id": 706,
        "change": 12345,
        "version": 2,
        "test": "mytest",
        "startTime": 1567895432,
        "completedTime": null,
        "status": "running",
        "messages": [
          "Mytest running",
          "another message"
        ],
        "url": "http://my.jenkins.com/projectx/main/1224",
        "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
      }
    ]
  }
}
```

If the request fails

```
HTTP/1.1 <response error code>

{
  "error": <high level description>,
  "messages": [{"code" : "<code string>", "msg" : "<error message>"}],
  "data" : null
}
```

Create a testrun for a review version using a specified test

Summary

Create a testrun for a review using a specified test

POST /api/v10/reviews/{reviewid}/testruns/{test}

Description

Create a testrun for a review version using a specified test. Tests are configured from the Swarm **Tests** page, see "Tests" on page 440. After you have created a testrun for a version of a review, you can update the testrun details as the test progresses using the PATCH and PUT API endpoints or the `{pass}`, `{fail}` and `{update}` callback urls.

Important
Only *admin* users can create a test run for a review.

Note
The `test` parameter must match a test name in Swarm, if it doesn't match an error is generated when the API request is made.

Parameters

Parameter	Description	Type	Parameter Type	Required	Restriction
`reviewid`	Review ID.	integer	path	Yes	
`version`	The version of the review the testrun is being run against.	integer	body	Yes	

Parameter	Description	Type	Parameter Type	Required	Restriction
`test`	Specifies the test used for the testrun and checks that it is a valid name. If there is a `test` name in the body of the request, Swarm will ignore it. If the `test` parameter is omitted, Swarm uses the `test` name in the body of the request.	string	path	Yes	1 to 32 characters
`startTime`	The time the testrun started, expressed as an epoch value.	integer	body	Yes	> 0
`completedTime`	The time the testrun completed, expressed as an epoch value.	integer	body	No	> 0
`status`	The status of the testrun, options are: pass, fail, and running.	string	body	No	Valid options are pass, fail, and running

Parameter	Description	Type	Parameter Type	Required	Restriction
`messages`	An array of one or more messages for the testrun. They should be formatted in JSON in the body of the POST request. You can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated.	array	body	No	
`url`	The CI system url for the testrun, this is the link to the CI that enables you to view the current testrun results.	string	body	No	1 to 1024 characters

Parameter	Description	Type	Parameter Type	Required	Restriction
`uuid`	The uuid for the testrun. Used to enable non-*admin* users to update the testrun. If the requested uuid does not match the review an error message is returned.	string	body	No	Minimum 32 max 64 characters

Example usage

Append a testrun for a review using test definition "test01" for version 3 of review 12345

```
curl -X POST -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt"
"https://myswarm.url/api/v10/reviews/12345/testruns/test01"
```

The "mybodyfilename.txt" file contains the testrun details:

```
{
  "change": 12345,
  "version": 3,
  "startTime": 1567895432,
  "status": "running",
  "messages": [
    "Test01 running",
    "another message"
  ],
  "url": "http://my.jenkins.com/projecty/main/1224",
  "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC4F"
}
```

Swarm responds with the testrun details:

```
HTTP/1.1 200 OK

{
```

```
  "error": null,
  "messages": [],
  "data" : {
    "testruns" : [
      {
        "id": 707,
        "change": 12345,
        "version": 3,
        "test": "test01",
        "startTime": 1567895432,
        "completedTime": null,
        "status": "running",
        "messages": [
          "Test01 running",
          "another message"
        ],
        "url": "http://my.jenkins.com/projecty/main/1224",
        "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC4F"
      }
    ]
  }
}
```

If the request fails

```
HTTP/1.1 <response error code>

{
  "error": <high level description>,
  "messages": [{"code" : "<code string>", "msg" : "<error message>"}],
  "data" : null
}
```

Update details for a testrun - PATCH

Summary

Update details for a testrun

PATCH /api/v10/reviews/{reviewid}/testruns/{id}

Description

Update the details for a testrun.

> **Important**
> Only *admin* users can update testrun details for a review.

Parameters

Parameter	Description	Type	Parameter Type	Required	Restriction
`Id`	Testrun ID, identifies a specific testrun for the review. Automatically generated by Swarm when the testrun is created.	integer	path	Yes	
`reviewid`	Review ID.	integer	path	Yes	
`version`	The version of the review the testrun is being run against.	integer	body	No	
`test`	Specifies the test used for the testrun and checks that it is a valid name.	string	body	No	1 to 32 characters

Parameter	Description	Type	Parameter Type	Required	Restriction
`startTime`	The time the testrun started, expressed as an epoch value.	integer	body	No	> 0
`completedTime`	The time the testrun completed, expressed as an epoch value.	integer	body	No	> 0
`status`	The status of the testrun, options are: pass, fail, and running	string	body	No	Valid options are pass, fail, and running

Parameter	Description	Type	Parameter Type	Required	Restriction
`messages`	An array of one or more messages for the testrun. They should be formatted in JSON in the body of the POST request. You can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated.	array	body	No	
`url`	The CI system url for the testrun, this is the link to the CI that enables you to view the current testrun results.	string	body	No	1 to 1024 characters

Parameter	Description	Type	Parameter Type	Required	Restriction
`uuid`	The uuid for the testrun. Used to enable non-*admin* users to update the testrun. If the requested uuid does not match the review an error message is returned.	string	body	No	Minimum 32 max 64 characters

Example usage

Update testrun 706 for review 12345

```
curl -X PATCH -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt"
"https://myswarm.url/api/v10/reviews/12345/testruns/706"
```

The "mybodyfilename.txt" file contains the testrun details you want to update.

```
{
  "completedTime": "1567895562",
  "status": "fail",
  "messages": [
    "Test has failed",
    "yet another message"
  ]
}
```

Swarm responds with the testrun details:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "testruns" : [
```

```
      {
        "id": 706,
        "change": 12345,
        "version": 2,
        "test": "mytest",
        "startTime": 1567895432,
        "completedTime": 1567895562,
        "status": "fail",
        "messages": [
          "Test has failed",
          "yet another message"
        ],
        "url": "http://my.jenkins.com/projectx/main/1224",
        "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
      }
    ]
  }
}
```

If the request fails

```
HTTP/1.1 <response error code>

{
  "error": <high level description>,
  "messages": [{"code" : "<code string>", "msg" : "<error message>"}],
  "data" : null
}
```

Update details for a testrun - PUT

Summary

Update details for a testrun

PUT /api/v10/reviews/{reviewid}/testruns/{id}

Description

Update the details for a testrun. All values must be provided in the request.

> **Important**
> Only *admin* users can update testrun details for a review.

Parameters

Parameter	Description	Type	Parameter Type	Required	Restriction
`Id`	Testrun ID, identifies a specific testrun for the review. Automatically generated by Swarm when the testrun is created.	integer	path	Yes	
`reviewid`	Review ID.	integer	path	Yes	
`version`	The version of the review the testrun is being run against.	integer	body	Yes	
`test`	Specifies the test used for the testrun and checks that it is a valid name.	string	body	Yes	1 to 32 characters
`startTime`	The time the testrun started, expressed as an epoch value.	integer	body	Yes	> 0
`completedTime`	The time the testrun completed, expressed as an epoch value.	integer	body	Yes	> 0

Parameter	Description	Type	Parameter Type	Required	Restriction
`status`	The status of the testrun, options are: pass, fail, and running	string	body	Yes	Valid options are pass, fail, and running
`messages`	An array of one or more messages for the testrun. They should be formatted in JSON in the body of the POST request. You can pass a maximum 10 messages, if you provide more than 10 messages only the first 10 are saved. Each message can contain a maximum of 80 characters, any messages with more than 80 characters will be automatically truncated.	array	body	Yes	

Parameter	Description	Type	Parameter Type	Required	Restriction
`url`	The CI system url for the testrun, this is the link to the CI that enables you to view the current testrun results.	string	body	Yes	1 to 1024 characters
`uuid`	The uuid for the testrun. Used to enable non-*admin* users to update the testrun. If the requested uuid does not match the review an error message is returned.	string	body	Yes	Minimum 32 max 64 characters

Example usage

Update testrun 706 for review 12345

```
curl -X PUT -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt"
"https://myswarm.url/api/v10/reviews/12345/testruns/706"
```

The "mybodyfilename.txt" file must contain all of the testrun details:

```
{
  "id": 706,
  "change": 12345,
  "version": 2,
  "test": "mytest",
  "startTime": 1567895432,
  "completedTime": 1567895562,
  "status": "fail",
  "messages": [
    "Test has failed",
```

```
    "and another message"
  ],
  "url": "http://my.jenkins.com/projectx/main/1224",
  "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
}
```

Swarm responds with the testrun details:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data" : {
    "testruns" : [
      {
        "id": 706,
        "change": 12345,
        "version": 2,
        "test": "mytest",
        "startTime": 1567895432,
        "completedTime": 1567895562,
        "status": "fail",
        "messages": [
          "Test has failed",
          "and another message"
        ],
        "url": "http://my.jenkins.com/projectx/main/1224",
        "uuid": "FAE4501C-E4BC-73E4-A11A-FF710601BC3F"
      }
    ]
  }
}
```

If the request fails

```
HTTP/1.1 <response error code>
```

```
{
  "error": <high level description>,
  "messages": [{"code" : "<code string>", "msg" : "<error message>"}],
  "data" : null
}
```

Workflows : Swarm workflows

Get workflows

Summary

Gets workflows

GET /api/v10/workflows/

Description

Get all workflows

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each workflow. Omitting this parameter or passing an empty value shows all fields.	string	query	No
`noCache`	If provided and has a value of 'true' a query will always be performed and the cache of workflows is ignored. Otherwise the cache will be used if it exists.	boolean	query	No

Usage example

Get a list of workflows

```
curl -u "username:password" "https://myswarm-url/api/v10/workflows"
```

JSON Response:

> **Tip**
> The global workflow has an `id` of `0`.

```
HTTP/1.1 200 OK


{
  "error": null,
  "messages": [],
  "data": {
    "workflows": [
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
                    "mode": "default"
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "policy"
                }
            },
            "name": "Global Workflow",
            "description": "This is the Global workflow",
            "shared": "false",
            "owners": [
                "swarm"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_checking",
                    "mode": "default"
                }
            },
            "auto_approve":{
                "rule": "never",
                "mode": "default"
```

```
            },
            "counted_votes":{
                "rule": "anyone",
                "mode": "default"
            },
            "group_exclusions":{
                "rule": [ ],
                "mode": "policy"
            },
            "user_exclusions":{
                "rule": [ ],
                "mode": "policy"
            },
            "tests": [],
            "id": "0"
        },
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                }
            },
            "name": "myWorkflow 1",
            "description": "Another description",
            "shared": "true",
            "owners": [
                "user3",
                "user4"
            ],
            "end_rules":{
```

```
                "update":{
                    "rule":"no_revision",
                    "mode": "inherit"
                }
            },
            "auto_approve":{
                "rule": "votes",
                "mode": "inherit"
            },
            "counted_votes": {
                "rule": "members",
                "mode": "inherit"
            },
            "tests": [],
            "id": "1"
        },
    ]
}
```

If a request fails

<error code>:

501 Workflows are not enabled. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Get a workflow by id

Summary

Gets a workflow by id

GET /api/v10/workflows/{id}

Description

Gets a workflow by id. The global workflow `id` is `0`.

Parameters

Parameter	Description	Type	Parameter Type	Required
`fields`	An optional comma-separated list (or array) of fields to show for each workflow. Omitting this parameter or passing an empty value shows all fields.	string	query	No

Example usage

Get a workflows by id

```
curl -u "username:password" "https://my-swarm-host/api/v10/workflows/1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "workflows": [
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
```

```
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                }
            },
            "name": "myWorkflow 1",
            "description": "Another description",
            "shared": "true",
            "owners": [
                "user3",
                "user4"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_revision",
                    "mode": "inherit"
                }
            },
            "auto_approve":{
                "rule": "votes",
                "mode": "inherit"
            },
            "counted_votes": {
                "rule": "members",
                "mode": "inherit"
            },
            "tests": [],
            "id": "1"
        },
    ]
}
```

If a request fails

<error code>:

- **401** Insufficient permissions to view the workflow
- **404** Workflow does not exist
- **501** Workflows are not enabled. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Get workflow details by workflow name or test id

Summary

Gets workflow details by workflow name or for workflows associated with a test id

GET /api/v10/workflows

Description

Gets workflow details by workflow name or for workflows associated with a test id. You can enter multiple workflow names or test ids in the request if required.

> **Tip**
> If the **name** and **testdefinitions** parameters are both in the request, only **name** is used for the request and the **testdefinitions** parameter is ignored.

Parameters

Parameter	Description	Type	Parameter Type	Required
`name`	Specify the workflow you want to return details for with the workflow `name` parameter. To request details for multiple workflows, enter the workflow names as a comma separated list.	string	query	No
`testdefinitions`	Get details of workflows associated with a test id with the `testdefinitions` parameter. To request multiple test ids, enter the test ids as a comma separated list.	string	query	No

Example usage

Get details for myWorkflow 1

```
curl -u "username:password" "https://my-swarm-host/api/v10/workflows?name=myWorkflow 1"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "workflows": [
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
```

```
                    "mode": "inherit"
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                }
            },
            "name": "myWorkflow 1",
            "description": "Another description",
            "shared": "true",
            "owners": [
                "user3",
                "user4"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_revision",
                    "mode": "inherit"
                }
            },
            "auto_approve":{
                "rule": "votes",
                "mode": "inherit"
            },
            "counted_votes": {
                "rule": "members",
                "mode": "inherit"
            },
            "tests": [],
            "id": "1"
        },
    ]
}
```

Get details for workflows associated with test id 1 or test id 2

```
curl -u "username:password" "https://my-swarm-host/api/v10/workflows?testdefinitions=1,2"
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "workflows": [
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
                    "mode": "default"
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "policy"
                }
            },
            "name": "Our Workflow",
            "description": "This is the our workflow and it is good",
            "shared": "true",
            "owners": [
                "swarm"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_checking",
                    "mode": "default"
                }
            },
            "auto_approve":{
                "rule": "never",
```

```
            "mode": "default"
        },
        "counted_votes":{
            "rule": "anyone",
            "mode": "default"
        },
        "group_exclusions":{
            "rule": [ ],
            "mode": "policy"
        },
        "user_exclusions":{
            "rule": [ ],
            "mode": "policy"
        },
        "tests": [
            {
                 "id": 1,
                 "event": "onUpdate"
            },
            {
                 "id": 2,
                 "event": "onSubmit"
            }
        ],
        "id": "3"
    },
    {
        "on_submit":{
            "with_review":{
                "rule": "no_checking",
                "mode": "inherit"
            },
            "without_review":{
                "rule": "no_checking",
                "mode": "inherit"
```

```
            }
        },
        "name": "Another Workflow",
        "description": "Yet another workflow description",
        "shared": "false",
        "owners": [
            "user3",
            "user4"
        ],
        "end_rules":{
            "update":{
                "rule":"no_revision",
                "mode": "inherit"
            }
        },
        "auto_approve":{
            "rule": "votes",
            "mode": "inherit"
        },
        "counted_votes": {
            "rule": "members",
            "mode": "inherit"
        },
        "tests": [
            {
                 "id": 2,
                 "event": "onSubmit"
            },
            {
                 "id": 5,
                 "event": "onSubmit"
            }
        ],
        "id": "6"
    },
```

```
    ]
}
```

If a request fails

<error code>:

- **401** Insufficient permissions to view the workflow
- **404** Workflow or id does not exist
- **501** Workflows are not enabled. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Create a workflow

Summary

Create a workflow

POST /api/v10/workflows/

Description

Create a new workflow.

> **Tip**
> The global workflow is present by default so it can only be edited.

Parameters

Parameter	Description	Type	Parameter Type	Required
`name`	The workflow name. Will be compared against other workflows and rejected if not unique.	string	body	Yes
`description`	Description for the new workflow.	string	body	Yes
`shared`	Whether this workflow is shared for other users that do not own it. Defaults to not shared.	boolean	body	Yes
`owners`	A list owners for the workflow. Can be users or group names (prefixed with swarm-group-). Users and group names must exist or the workflow will be rejected.	array (of strings)	body	Yes
`mode`	Each workflow `rule` must have a `mode` parameter. The only valid value for `mode` is `inherit`.	string	body	Yes
`on_submit`	Data for rules when changes are submitted. Valid values for `with_review` are `no_checking`, `approved`, `strict`. Valid values for `without_review` are `no_checking`, `auto_create`, `reject`.	array	body	Yes
`end_rules`	Data for rules when changes are submitted. Valid values are `no_checking`, `no_revision`.	array	body	Yes
`auto_approve`	Data for rules when changes are submitted. Valid values are `votes`, `never`.	array	body	Yes
`counted_votes`	Data for rules when counting votes up. Valid values are `anyone`, `members`.	array	body	Yes
`tests`	A list of test `id`s for the workflow. Each test `id` has an `event` that determines when the test is run. Valid values for `event` are `onUpdate`, `onSubmit`.	array	body	No

Example usage

Create a workflow

```
curl -X POST -H "Content-Type: application/json" -u
"username:ticket" -d "@mybodyfilename.txt" "https://myswarm-
url/api/v10/workflows/"
```

The "mybodyfilename.txt" file contains the information for the new workflow:

```
{
  "name": "My workflow"
  "description": "This is my workflow."
  "shared": "true"
  "owners": "Francois_Piccard,Anna_Schmidt"
  "on_submit":{
      "with_review":{
          "rule":"no_checking",
          "mode": "inherit"
      },
      "without_review":{
          "rule":"no_checking",
          "mode": "inherit"
      }
  },
  "end_rules": {
      "update":{
          "rule":"no_revision",
          "mode": "inherit"
      }
  },
  "auto_approve":{
      "rule": "never",
      "mode": "inherit"
  },
  "counted_votes": {
      "rule": "members",
      "mode": "inherit"
  },
```

```
  "tests": [
      {
           "id": 2,
           "event": "onUpdate"
      },
      {
           "id": 5,
           "event": "onSubmit"
      }
  ]
}
```

Swarm responds with:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "workflows": [
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                }
            },
            "name": "My workflow",
            "description": "This is my workflow.",
            "shared": "true",
            "owners": [
                "Francois_Piccard",
```

```
                "Anna_Schmidt"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_revision",
                    "mode": "inherit"
                }
            },
            "auto_approve":{
                "rule": "votes",
                "mode": "inherit"
            },
            "counted_votes": {
                "rule": "members",
                "mode": "inherit"
            },
            "tests": [
                {
                     "id": 2,
                     "event": "onUpdate"
                },
                {
                     "id": 5,
                     "event": "onSubmit"
                }
            ],
            "id": "2"
        },
    ]
}
```


If a request fails

<error code>:

- **400** Invalid parameter data specified
- **401** Insufficient permissions to create a workflow
- **501** Workflows are not enabled. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Update a workflow

Summary

Update a workflow

PUT /api/v10/workflows/{id}

Description

Update a workflow. All values should be provided in the request. If not provided any missing values are reverted to default.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	The id of the workflow being updated	integer	path	Yes
`name`	The workflow name. Will be compared against other workflows and rejected if not unique	string	body	Yes

Parameter	Description	Type	Parameter Type	Required
`owners`	A list owners for the workflow. Can be users or group names (prefixed with swarm-group-). Users and group names must exist or the workflow will be rejected	array (of strings)	body	Yes
`description`	Description for the new workflow	string	body	Yes
`shared`	Whether this workflow is shared for other users that do not own it. Defaults to not shared	boolean	body	Yes
`mode`	**Workflow rules:** Each workflow `rule` must have a `mode` parameter. The only valid value for `mode` is `inherit`. **Global workflow rules only:** ■ `default` applies the workflow rule setting to projects and project branches that don't have an associated workflow. If a project or project branch has an associated Swarm workflow, the global rule is ignored. ■ `policy` applies a minimum workflow rule setting to all projects and project branches. If a project or project branch has an associated workflow, the global rule is merged with the workflow rule and the most restrictive setting is used. Displayed as **Enforce on** in the Swarm UI, see "Global workflow" on page 605.	string	body	Yes

Parameter	Description	Type	Parameter Type	Required
`on_submit`	Data for rules when changes are submitted. Valid values for `with_review` are `no_checking`, `approved`, `strict`. Valid values for `without_review` are `no_checking`, `auto_create`, `reject`	array	body	Yes
`end_rules`	Data for rules when changes are submitted. Valid values are `no_checking`, `no_revision`.	array	body	Yes
`auto_approve`	Data for rules when changes are submitted. Valid values are `votes`, `never`.	array	body	Yes
`counted_votes`	Data for rules when counting votes up. Valid values are `anyone`, `members`.	array	body	Yes
`tests`	A list of test `id`s for the workflow. Each test `id` has an `event` that determines when the test is run. Valid values for `event` are `onUpdate`, `onSubmit`.	array	body	No

Example usage

Update a workflow

```
curl -X PUT -H "Content-Type: application/json" -u "username:ticket" -d "@mybodyfilename.txt" "https://myswarm-url/api/v10/workflows/2"
```

The "mybodyfilename.txt" file contains the updated workflow information:

```
{
  "name": "My workflow"
  "description": "This is my workflow and I have changed the description of it."
  "shared": "true"
  "owners": "Francois_Piccard,Anna_Schmidt"
  "on_submit":{
```

```
        "with_review":{
            "rule":"no_checking",
            "mode": "inherit"
        },
        "without_review":{
            "rule":"no_checking",
            "mode": "inherit"
        }
    },
    "end_rules": {
        "update":{
            "rule":"no_revision",
            "mode": "inherit"
        }
    },
    "auto_approve":{
        "rule": "never",
        "mode": "inherit"
    },
    "counted_votes": {
        "rule": "members",
        "mode": "inherit"
    },
    "tests": [
        {
             "id": 2,
             "event": "onUpdate"
        },
        {
             "id": 5,
             "event": "onSubmit"
        }
    ]
}
```

JSON Response:

```
HTTP/1.1 200 OK

{
  "error": null,
  "messages": [],
  "data": {
    "workflows": [
        {
            "on_submit":{
                "with_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                },
                "without_review":{
                    "rule": "no_checking",
                    "mode": "inherit"
                }
            },
            "name": "My workflow",
            "description": "This is my workflow and I have changed the
description of it.",
            "shared": "true",
            "owners": [
                "Francois_Piccard",
                "Anna_Schmidt"
            ],
            "end_rules":{
                "update":{
                    "rule":"no_revision",
                    "mode": "inherit"
                }
            },
            "auto_approve":{
                "rule": "votes",
                "mode": "inherit"
```

```
            },
            "counted_votes": {
                "rule": "members",
                "mode": "inherit"
            },
            "tests": [
                {
                     "id": 2,
                     "event": "onUpdate"
                },
                {
                     "id": 5,
                     "event": "onSubmit"
                }
            ],
            "id": "2"
        },
    ]
}
```

If a request fails

<error code>:

- **400** Invalid parameter data specified
- **401** Insufficient permissions to update the workflow
- **404** Workflow does not exist or you do not have permission to update the workflow
- **501** Workflows are not enabled. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
```

```
  "data" : null
}
```

Delete a workflow

Summary

Delete a workflow

DELETE /api/v10/workflows/{id}

Description

Delete a workflow for the provided id. This call must be authenticated and the user must have permission to edit the workflow. If the workflow is in use it cannot be deleted and an error message will be returned. The global workflow cannot be deleted.

Parameters

Parameter	Description	Type	Parameter Type	Required
`id`	The id of the workflow being deleted	string	form	Yes

Example usage

Delete a workflow that is not in use by a project

```
curl -u "username:password" -X DELETE "https://my-swarm-host/api/v10/workflows/1"
```

JSON Response

```
HTTP/1.1 200

{
  "error": null,
  "messages": [
    "Workflow [1] was deleted"
  ],
  "data": null
}
```

If a request fails

<error code>:

- **401** Insufficient permissions to delete the workflow
- **403** Cannot delete the global workflow
- **404** Workflow does not exist or you do not have permission to view this workflow
- **501** Workflows are not enabled. To enable workflows, see "workflow " on page 605

```
HTTP/1.1 <response error code>

{
  "error": <error code>,
  "messages": [{
    "code" : "<code string>",
    "text" : "<error message>"
  }],
  "data" : null
}
```

Glossary

A

access level

A permission assigned to a user to control which commands the user can execute. See also the 'protections' entry in this glossary and the 'p4 protect' command in the P4 Command Reference.

admin access

An access level that gives the user permission to privileged commands, usually super privileges.

APC

The Alternative PHP Cache, a free, open, and robust framework for caching and optimizing PHP intermediate code.

archive

1. For replication, versioned files (as opposed to database metadata). 2. For the 'p4 archive' command, a special depot in which to copy the server data (versioned files and metadata).

atomic change transaction

Grouping operations affecting a number of files in a single transaction. If all operations in the transaction succeed, all the files are updated. If any operation in the transaction fails, none of the files are updated.

avatar

A visual representation of a Swarm user or group. Avatars are used in Swarm to show involvement in or ownership of projects, groups, changelists, reviews, comments, etc. See also the "Gravatar" entry in this glossary.

B

base

For files: The file revision, in conjunction with the source revision, used to help determine what integration changes should be applied to the target revision. For checked out streams: The public have version from which the checked out version is derived.

binary file type

A Helix server file type assigned to a non-text file. By default, the contents of each revision are stored in full, and file revision is stored in compressed format.

branch

(noun) A set of related files that exist at a specific location in the Perforce depot as a result of being copied to that location, as opposed to being added to that location. A group of related files is often referred to as a codeline. (verb) To create a codeline by copying another codeline with the 'p4 integrate', 'p4 copy', or 'p4 populate' command.

branch form

The form that appears when you use the 'p4 branch' command to create or modify a branch specification.

branch mapping

Specifies how a branch is to be created or integrated by defining the location, the files, and the exclusions of the original codeline and the target codeline. The branch mapping is used by the integration process to create and update branches.

branch view

A specification of the branching relationship between two codelines in the depot. Each branch view has a unique name and defines how files are mapped from the originating codeline to the target codeline. This is the same as branch mapping.

broker

Helix Broker, a server process that intercepts commands to the Helix server and is able to run scripts on the commands before sending them to the Helix server.

C

change review

The process of sending email to users who have registered their interest in changelists that include specified files in the depot.

changelist

A list of files, their version numbers, the changes made to the files, and a description of the changes made. A changelist is the basic unit of versioned work in Helix server. The changes specified in the changelist are not stored in the depot until the changelist is submitted to the depot. See also atomic change transaction and changelist number.

changelist form

The form that appears when you modify a changelist using the 'p4 change' command.

changelist number

An integer that identifies a changelist. Submitted changelist numbers are ordinal (increasing), but not necessarily consecutive. For example, 103, 105, 108, 109. A pending changelist number might be assigned a different value upon submission.

check in

To submit a file to the Helix server depot.

check out

To designate one or more files, or a stream, for edit.

checkpoint

A backup copy of the underlying metadata at a particular moment in time. A checkpoint can recreate db.user, db.protect, and other db.* files. See also metadata.

classic depot

A repository of Helix server files that is not streams-based. Uses the Perforce file revision model, not the graph model. The default depot name is depot. See also default depot, stream depot, and graph depot.

client form

The form you use to define a client workspace, such as with the 'p4 client' or 'p4 workspace' commands.

client name

A name that uniquely identifies the current client workspace. Client workspaces, labels, and branch specifications cannot share the same name.

client root

The topmost (root) directory of a client workspace. If two or more client workspaces are located on one machine, they should not share a client root directory.

client side

The right-hand side of a mapping within a client view, specifying where the corresponding depot files are located in the client workspace.

client workspace

Directories on your machine where you work on file revisions that are managed by Helix server. By default, this name is set to the name of the machine on which your client workspace is located, but it can be overridden. Client workspaces, labels, and branch specifications cannot share the same name.

code review

A process in Helix Swarm by which other developers can see your code, provide feedback, and approve or reject your changes.

codeline

A set of files that evolve collectively. One codeline can be branched from another, allowing each set of files to evolve separately.

comment

Feedback provided in Helix Swarm on a changelist, review, job, or a file within a changelist or review.

commit server

A server that is part of an edge/commit system that processes submitted files (checkins), global workspaces, and promoted shelves.

conflict

1. A situation where two users open the same file for edit. One user submits the file, after which the other user cannot submit unless the file is resolved. 2. A resolve where the same line is changed when merging one file into another. This type of conflict occurs when the comparison of two files to a base yields different results, indicating that the files have been changed in different ways. In this case, the merge cannot be done automatically and must be resolved manually. See file conflict.

copy up

A Helix server best practice to copy (and not merge) changes from less stable lines to more stable lines. See also merge.

counter

A numeric variable used to track variables such as changelists, checkpoints, and reviews.

CSRF

Cross-Site Request Forgery, a form of web-based attack that exploits the trust that a site has in a user's web browser.

D

default changelist

The changelist used by a file add, edit, or delete, unless a numbered changelist is specified. A default pending changelist is created automatically when a file is opened for edit.

deleted file

In Helix server, a file with its head revision marked as deleted. Older revisions of the file are still available. in Helix server, a deleted file is simply another revision of the file.

delta

The differences between two files.

depot

A file repository hosted on the server. A depot is the top-level unit of storage for versioned files (depot files or source files) within a Helix Core server. It contains all versions of all files ever submitted to the depot. There can be multiple depots on a single installation.

depot root

The topmost (root) directory for a depot.

depot side

The left side of any client view mapping, specifying the location of files in a depot.

depot syntax

Helix server syntax for specifying the location of files in the depot. Depot syntax begins with: //depot/

diff

(noun) A set of lines that do not match when two files, or stream versions, are compared. A conflict is a pair of unequal diffs between each of two files and a base, or between two versions of a stream. (verb) To compare the contents of files or file revisions, or of stream versions. See also conflict.

donor file

The file from which changes are taken when propagating changes from one file to another.

E

edge server

A replica server that is part of an edge/commit system that is able to process most read/write commands, including 'p4 integrate', and also deliver versioned files (depot files).

exclusionary access

A permission that denies access to the specified files.

exclusionary mapping

A view mapping that excludes specific files or directories.

extension

Similar to a trigger, but more modern. See "Helix Core Server Administrator Guide" on "Extensions".

F

file conflict

In a three-way file merge, a situation in which two revisions of a file differ from each other and from their base file. Also, an attempt to submit a file that is not an edit of the head revision of the file in the depot, which typically occurs when another user opens the file for edit after you have opened the file for edit.

file pattern

Helix server command line syntax that enables you to specify files using wildcards.

file repository

The master copy of all files, which is shared by all users. In Helix server, this is called the depot.

file revision

A specific version of a file within the depot. Each revision is assigned a number, in sequence. Any revision can be accessed in the depot by its revision number, preceded by a pound sign (#), for example testfile#3.

file tree

All the subdirectories and files under a given root directory.

file type

An attribute that determines how Helix server stores and diffs a particular file. Examples of file types are text and binary.

fix

A job that has been closed in a changelist.

form

A screen displayed by certain Helix server commands. For example, you use the change form to enter comments about a particular changelist to verify the affected files.

forwarding replica

A replica server that can process read-only commands and deliver versioned files (depot files). One or more replicate servers can significantly improve performance by offloading some of the master server load. In many cases, a forwarding replica can become a disaster recovery server.

G

Git Fusion

A Perforce product that integrates Git with Helix, offering enterprise-ready Git repository management, and workflows that allow Git and Helix server users to collaborate on the same projects using their preferred tools.

graph depot

A depot of type graph that is used to store Git repos in the Helix server. See also Helix4Git and classic depot.

group

A feature in Helix server that makes it easier to manage permissions for multiple users.

H

have list

The list of file revisions currently in the client workspace.

head revision

The most recent revision of a file within the depot. Because file revisions are numbered sequentially, this revision is the highest-numbered revision of that file.

heartbeat

A process that allows one server to monitor another server, such as a standby server monitoring the master server (see the p4 heartbeat command).

Helix server

The Helix server depot and metadata; also, the program that manages the depot and metadata, also called Helix Core server.

Helix TeamHub

A Perforce management platform for code and artifact repository. TeamHub offers built-in support for Git, SVN, Mercurial, Maven, and more.

Helix4Git

Perforce solution for teams using Git. Helix4Git offers both speed and scalability and supports hybrid environments consisting of Git repositories and 'classic' Helix server depots.

hybrid workspace

A workspace that maps to files stored in a depot of the classic Perforce file revision model as well as to files stored in a repo of the graph model associated with git.

I

iconv

A PHP extension that performs character set conversion, and is an interface to the GNU libiconv library.

integrate

To compare two sets of files (for example, two codeline branches) and determine which changes in one set apply to the other, determine if the changes have already been propagated, and propagate any outstanding changes from one set to another.

J

job

A user-defined unit of work tracked by Helix server. The job template determines what information is tracked. The template can be modified by the Helix server system administrator. A job describes work to be done, such as a bug fix. Associating a job with a changelist records which changes fixed the bug.

job daemon

A program that checks the Helix server machine daily to determine if any jobs are open. If so, the daemon sends an email message to interested users, informing them the number of jobs in each category, the severity of each job, and more.

job specification

A form describing the fields and possible values for each job stored in the Helix server machine.

job view

A syntax used for searching Helix server jobs.

journal

A file containing a record of every change made to the Helix server's metadata since the time of the last checkpoint. This file grows as each Helix server transaction is logged. The file should be automatically truncated and renamed into a numbered journal when a checkpoint is taken.

journal rotation

The process of renaming the current journal to a numbered journal file.

journaling

The process of recording changes made to the Helix server's metadata.

L

label

A named list of user-specified file revisions.

label view

The view that specifies which filenames in the depot can be stored in a particular label.

lazy copy

A method used by Helix server to make internal copies of files without duplicating file content in the depot. A lazy copy points to the original versioned file (depot file). Lazy copies minimize the consumption of disk space by storing references to the original file instead of copies of the file.

license file

A file that ensures that the number of Helix server users on your site does not exceed the number for which you have paid.

list access

A protection level that enables you to run reporting commands but prevents access to the contents of files.

local depot

Any depot located on the currently specified Helix server.

local syntax

The syntax for specifying a filename that is specific to an operating system.

lock

1. A file lock that prevents other clients from submitting the locked file. Files are unlocked with the 'p4 unlock' command or by submitting the changelist that contains the locked file. 2. A database lock that prevents another process from modifying the database db.* file.

log

Error output from the Helix server. To specify a log file, set the P4LOG environment variable or use the p4d -L flag when starting the service.

M

mapping

A single line in a view, consisting of a left side and a right side that specify the correspondences between files in the depot and files in a client, label, or branch. See also workspace view, branch view, and label view.

MDS checksum

The method used by Helix server to verify the integrity of versioned files (depot files).

merge

1. To create new files from existing files, preserving their ancestry (branching). 2. To propagate changes from one set of files to another. 3. The process of combining the contents of two conflicting file revisions into a single file, typically using a merge tool like P4Merge.

merge file

A file generated by the Helix server from two conflicting file revisions.

metadata

The data stored by the Helix server that describes the files in the depot, the current state of client workspaces, protections, users, labels, and branches. Metadata is stored in the Perforce database and is separate from the archive files that users submit.

modification time or modtime

The time a file was last changed.

MPM

Multi-Processing Module, a component of the Apache web server that is responsible for binding to network ports, accepting requests, and dispatch operations to handle the request.

N

nonexistent revision

A completely empty revision of any file. Syncing to a nonexistent revision of a file removes it from your workspace. An empty file revision created by deleting a file and the #none revision specifier are examples of nonexistent file revisions.

numbered changelist

A pending changelist to which Helix server has assigned a number.

O

opened file

A file you have checked out in your client workspace as a result of a Helix Core server operation (such as an edit, add, delete, integrate). Opening a file from your operating system file browser is not tracked by Helix Core server.

owner

The Helix server user who created a particular client, branch, or label.

P

p4

1. The Helix Core server command line program. 2. The command you issue to execute commands from the operating system command line.

p4d

The program that runs the Helix server; p4d manages depot files and metadata.

P4PHP

The PHP interface to the Helix API, which enables you to write PHP code that interacts with a Helix server machine.

PECL

PHP Extension Community Library, a library of extensions that can be added to PHP to improve and extend its functionality.

pending changelist

A changelist that has not been submitted.

Perforce

Perforce Software, Inc., a leading provider of enterprise-scale software solutions to technology developers and development operations (“DevOps”) teams requiring productivity, visibility, and scale during all phases of the development lifecycle.

project

In Helix Swarm, a group of Helix server users who are working together on a specific codebase, defined by one or more branches of code, along with options for a job filter, automated test integration, and automated deployment.

protections

The permissions stored in the Helix server’s protections table.

proxy server

A Helix server that stores versioned files. A proxy server does not perform any commands. It serves versioned files to Helix server clients.

R

RCS format

Revision Control System format. Used for storing revisions of text files in versioned files (depot files). RCS format uses reverse delta encoding for file storage. Helix server uses RCS format to store text files. See also reverse delta storage.

read access

A protection level that enables you to read the contents of files managed by Helix server but not make any changes.

remote depot

A depot located on another Helix server accessed by the current Helix server.

replica

A Helix server that contains a full or partial copy of metadata from a master Helix server. Replica servers are typically updated every second to stay synchronized with the master server.

repo

A graph depot contains one or more repos, and each repo contains files from Git users.

reresolve

The process of resolving a file after the file is resolved and before it is submitted.

resolve

The process you use to manage the differences between two revisions of a file, or two versions of a stream. You can choose to resolve file conflicts by selecting the source or target file to be submitted, by merging the contents of conflicting files, or by making additional changes. To resolve stream conflicts, you can choose to accept the public source, accept the checked out target, manually accept changes, or combine path fields of the public and checked out version while accepting all other changes made in the checked out version.

reverse delta storage

The method that Helix server uses to store revisions of text files. Helix server stores the changes between each revision and its previous revision, plus the full text of the head revision.

revert

To discard the changes you have made to a file in the client workspace before a submit.

review access

A special protections level that includes read and list accesses and grants permission to run the p4 review command.

review daemon

A program that periodically checks the Helix server machine to determine if any changelists have been submitted. If so, the daemon sends an email message to users who have subscribed to any of the files included in those changelists, informing them of changes in files they are interested in.

revision number

A number indicating which revision of the file is being referred to, typically designated with a pound sign (#).

revision range

A range of revision numbers for a specified file, specified as the low and high end of the range. For example, myfile#5,7 specifies revisions 5 through 7 of myfile.

revision specification

A suffix to a filename that specifies a particular revision of that file. Revision specifiers can be revision numbers, a revision range, change numbers, label names, date/time specifications, or client names.

RPM

RPM Package Manager. A tool, and package format, for managing the installation, updates, and removal of software packages for Linux distributions such as Red Hat Enterprise Linux, the Fedora Project, and the CentOS Project.

S

server data

The combination of server metadata (the Helix server database) and the depot files (your organization's versioned source code and binary assets).

server root

The topmost directory in which p4d stores its metadata (db.* files) and all versioned files (depot files or source files). To specify the server root, set the P4ROOT environment variable or use the p4d -r flag.

service

In the Helix Core server, the shared versioning service that responds to requests from Helix server client applications. The Helix server (p4d) maintains depot files and metadata describing the files and also tracks the state of client workspaces.

shelve

The process of temporarily storing files in the Helix server without checking in a changelist.

status

For a changelist, a value that indicates whether the changelist is new, pending, or submitted. For a job, a value that indicates whether the job is open, closed, or suspended. You can customize job statuses. For the 'p4 status' command, by default the files opened and the files that need to be reconciled.

storage record

An entry within the db.storage table to track references to an archive file.

stream

A "branch" with built-in rules that determines what changes should be propagated and in what order they should be propagated.

stream depot

A depot used with streams and stream clients. Has structured branching, unlike the free-form branching of a "classic" depot. Uses the Perforce file revision model, not the graph model. See also classic depot and graph depot.

submit

To send a pending changelist into the Helix server depot for processing.

super access

An access level that gives the user permission to run every Helix server command, including commands that set protections, install triggers, or shut down the service for maintenance.

symlink file type

A Helix server file type assigned to symbolic links. On platforms that do not support symbolic links, symlink files appear as small text files.

sync

To copy a file revision (or set of file revisions) from the Helix server depot to a client workspace.

T

target file

The file that receives the changes from the donor file when you integrate changes between two codelines.

text file type

Helix server file type assigned to a file that contains only ASCII text, including Unicode text. See also binary file type.

theirs

The revision in the depot with which the client file (your file) is merged when you resolve a file conflict. When you are working with branched files, theirs is the donor file.

three-way merge

The process of combining three file revisions. During a three-way merge, you can identify where conflicting changes have occurred and specify how you want to resolve the conflicts.

trigger

A script that is automatically invoked by Helix server when various conditions are met. (See "Helix Core Server Administrator Guide" on "Triggers".)

two-way merge

The process of combining two file revisions. In a two-way merge, you can see differences between the files.

typemap

A table in Helix server in which you assign file types to files.

U

user

The identifier that Helix server uses to determine who is performing an operation. The three types of users are standard, service, and operator.

V

versioned file

Source files stored in the Helix server depot, including one or more revisions. Also known as an archive file. Versioned files typically use the naming convention 'filenamev' or '1.changelist.gz'.

view

A description of the relationship between two sets of files. See workspace view, label view, branch view.

W

wildcard

A special character used to match other characters in strings. The following wildcards are available in Helix server: * matches anything except a slash; ... matches anything including slashes; %%0 through %%9 is used for parameter substitution in views.

workspace

See client workspace.

workspace view

A set of mappings that specifies the correspondence between file locations in the depot and the client workspace.

write access

A protection level that enables you to run commands that alter the contents of files in the depot. Write access includes read and list accesses.

X

XSS

Cross-Site Scripting, a form of web-based attack that injects malicious code into a user's web browser.

Y

yours

The edited version of a file in your client workspace when you resolve a file. Also, the target file when you integrate a branched file.

Getting help

> **Tip**
> Perforce Support cannot help you with user account creation or with resetting your password, please contact your company's Perforce administrator for help with this.

We look forward to hearing about your experiences with Swarm, positive or negative, including *must-haves* or *it would be great if Swarm...*. Please feel free to contact us.

Please visit the Perforce Support portal for product support and contact information:

- Search our knowledge base
- Post on the Perforce forum
- Submit a Support request
- Call us for support
- View our video tutorials
- View our training options
- View the documentation and release notes
- Download Perforce software

Complete contact information is available on the Perforce web site.

License statements

For complete licensing information pertaining to Helix Swarm, see the license file at https://www.perforce.com/perforce/doc.current/user/swarm_license.txt.