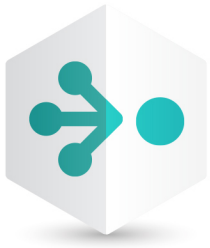


HelixTeamHub

Helix TeamHub User Guide

2020.1 Enterprise

April 2020

PERFORCE

www.perforce.com

Contents

How to Use this Guide

Helix TeamHub is a collection of software development tools for your organization. Helix TeamHub helps you and your team to organize your efforts, stay connected, and store your work.

This guide helps you get started and provides conceptual and procedural information on user profiles and authentication, project and team roles, project and repository hooks, version control and repositories, and bots and programmatic repository access.

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.
[`-f`]	The enclosed elements are 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*.
element1 | *element2*	Either *element1* or *element2* is required.

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.

What's new in this guide for this release

Following is a summary of new information with links to the most prominent topics. For a complete list of new features and bug fixes, see the *Release Notes*.

- New command line tool that enables you to perform cross-repository actions on your Git repositories, see "Helix TeamHub CLI tool - Technology preview feature" on page 144.
- Added user option to set the number of spaces used to display tabs in files, see "Code browser views" on page 73 and "Compare view" on page 77.
- You can now use special characters to make your search results more relevant, see "Advanced searches" on page 119.
- You can now add assignees and the due date when you set up a new issue, see "Add an issue" on page 83
- You can now create a Git branch when you set up a new issue, see "Add an issue" on page 83
- You can now disable file sharing on a per company basis, see "Feature settings" on page 169.
- Added support for Mercurial repository based Wikis, see "Wiki" on page 142.
- Added support for editing the background color and the initials used for the project initials of a project, see "Editing a project" on page 27.
- Improved side menus make it easier to navigate TeamHub and to see where you are in the UI, see "Navigation" on page 12.

1 | Limitations with Helix authentication

TeamHub is integrated with Helix server. If your TeamHub instance is configured to use Helix authentication, you can manage repositories and kick off reviews using the TeamHub user interface.

Helix authentication implies that part of the configuration and management happens in Helix server. As a result, some elements in the TeamHub UI have been removed or disabled and others have been added, as detailed in the following table.

Entity	Change	Helix server Documentation (where applicable)
Collaborators	Removed the **Collaborators** view and ability to view or add collaborators in **Team** view	N/A
Bots	Disabled ability to change a bot's short name	
Repositories	Added ability to: ■ Create Git repositories stored in Helix server. ■ View Helix server connection details (port and path) by selecting the **Clone > Helix P4** option Removed ability to: ■ Fork native Git repositories ■ Configure the garbage collection feature for Git repositories managed by Helix server Git repositories managed by Helix server do not require garbage collection because they do not store data on the disk where the Helix TeamHub instance resides.	
Company settings	■ **Authentication** tab: Disabled ability to configure SAML authentication **Note** With Helix authentication, Helix TeamHub supports only one company per instance. Company creation is disabled.	

Getting started

Welcome aboard! This guide will get you started on using Helix TeamHub.

When you first log in, you land in the **My Dashboard** view. From there, you can quickly access your favorite projects and see open issues or code reviews you are part of.

Navigation

Navigation in Helix TeamHub happens at the following scope levels:

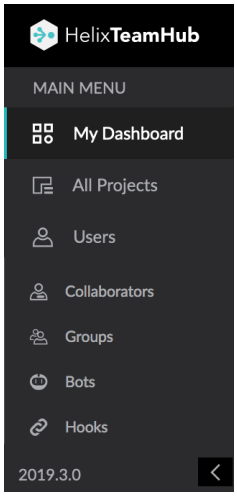
- Account (My Dashboard & User Preferences)
- Company
- Project

> **Tip**
> The links displayed in the side menu depend on the scope level you are viewing.

Company scope

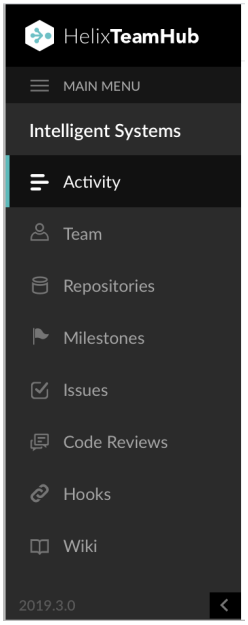
At the company scope level, you can manage your company and its projects, users, collaborators, groups, bots and hooks. You can access the company scope from the header bar by clicking the company name or icon in the top left corner.

> **Note**
> Access to company scope is restricted for collaborators.



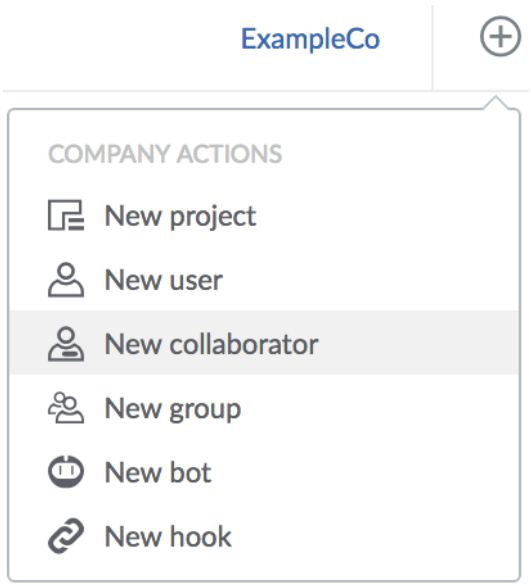
Project scope

The project scope is activated whenever you access or create a new project. In TeamHub, each project is an isolated entity with its own team, repositories, milestones and so on. Projects are created at the company scope level. A company can have multiple projects. The project's team, for example, is formed by granting access to individual company users, collaborators, groups or bots, and giving them appropriate project roles.

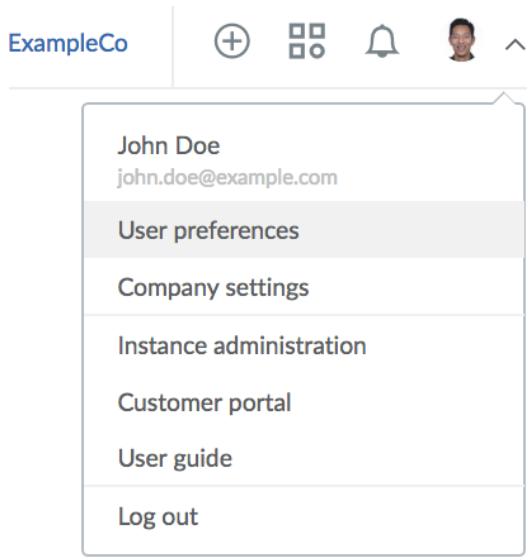


Header

Quick actions in the header enable access to common creative actions in Helix TeamHub.



The avatar dropdown has useful links and lets you access the user preferences.



That's it - we hope you enjoy using TeamHub!

For more information on different user profiles and roles in TeamHub, see:

- User Profiles
- Project & Repository Roles

Repository-type dependent features

TeamHub supports the following combination of features and repository types:

Repository	Git	Helix Git	Mercurial	Subversion	WebDAV	Maven	Ivy	Docker
Edit/Commit from the UI	✓	✓			✓			
Code search	✓	✓	✓					
Code review	✓	✓	✓					
Multi-repo code review		✓						
Compare view	✓	✓	✓	✓				
Branch from UI	✓	✓						
Fork from UI	✓		✓					
Manage protected branches	✓	✓						
"Wiki" on page 142	✓		✓					
Garbage collection	✓							
Change UUID				✓				
Repository Site Replication				✓				

Repository	Git	Helix Git	Mercurial	Subversion	WebDAV	Maven	Ivy	Docker
"Helix TeamHub CLI tool - Technology preview feature" on page 144	✓	✓						

Notation conventions

The Helix TeamHub uses the following notations throughout the UI:

Notation	Description
`#x`	Issue
`!x`	Single code review
`%x`	multi-repo code review

User Profiles & authentication

User profiles

Helix TeamHub supports the following types of users:

- **Company admins**: Have access to all information inside your company but may not have access to a user's private projects

  > **Note**
  > Company admins are different from TeamHub instance administrators. The latter are users with elevated admin privileges and access to the Admin UI.

- **Normal users**: Can access public company data and private projects they have been granted access to
- **Collaborators** (not available with Helix authentication): Can only access projects they have been granted access to

Each person using TeamHub within your company has a TeamHub user profile. The profile includes the user's name, email address, phone number, avatar (image), creation date, last login date, plus additional details about the user. Because TeamHub is keen on storing information on who does what, your public profile is a way for others to see how you contribute to projects and what your expertise is. To edit your profile, click your username at the top right and select **User Preferences**; then click **Edit profile**.

You should always keep your profile up to date, including a photo of yourself, so others can recognize and contact you when needed.

For information on adding users, see "Users" on page 28.

Logging in

All access rights controlled by TeamHub are tied to your account. You can log into TeamHub using any of the following information:

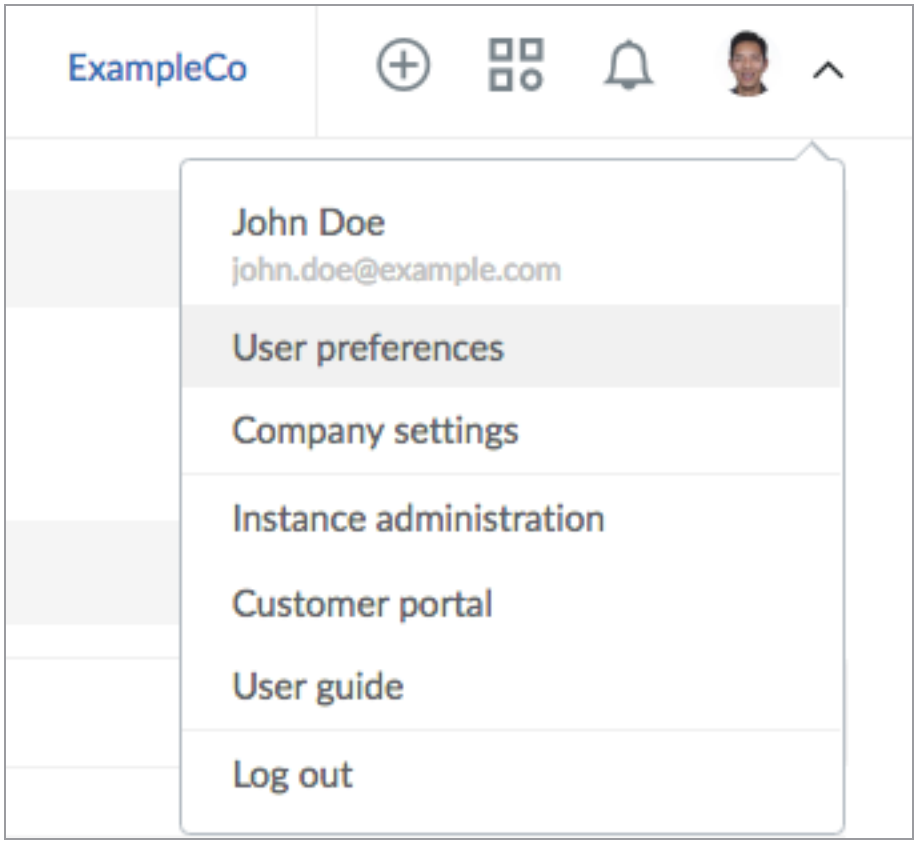
- Company ID, E-mail and password (not available with Helix authentication)
- Company ID, username and password
- SSH key(s) (not available with Helix authentication)

The first two use HTTP. They are used in *helixteamhub.cloud* and other TeamHub services. When using SSH key(s) for repository access, the username is always `hth`. When accessing repositories using SSH protocol, only SSH key(s) can be used for authentication and authorization. When accessing repositories using HTTP protocol, the username can be found either from the clone URL or from user preferences.

When using version control or other developer services, you can also use SSH keys for faster connections and to authenticate without typing your password each time you contact TeamHub. Each device has its own SSH key which you can connect to your Helix TeamHub account in your settings.

Configuring SSH keys

To use SSH keys for repository access, you need to add your public key(s) to TeamHub. SSH keys are managed under user preferences. You can open your user preferences by clicking the user avatar at the top right of the screen.



TeamHub supports ssh-rsa, ssh-ed25519 and ecdsa keys. You can add multiple public keys by naming them differently.

Collaborator accounts

Collaborator accounts allow you to invite external collaborators or contractors, such as freelance designers, researchers, or simply engineering workforce, to work on your projects temporarily. Restrictions defined at the company level determine who can invite external contributors (for details, see "Feature settings" on page 169). If you have permission, you can invite a collaborator by providing their email address. The invited collaborators receive an email with a link for setting their initial password. After selecting a password, TeamHub automatically signs them in.

Collaborator accounts are subject to restrictions. Collaborators can:

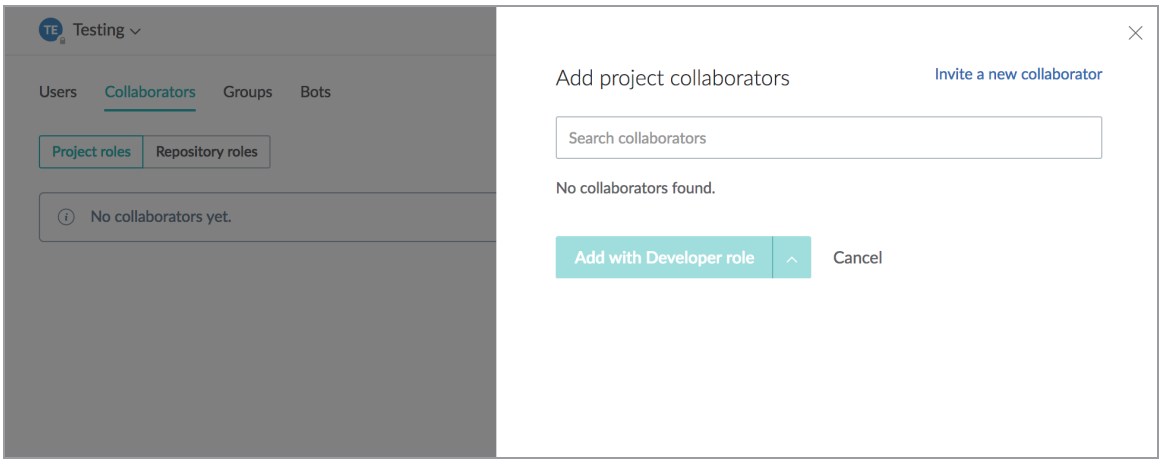
- Only see the projects in which they have a direct role.
- Not see any company scope data. See also Getting started.
- Only hold a `Guest`, `Developer`, or `Master` role in projects.

> **Note**
> With Helix authentication, certain restrictions apply to TeamHub functionality. For details, see "Limitations with Helix authentication" on page 11.

Invite new collaborators

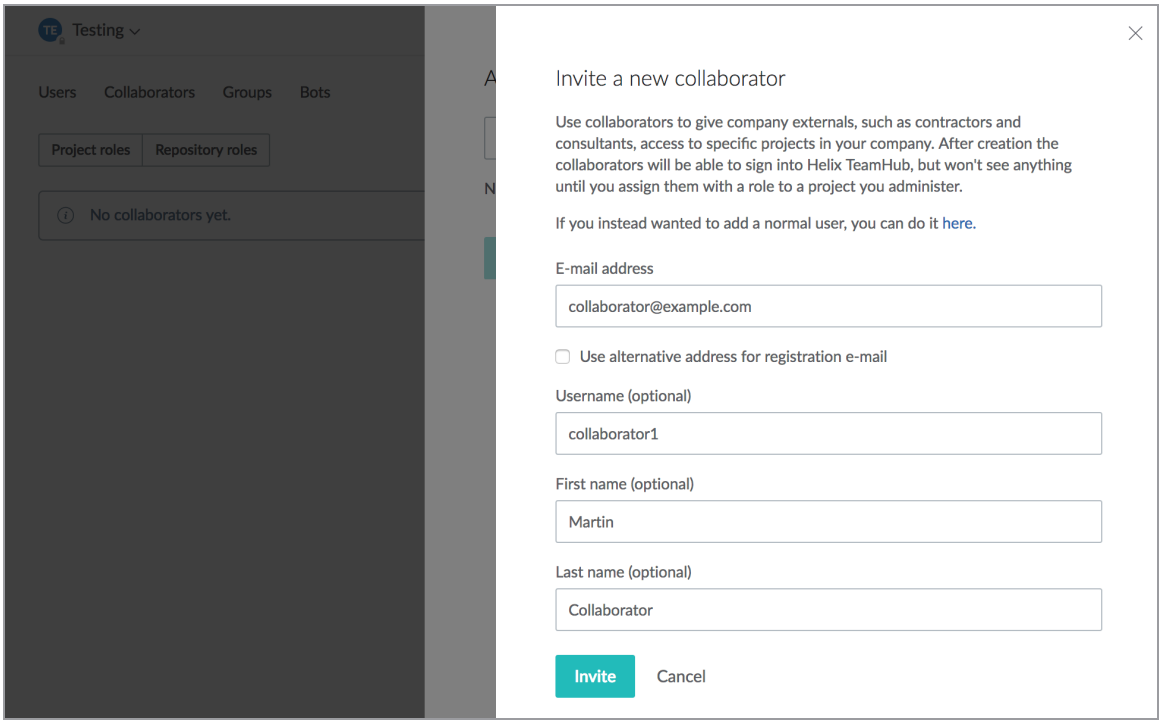
To invite a new collaborator:

1. In the project's **Team** tab, click the plus icon to the right of the search field and select **Add Collaborators**.



2. In the **Add project collaborators** form, click **Invite a new collaborator**.
3. In the **Invite a new collaborator** form, enter an email address. Optionally, specify an alternative

e-mail address for registration purposes and provide information in the **Username**, **First name**, and **Last name** fields.



4. Click **Invite**.

   TeamHub adds the user name (if specified) or email address to the list of collaborators.

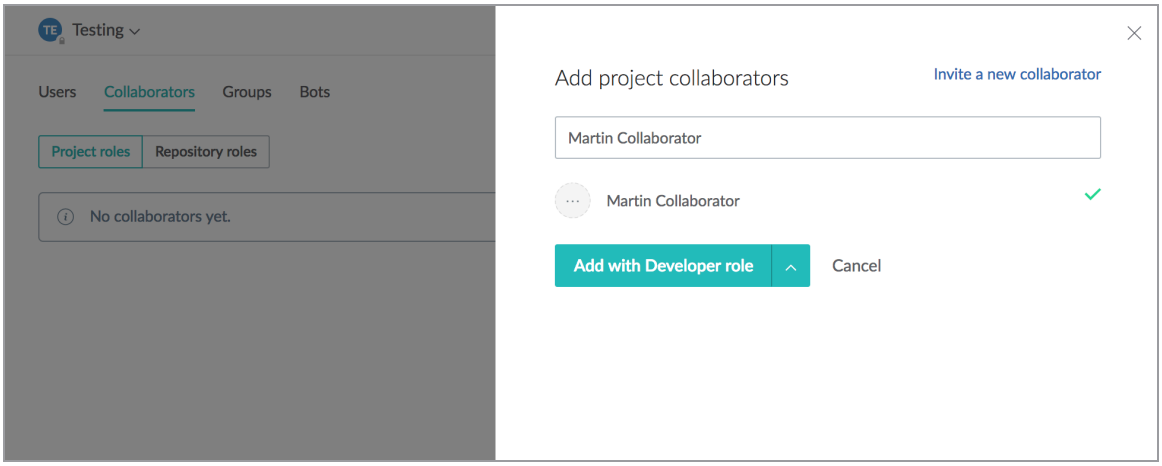
Next, you need to add the collaborator to the project.

Add project collaborators

To add a collaborator to a project:

1. In the **Add project collaborators** form, select one or more collaborators from the list. A green check mark appears next to the selected collaborators.
2. Click **Add with Developer role**, or click the up arrow to the right to select a different role, such as **Guest** or **Master**.

TeamHub adds the collaborator to the **Team** tab > **Collaborators** view.



Delete project collaborators

To remove a collaborator from the project:

1. In the **Team** tab > **Collaborators** view, click the role of the collaborator you want to remove.
2. In the **Select role** form, select **Remove from project**.
3. Click **Select** and confirm your select.

   TeamHub removes the collaborator from the project.

Bots & programmatic repository access

Bots enable access to projects without using user credentials. Helix TeamHub can securely grant program access to projects using either a bot's credentials, SSH keys, or API keys.

TeamHub supports the following types of bots:

- Normal bots: Can be assigned to projects and repositories with guest, developer, or master role.
- Company admin bots: Have access to all projects and repositories by default. Company admin bots can also manage users, collaborators, and groups at the company level.

Privilege	Guest	Developer	Master	Company admin
Create and update build events	✓	✓	✓	✓
Read repositories	✓	✓	✓	✓
Write to repositories[1]		✓	✓	✓
Merge code reviews[2]		✓	✓	✓
Write to protected branches			✓	✓
Merge multi-repo code reviews			✓	✓
Manage users and collaborators[3]				✓
Manage groups and members[4]				✓

Managing bots

Note
Each bot you create for programmatic repository access to your Helix server consumes a seat on your Helix server license and a background license seat on your Helix TeamHub license. You must allow for this when creating your bots, see License plan types and number of seats required in the

[1] Master or admin role is required for writing to a protected branch.

[2] Master or admin role is required for merging a code review where the destination branch is protected.

[3] At the company level only.

[4] At the company level only.

> *Helix TeamHub Administrator Guide*.

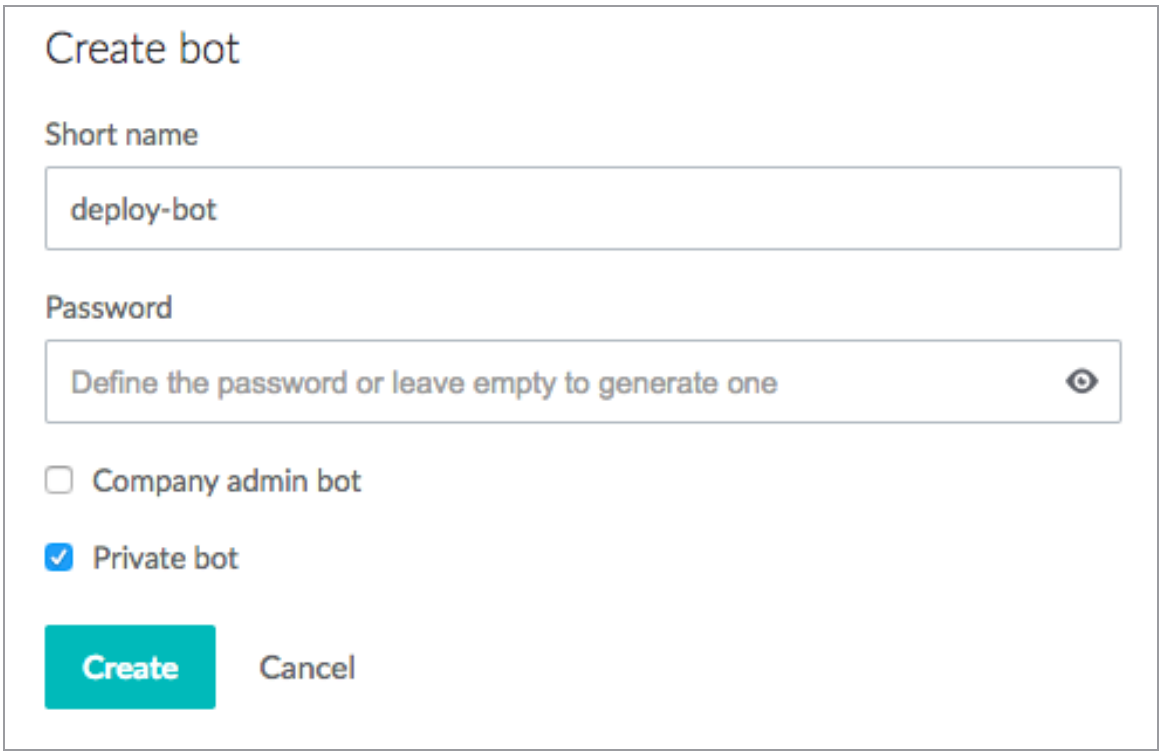
Normal bots can be created by users, assigned to projects by project admins, and assigned to repositories by repository admins. Owners of the bot can manage the bot and its credentials, while members can only access them.

Public bots are visible to everyone. Private bots are only visible to owners and members. Collaborators can only see bots. Company admin bots are only available to company admins.

Creating a new bot

Perform the following steps to create a new bot:

1. To make sure you are at the company scope level, click the company name or logo in the top left corner. Then click **Bots**.
2. In the **Bots** view, click the plus icon to the right of the search field to open the **Create bot** form.



3. In the **Create bot** form, provide the following information:
   - **Short name**: Username for the bot.
   - **Password**: Password for the bot. Leaving the field empty will generate a random password. Bot users can view the password in the bot settings form.

- **Company admin bot**: Select to create a company admin bot instead of a normal bot. This option is only available for company admins and cannot be changed after creation.
- **Private bot**: Select to hide bot from other users.

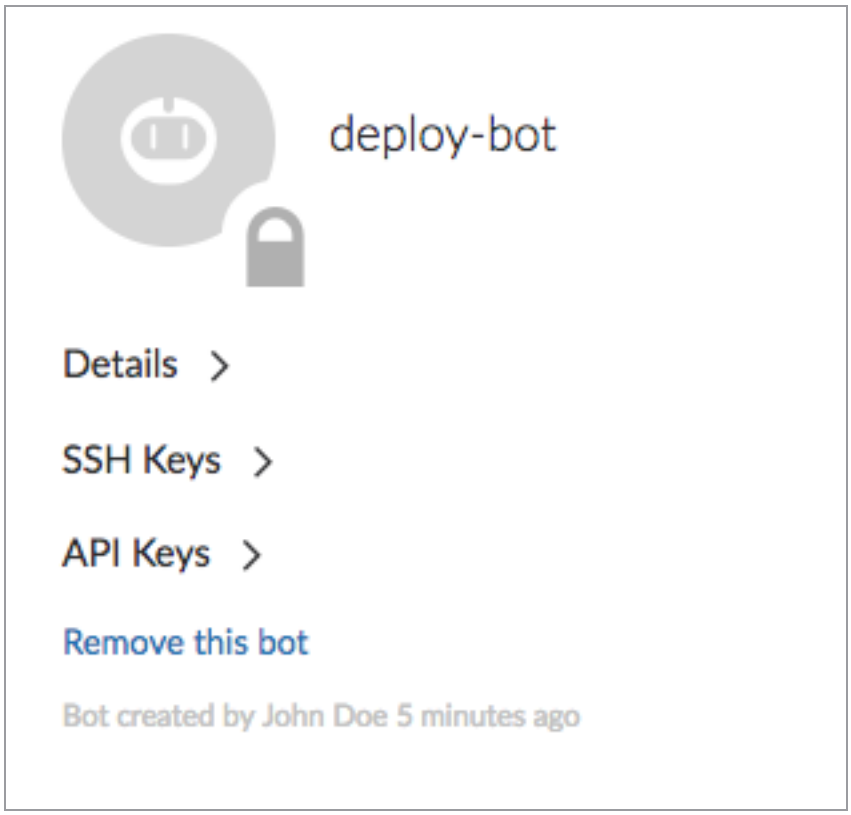
4. Click **Create**.

   After creating a bot, you can manage its settings (excluding **Company admin bot** status) from the bot settings form or the **Bot > Users** view. To access the settings, click the cogwheel icon next to the bot name in the **Bots** tab.

   > **Note**
   > With Helix authentication, you cannot modify the short name after creating a bot.

   You can also delete the bot using the settings form.

Bot settings



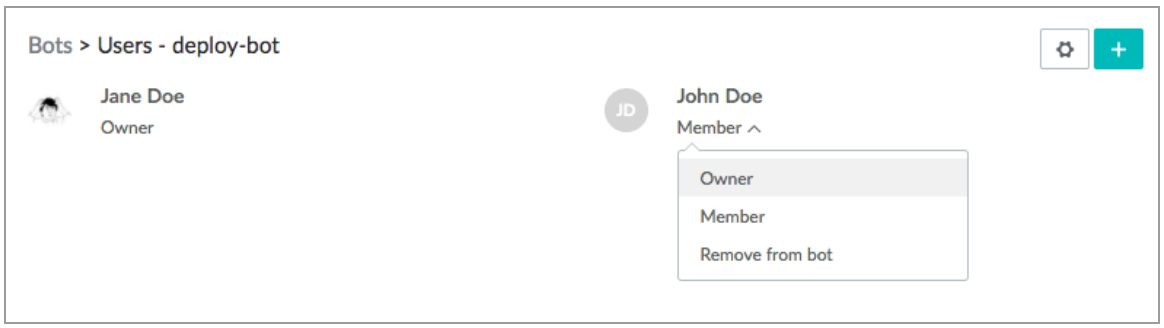
You can manage the username, password, and visibility of the bot (public or private) from the **Details** section.

Bots can have multiple SSH keys, which you can manage from the **SSH Keys** section. Helix TeamHub supports ssh-rsa, ssh-ed25519, and ecdsa keys.

The **Account Key** and **Company Key** of the bot are locatedin the **API Keys** section. These keys are required when you use the bot with integrations and for API access.

Bot users

After creating a bot, you can manage its users from the **Bots > Users** view. To access this view, click a bot in the **Bots** tab.



When you add users to bots, they can have one of the following roles:

- *Member*: Users with this role can always see the bot and its settings (including credentials).
- *Owner*: Users with this role can also manage the bot's settings and users.

Projects

A project can contain one or more repositories. You can grant users role permissions on the project level or, more granular, on the repository level.

Creating a new project

Follow these steps to create a new project:

1. At the company scope, in the **All Projects** view, click the plus icon  to the right of the **Search** field.
2. In the **New project** form, enter a name for the project.

   Helix TeamHub automatically populates the **Short name** field based on the name you provide. This string is used for URLs.
3. **Optional:** edit the **Short name** as needed.

   This field only allows alphanumeric characters, dashes, and underscores.
4. Select any of the following:
   - **Visible to everyone in Helix TeamHub** if you want all users to have access to this project.
   - **Create Wiki** to maintain a collaborative space along with your project. Only available if at least one of the supported Wiki repository types is enabled. For instructions on working with Wikis, see "Wiki" on page 142 .
     - **Type** select the Wiki repository type from the dropdown list.

       > **Tip**
       > Not displayed if only one Wiki repository type is available.
   - **Automate depot creation and permission management** to automatically create a depot and grant depot permissions for the user `gconn-user`.

     > **Note**
     > This option is only available with Helix authentication and if the **Automatic depot creation** feature is turned on for the company.

     When you select this option, TeamHub automatically grants depot-level permissions for masters and admins for the new project. As a result, the **New repository** form in this project appears simplified, only asking for the repository type and short name.
5. **Optional:** enter a **Description** for the project.
6. Click **Create project**.

Accessing an existing project

Follow these steps to access an existing project:

- At the company scope, on **My Dashboard**, click the required project.

Editing a project

Follow these steps to edit a project:

1. At the project scope, under the project name in the right pane, click **Settings**.
2. To modify the project name, project initials, or description, click **Edit details**. Edit as required and click **Save**.
3. In the projects settings form, do any of the following:

   On the **Settings** tab:

   - Select a color for the background of the project initials. To specify a custom color, click the color dropdown, select or specify a color, and click **Select**.
   - To change the project's visibility, click **Make <*project*> visible to everyone in company** or **Make <*project*> hidden from non-members**.

   On the **Maintenance** tab you can either modify the project short name or delete the project:

   - To modify the short name (the string that TeamHub uses for URLs):
     a. Click **Rename project**.
     b. Modify the short name as needed, making sure to use only alphanumeric characters, dashes, and underscores.
     c. Click **Update**.
     d. Remember to update any clone URLs after renaming the project.
   - To delete project:
     a. Click **Delete project**.
     b. When requested, confirm the delete by entering the project name and clicking **Delete**.
4. Click the **X** in the top right corner or click anywhere outside of the form to close the form.

Users

To be able to add or deactivate a user, you need to have the role of *company admin*. For more information, see "Roles" on page 33.

TeamHub sends a registration email to new users. The email includes a link to set up their password. After configuring a password, new users can log in and start using TeamHub. They should make sure to keep their user profiles up to date.

Creating a new user

Follow these steps to create a new use:

1. At the company scope level, click **Users**.
2. In the **Users** view, click the plus icon  to the right of the **Search** field.
3. In the **New user** form, enter the user's email address.
4. (Optional) To notify the user via an email address that is different from the one provided in step 3, select the **Use alternative address for registration e-mail** check box and provide a second e-mail address.
5. (Option) Enter a username.
6. Click **Add user**.

Deactivating a user

Follow these steps to deactivate a user:

1. In the **Users** view, click the user you want to deactivate.
2. In the user form, click **Deactivate this account**.
3. Click **Yes** to confirm.

   TeamHub removes the user from the **Users** view.

Groups

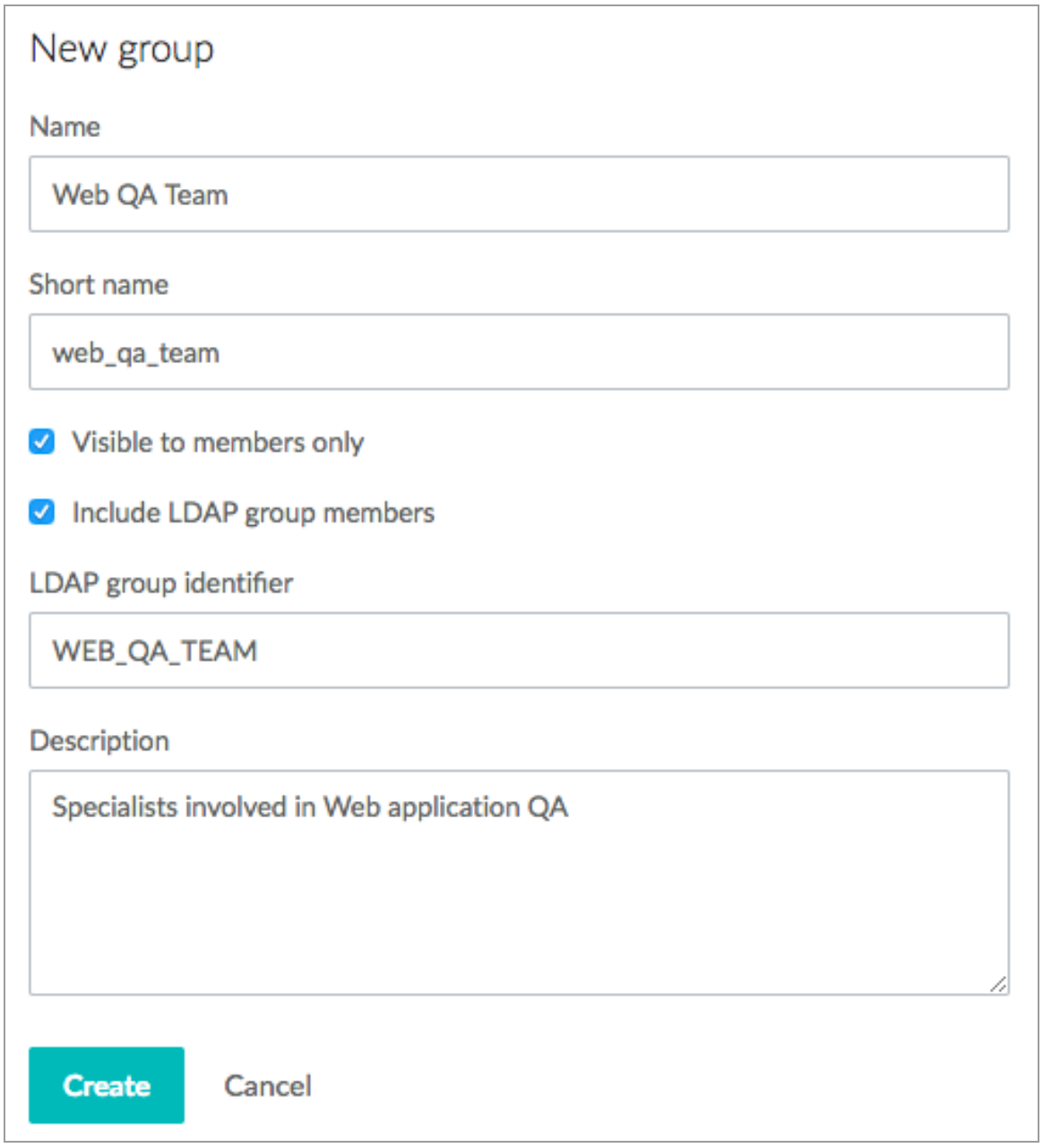
Groups make it easier to manage project and repository roles for multiple users at the same time. Groups can contain only user accounts. Nested groups are not supported.

The **Groups** view is where you add and delete groups. You can then manage team members from the **Groups > Members** view.

Creating a new group

Follow these steps to create a new group:

1. Click the company name or logo in the top left corner to make sure you are at the company scope level. Then click **Groups**.
2. In the **Groups** view, click the plus icon to the right of the search field to open the **New group** form.



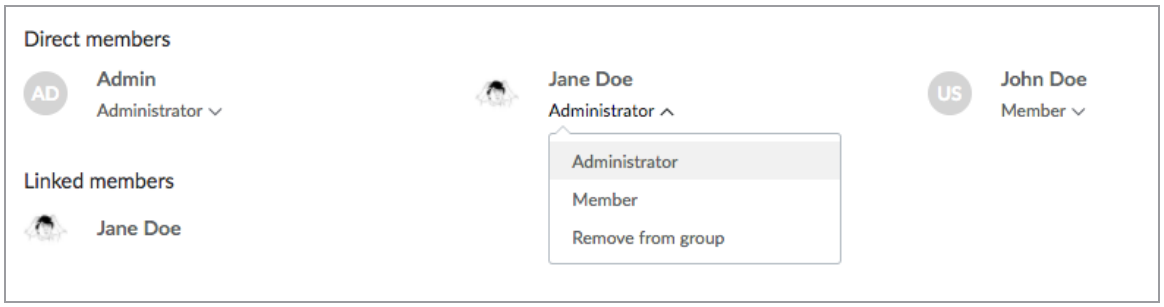
3. Provide the following information:
   - **Name**: Enter a human readable name for the group.
   - **Short name**: Enter a human readable identifier for the group. TeamHub automatically generates the short name from the name. You can modify it before creating the group, but once the group exists, you cannot change it.

- **Visible to members only**: Select to hide the group's members and details from other users.
- **Include LDAP group members**: Select to bring up *LDAP group identifier* input for mapping an LDAP group to the Helix TeamHub group by using either the distinguished name (dn) or the common name (cn) of the LDAP group. You can use this option with LDAP authentication, or when users are synchronized from LDAP. Only users that exist in Helix TeamHub are linked to the group.
- **Description**: (Optional) Enter a free-form description for the group.

After creating the group, you can manage these settings (except for **Short name**) from the group settings form (**Groups** tab > click cogwheel icon next to the group name) or from the **Groups > Members** view (only visible if you are an admin). Admin users can also delete a group using the group settings form.

Group members

After creating a group, you can manage its members by navigating to the **Groups > Members** view from the **Groups** tab. In the **Groups** tab, click the group whose members you want to view or manage.



Groups have two categories of members: direct members and linked members. Members of both categories behave the same way regarding privileges assigned to the group, but only direct members can be managed from the UI.

Direct members

Direct members can have one of the following roles:

- **Member**: Get the privileges assigned to the group and can always see the group and its details.
- **Administrator**: Behave the same way as users with the **Member** role but can also manage group members and group details and delete the group.

Linked members

Linked members behave the same way as users with the **Member** role, but you cannot manage their group membership from the UI. Instead, you manage linked members on the external LDAP directory. You can also add a linked member as a direct member to the group to grant the **Administrator** role for the group.

Roles

Your company's work is grouped into Helix TeamHub projects. Each project has a team with their own tasks, bots, source code, and so on. Project roles control what team members are allowed to do in the project. Optionally, repository roles allow further access control over repositories. Both project and repository roles can be managed individually or in batches in the project team view.

Project roles

Privilege	Public [1]	Guest	Developer	Master	Manager [2]	Admin [2]
Access project contents	✓	✓	✓	✓	✓	✓
See and create issues	✓	✓	✓	✓	✓	✓
See code reviews or multi-repo code reviews	✓	✓	✓	✓	✓	✓
See milestones	✓	✓	✓	✓	✓	✓
See repositories and wikis	✓	✓	✓	✓	✓	✓
Update and delete issues [3]			✓	✓	✓	✓
Manage milestones				✓	✓	✓
Manage code reviews or multi-repo code reviews			✓	✓		✓
Merge code reviews [4]			✓	✓		✓

Privilege	Public [1]	Guest	Developer	Master	Manager [2]	Admin [2]
Write to repositories and wikis			✓	✓		✓
Merge multi-repo code reviews [4]				✓		✓
Write to protected branches				✓		✓
Change project settings					✓	✓
Manage project members [5]					✓	✓
Manage hooks						✓
Manage project bots						✓
Manage repositories						✓

- [1] Public projects are visible in project listings to everyone. You can set the visibility in project settings.
- [2] Manager and Admin roles are not available for collaborators.
- [3] Creator of the issue can also update and delete the issue.
- [4] Merging code reviews with a protected destination branch is limited to master and admin roles.
- [5] Managers cannot manage admin roles.

See Bots & programmatic repository access for bot privileges.

Repository roles

Repository related authorization can be further controlled with repository roles. Enabling authorization for a repository assigns project admin users and groups with admin role to the repository by default. After enabling repository specific authorization, project roles have no effect anymore over the repository.

Privilege	Guest	Developer	Master	Manager [1]	Admin [1]
See code reviews or multi-repo code reviews	✓	✓	✓	✓	✓
See repository	✓	✓	✓	✓	✓

Privilege	Guest	Developer	Master	Manager [1]	Admin [1]
Manage code reviews or multi-repo code reviews		✓	✓		✓
Merge code reviews [2]		✓	✓		✓
Write to repository		✓	✓		✓
Merge multi-repo code reviews [2]			✓		✓
Write to protected branches			✓		✓
Manage repository					✓
Manage repository hooks					✓
Manage repository members					✓

- [1] Manager and Admin roles are not available for collaborators.
- [2] Merging code reviews with a protected destination branch is limited to master and admin roles.

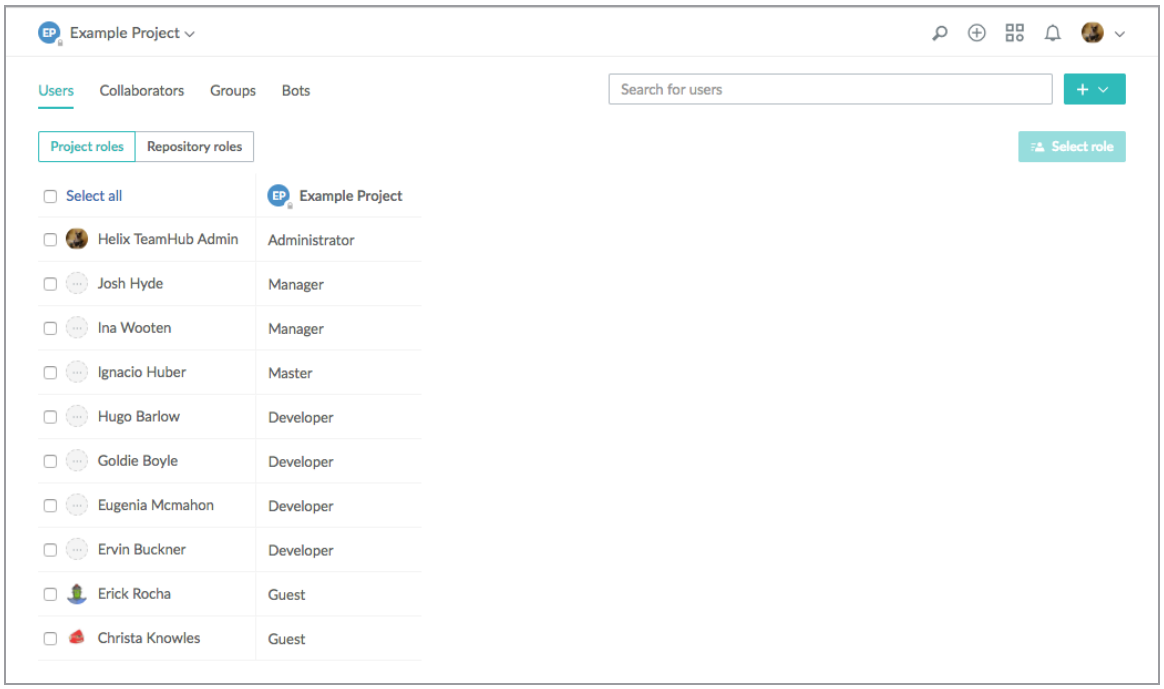
See Bots & programmatic repository access for bot privileges.

Team management

A project's team members can include users, groups, collaborators, and bots. Projects are private by default. This means that you have to explicitely grant access to a project. You can mark a project as public (open to everyone in the company) when you create it or by editing its settings later on. To access the project settings, click **Settings** in the right-side menu in the **Activity** view.

In Helix TeamHub, each repository always belongs to a single project. In addition to project roles, it is possible to configure member permissions per repository. Project-level permissions are often sufficient for companies with few users and repositories, while repository-level roles allow more fine-grained access control over specific repositories.

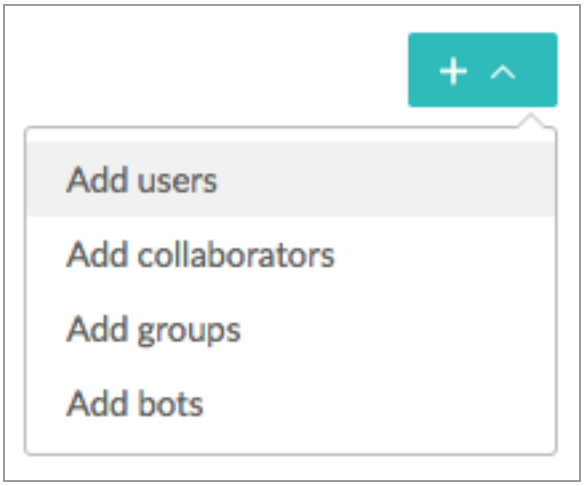
You manage project- and repository-level permissions in the **Team** view. This view is divided into the following tabs: **Users**, **Collaborators**, **Groups**, and **Bots**. Each tab in turn includes the sub-tabs **Project roles** and **Repository roles** for the different level of permissions configured.



Project Level Permissions

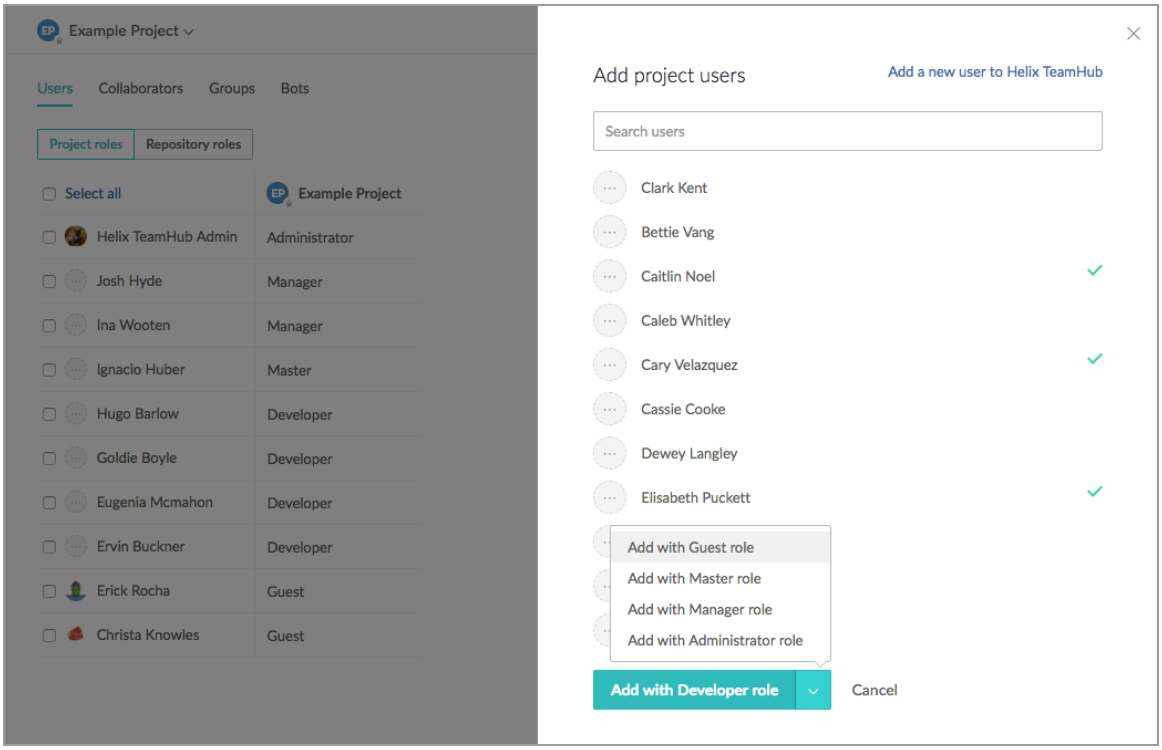
You manage project-level permissions on the **Project roles** tab. Do the following:

1. Click the drop-down list on the right to add project users.



2. In the **Add project users** form, select new team members and then specify the role that you would like to assign to them. For more information about the available roles, see Project &

Repository Roles.

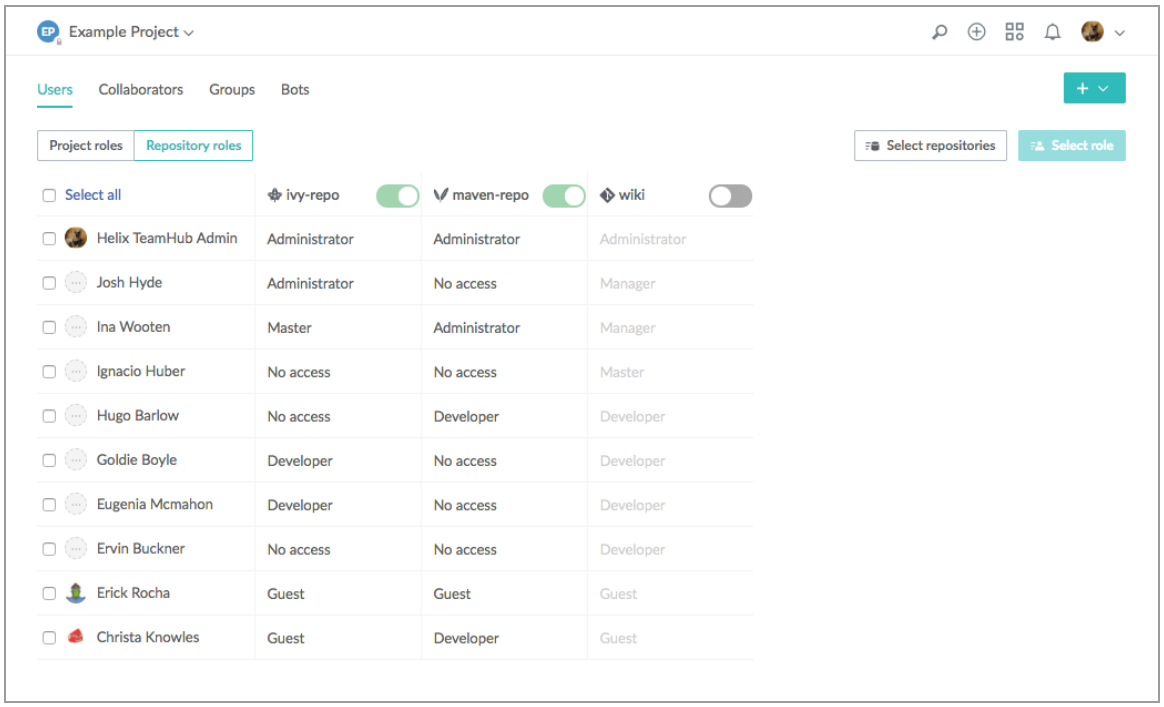


Repository Level Permissions

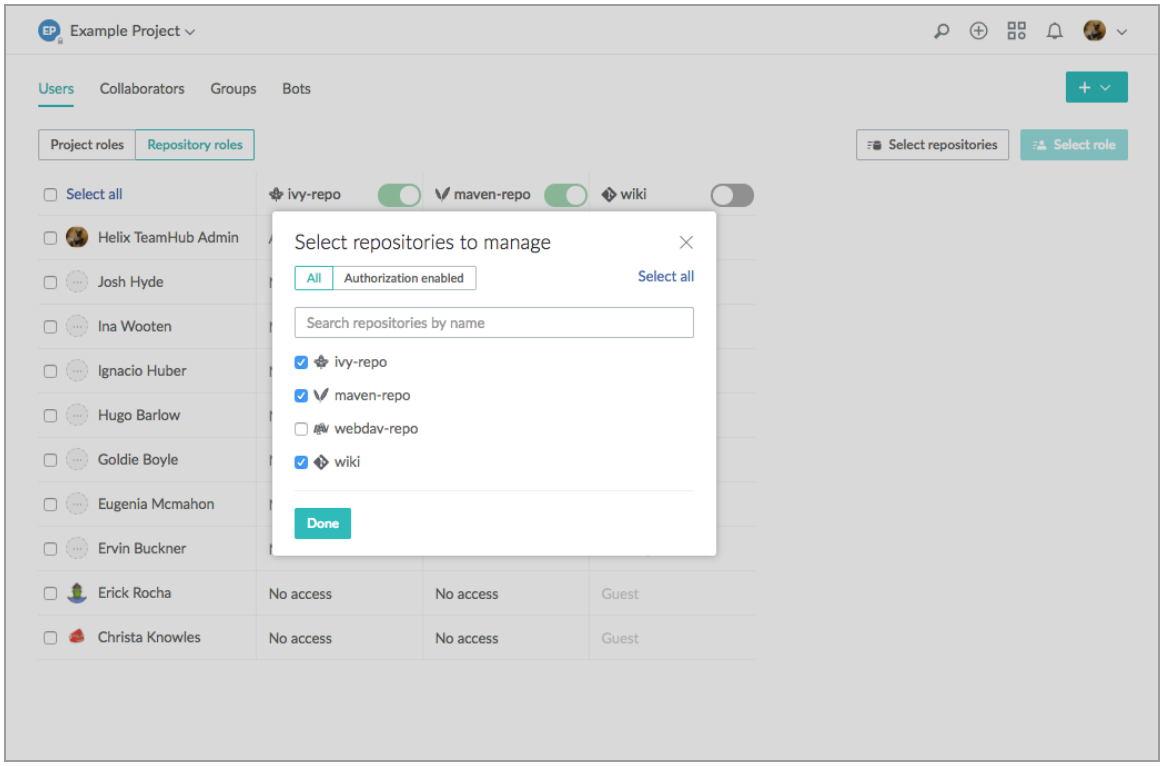
You manage repository-level permissions on the **Repository roles** tab. The team members listed on this tab always match those listed on the **Project roles** tab. For each repository, you need to explicitly turn on repository-level authorization.

To turn on repository-level permissions, click the button next to a repository name to turn on authorization.

Important note: When enabling repository authorization, keep in mind that by default, TeamHub only grants the current project administrators access to the repository. Likewise, disabling repository authorization downgrades all given roles to match the current project roles.



To specify the repositories to manage, click **Select repositories**. In the **Select repositories to manage** form, select (or clear) the check boxes of the required repositories and click **Done.**



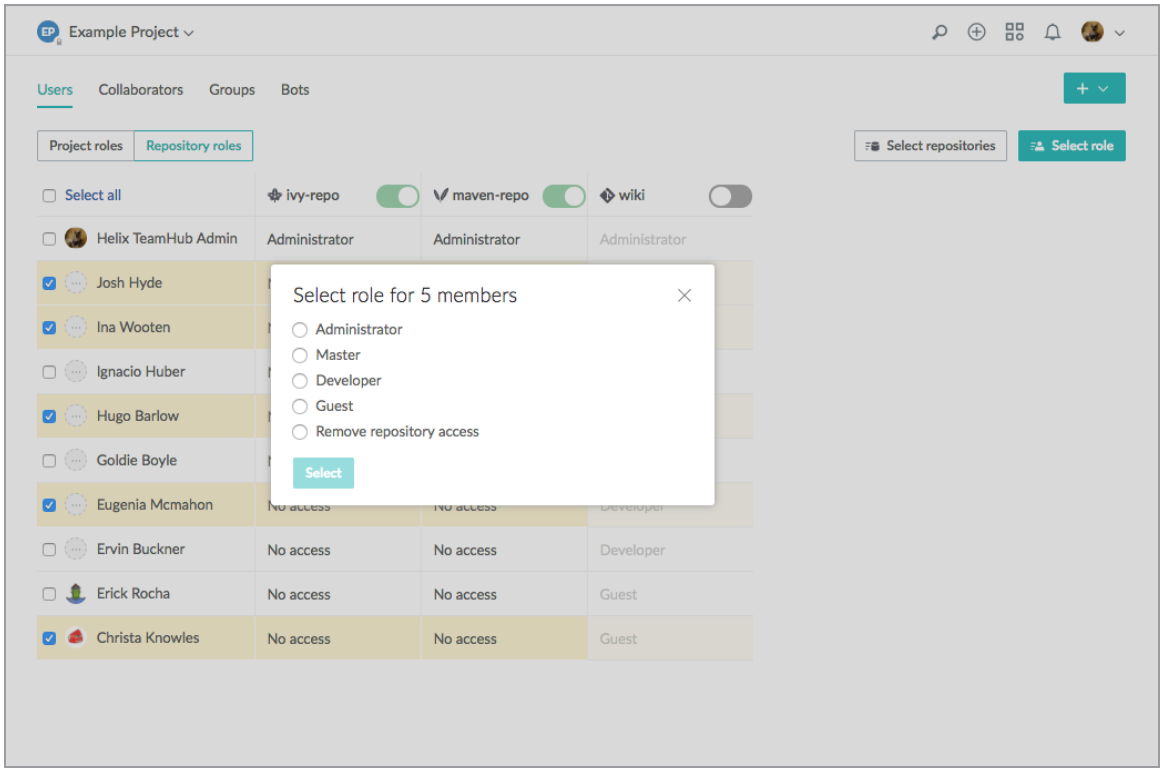
For more information on how repository-level permissions affect access rights, see Project & Repository Roles.

Role selection and batch actions

When permissions are turned on for a repository, you can assign a role to each member of the team.

To assign a role:

1. Select one or more members and click **Select role**.



2. In the **Select role** form, select the required role or select **Remove repository access** if you want to remove the selected member or members from the project or revoke a member's access to the repository.
3. Click **Select**.

Repositories

This section includes the following information:

Repositories & version control

Helix TeamHub hosts all your code. When you use TeamHub for storing your code, you let your team always see your latest changes as well as get the latest changes from others.

Currently, TeamHub supports Git, Mercurial, Subversion, WebDAV, Maven, Ivy, and Docker repositories with the following access protocols.

Repository	Git	Mercurial	Subversion	WebDAV	Maven	Ivy	Docker
SSH	✓	✓	✓				
HTTP	✓	✓	✓	✓	✓	✓	
HTTPS	✓	✓	✓	✓	✓	✓	✓
Helix P4	✓						

Note
With Helix authentication, certain restrictions apply to TeamHub functionality. For details, see "Limitations with Helix authentication" on page 11.

TeamHub supports the following combination of features and repository types:

Repository	Git	Helix Git	Mercurial	Subversion	WebDAV	Maven	Ivy	Docker
Edit/Commit from the UI	✓	✓			✓			
Code search	✓	✓	✓					
Code review	✓	✓	✓					
Multi-repo code review		✓						
Compare view	✓	✓	✓	✓				
Branch from UI	✓	✓						
Fork from UI	✓		✓					
Manage protected branches	✓	✓						
"Wiki" on page 142	✓		✓					
Garbage collection	✓							
Change UUID				✓				
Repository Site Replication				✓				

Repository	Git	Helix Git	Mercurial	Subversion	WebDAV	Maven	Ivy	Docker
"Helix TeamHub CLI tool - Technology preview feature" on page 144	✓	✓						

Version control with Git

Git is an open source distributed version control system initially designed for Linux kernel development. In Helix TeamHub, you can create any number of Git repositories in your projects. Git is also used to host Wiki pages and attachments.

The below is the list of resources to help you learn Git:

- Official Git SCM site
- Try Git interactively in 15 min
- Pro Git book

If your TeamHub instance is set up with Helix authentication, all Git repositories are stored in Helix server. Helix server is a secure, scalable, and highly available version control system that supports parallel development. An unlicensed Helix server is limited to 10 repositories. With a license key, you can create any number of Git repositories in your projects.

Following is a list of resources to help you get familiar with Helix server:

- *Solutions Overview: Helix Version Control System*
- *Helix Core P4 Command Reference*, in particular the graph depot commands.

This section includes information on:

- "Setting up" on the facing page
- "Creating Git repositories stored in Helix server" on the facing page
- "Cloning a repository" on page 44

- "Migrating existing Git repositories to Helix TeamHub" on page 45
- "Migrating existing Git repositories into Helix server via Helix TeamHub" on page 47
- "Importing an existing graph depot (Helix4Git repo) into Helix TeamHub" on page 49

Setting up

Download and install latest version of Git client and open your *terminal*.

Introduce your name and email to Git, so that Helix TeamHub can display your user account correctly. Please specify the same email address you use to sign in to Helix TeamHub:

```
git config --global user.name YOUR NAME
git config --global user.email HELIX TEAMHUB EMAIL ADDRESS
```

Configure your SSH keys under your TeamHub Profile settings and start working with Git.

> **Note**
> If your instance uses Helix authentication, wait 10 minutes for the SSH keys to sync. Otherwise, the Git Connector will not have the updated SSH keys in the list of authorized keys, and you will not be able to connect.

Creating Git repositories stored in Helix server

The steps for creating a Git repository to be stored in Helix server depend on whether the **Automatic depot creation** feature is turned on or off for the related project. For more information this feature, see "Creating a new project" on page 26.

To connect to your clean repository using Git or the P4 command line client, follow the instructions in TeamHub. If your repository already has data, it is displayed on the page. In this case, continue with cloning the repository. For more information on setting up Git repositories, see "Version control with Git" on the previous page.

Without automatic depot creation

You need to provide the name of an existing graph depot and a short name for the new repository. As you provide this information, TeamHub dynamically creates the path to the new repository using the following format: ***//<graph-depot-name>/<project-short-name>/<repository-short-name>***

For example: **`//test-depot/test/test-repo`**

You can edit this path to meet your needs, but it is recommended that you use the path provided.

To create a Git repository to be stored in Helix server:

1. On the **Repositories** tab, click the add button  .
2. In the **New repository** form, from the list of repository types, select **Git**.

3. Select a depot from the list of existing graph depots in Helix server.
4. Enter a short name for the repository. This is the repository name displayed in the TeamHub UI.
5. (Optional) If needed, edit the path to the repository within the depot you selected.

   > **Note**
   > Modifying the path is not recommended.

6. Verify that the full path displayed at the bottom of the form is correct.
7. Click **Create**.

With automatic depot creation

You only need to provide a repo short name for the new repository. The repository is created using the following format: ***//<project-short-name>/<repository-short-name>***

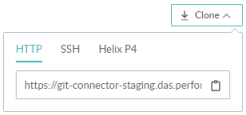
1. On the **Repositories** tab, click the add button +.
2. In the **New repository** form, from the list of repository types, select **Git**.
3. Enter a short name for the repository. This is the repository name displayed in the TeamHub UI.
4. Click **Create**.

Cloning a repository

For a repository that includes content, go ahead and clone the repository.

If your instance uses Helix authentication and you want to use p4 commands, see the *Helix Core P4 Command Reference*, in particular the graph depot commands.

1. In the **Repositories** view, click the Git repository you want to clone.
2. Click **Clone**, select the authentication method (**HTTP**, **SSH**, or **Helix P4**), and copy the path.



3. Do one of the following:
   - If you selected HTTP or SSH, open the command line client and run the following command:

     ```
     git clone <paste the path you copied>
     ```

     For example, for user 'hth' to clone the "bar" Git repository from the "foo" project in "acme" company to host 'helixteamhub.cloud' with SSH, issue the following command in the Terminal:

```
git clone
hth@helixteamhub.cloud:acme/projects/foo/repositories/git/bar
```

To clone the same repository with HTTP:

```
git clone
http://
hth@helixteamhub.cloud/acme/projects/foo/repositories/git/bar
```

- If you selected **Helix P4**, open a p4 command line client and run the following command to create a client of type graph; then edit the client spec to update the view mapping.

```
p4 -p <paste the port you copied> -u <user> client -T graph
<client name>
```

For example, to create a client workspace of type **`graph`** called 'hth-testing' for user 'hth' on host (p4port) 'hth.test.perforce.com:1666', issue the following command in the p4 command line client:

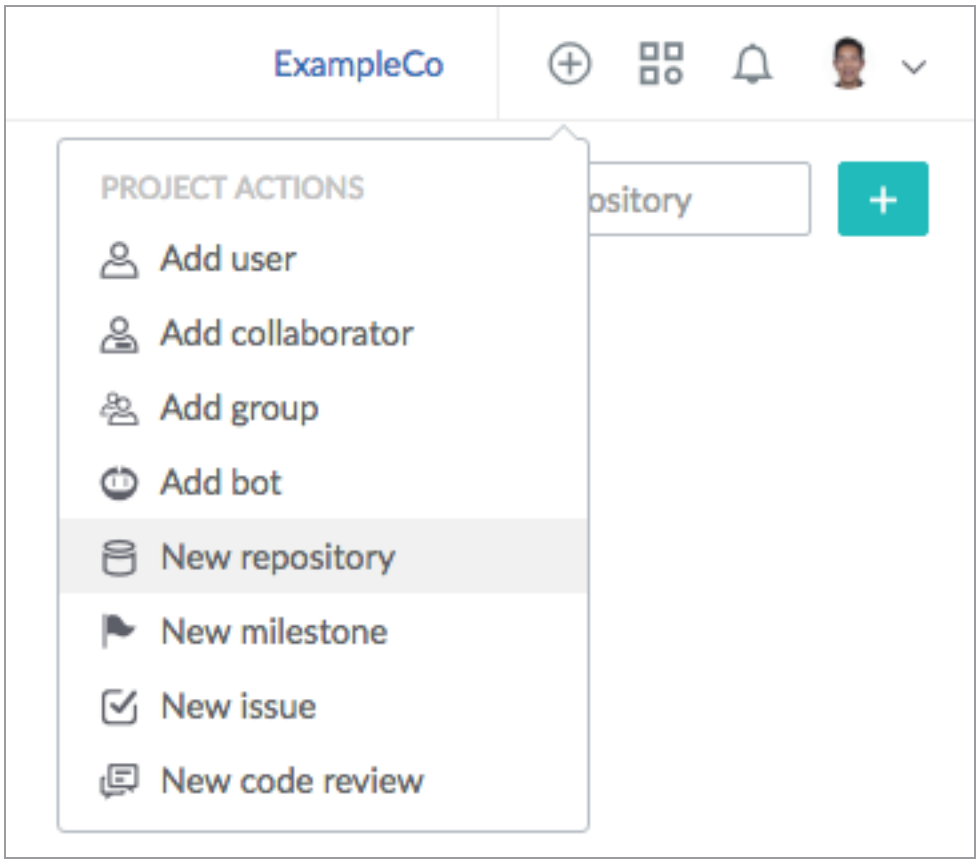
```
p4 -p hth.test.perforce.com:1666 -u hth client -T graph hth-
testing
```

For more information on creating client workspaces for graph depots, see the *Helix4Git Administrator Guide* and the *Helix Core P4 Command Reference*, sections Including Graph Depots repos in your client and p4 client (graph).

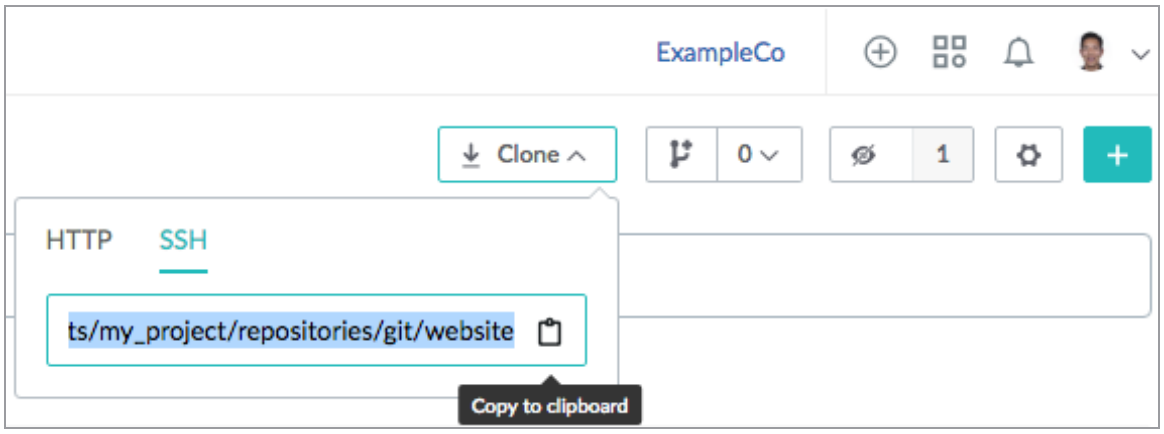
Migrating existing Git repositories to Helix TeamHub

The following steps are needed in order to migrate existing Git repositories to Helix TeamHub:

1. Create a new repository in your TeamHub project. You can accomplish this either through quick actions or from the **Repositories** view:



2. Clone the repository you wish to migrate:
   `git clone --bare <OLD-REPOSITORY-URL>`
3. Go to the repository directory:
   `cd <YOUR-REPOSITORY-NAME>`
4. Obtain the clone URL for the repository you created in Helix TeamHub from the repository view:



5. Change the clone URL of the repository you cloned to point to the repository you created in Helix TeamHub with the following command:

   ```
   git remote set-url origin <HELIX-TEAMHUB-REPOSITORY-URL>
   ```

6. Push the repository to Helix TeamHub:

   ```
   git push --mirror
   ```

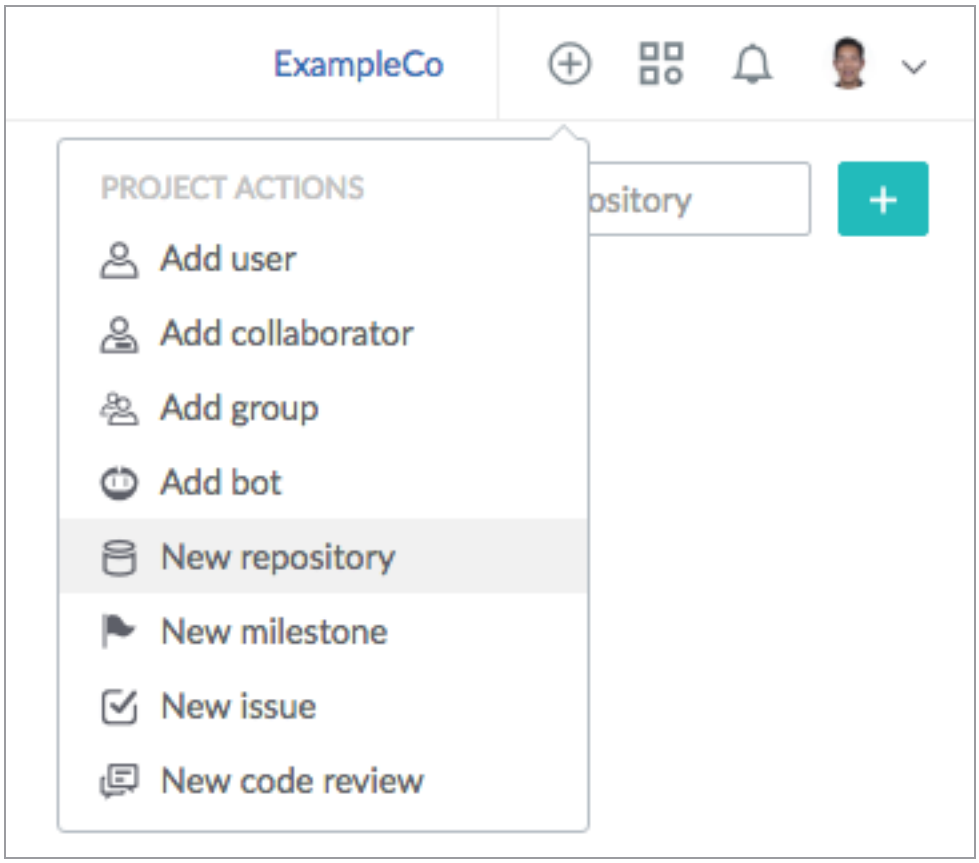
   If you copied the clone URL with SSH as an access method, you need to have your SSH key added to Helix TeamHub in order to push the changes.

Migrating existing Git repositories into Helix server via Helix TeamHub

For Git repositories not in Helix server, you need to perform the following steps to migrate existing Git repositories to Helix server via TeamHub.

1. Create a new repository in your TeamHub project:

   You can accomplish this either through quick actions or from the **Repositories** view:



For details, see "Creating Git repositories stored in Helix server" on page 43.

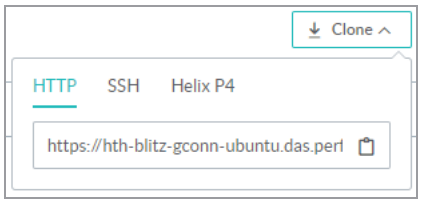
2. Clone the repository you wish to migrate:

   ```
   git clone --bare <OLD-REPOSITORY-URL>
   ```

3. Go to the repository directory:

   ```
   cd <YOUR-REPOSITORY-NAME>
   ```

4. Obtain the clone URL for the repository you created in TeamHub from the repository view:

5. Change the clone URL of the repository you cloned to point to the repository you created in TeamHub with the following command:

   ```
   git remote set-url origin <HELIX-TEAMHUB-REPOSITORY-URL>
   ```

6. Push the repository to TeamHub:

```
git push --mirror
```

For existing Git repositories in Helix server, the following steps are needed to migrate existing Git repositories to Helix server via TeamHub.

Follow steps here: "Creating Git repositories stored in Helix server" on page 43. Make sure that you select the correct depot and modify the path to point to your existing repository.

Importing an existing graph depot (Helix4Git repo) into Helix TeamHub

If you are a current user of Helix4Git and you want to start using Git or new users who joined the project want to use Git, you can import existing graph depots into TeamHub. This involves creating a new Git repository in TeamHub and providing the correct path to the graph depot.

> **Tip**
> Because you are just adding repositories that already exist to your Helix TeamHub project, they can have unusual paths.

With automatic depot creation: when you import an existing repo in a graph depot, create a project with the same name as the graph depot you are importing the repository from. Then select **Import Helix Repository** when adding the Git repository to the project.

Without automatic depot creation: Helix TeamHub allows you to add existing repositories from any graph depot available for import. You can modify its short name in Helix TeamHub but the existing graph depot path remains unchanged. Use Helix server to manage permissions for users and the `gconn-user` for the repository to be available.

To import an existing graph depot repository:

Before you start, make sure that the `gconn-user` and the users have the correct permissions for the graph depot and its repositories. For instructions on setting permissions, see Grant permissions in the *Helix4Git Administrator Guide*.

1. Create a new repository:
2. On the **Repositories** tab, click the add button  .
3. In the **New repository** form, from the list of repository types, select **Git**.
4. Click **Import Helix repository** at the top of the form.
5. Select the repository to add.
6. The **Short name** is the repository name displayed in the TeamHub UI. We recommend that you leave it as it is, but you can change it if you need to.
7. Click **Create**.
8. Clone the repository.

Version control with Mercurial

Mercurial is an open source distributed version control system initially designed as a replacement tool for Linux kernel development. In Helix TeamHub, you can create any number of Mercurial repositories in your projects.

The below is the list of resources to help you learn Mercurial:

- Official Mercurial site
- Hg Init: a Mercural Tutorial
- Mercurial: The Definitive Guide

Setting up

Download and install the latest version of Mercurial client and open your *Terminal*.

Introduce your name and email to Mercurial, so that Helix TeamHub can display your user account correctly. Please specify the same email address you use to sign in to Helix TeamHub:

Open `~/.hgrc` (or on a `Windows system in %USERPROFILE%\Mercurial.ini`) by creating it and adding the following lines:

```
username = YOUR NAME <[HELIX TEAMHUB EMAIL ADDRESS]>
```

Configure your SSH keys under your Helix TeamHub Profile settings and start working with Mercurial.

For example, to clone the "bar" Mercurial repository from the "foo" project in "acme" company with SSH, issue the following command in the terminal:

```
hg clone
ssh://hth@helixteamhub.cloud/acme/projects/foo/repositories/git/bar
```

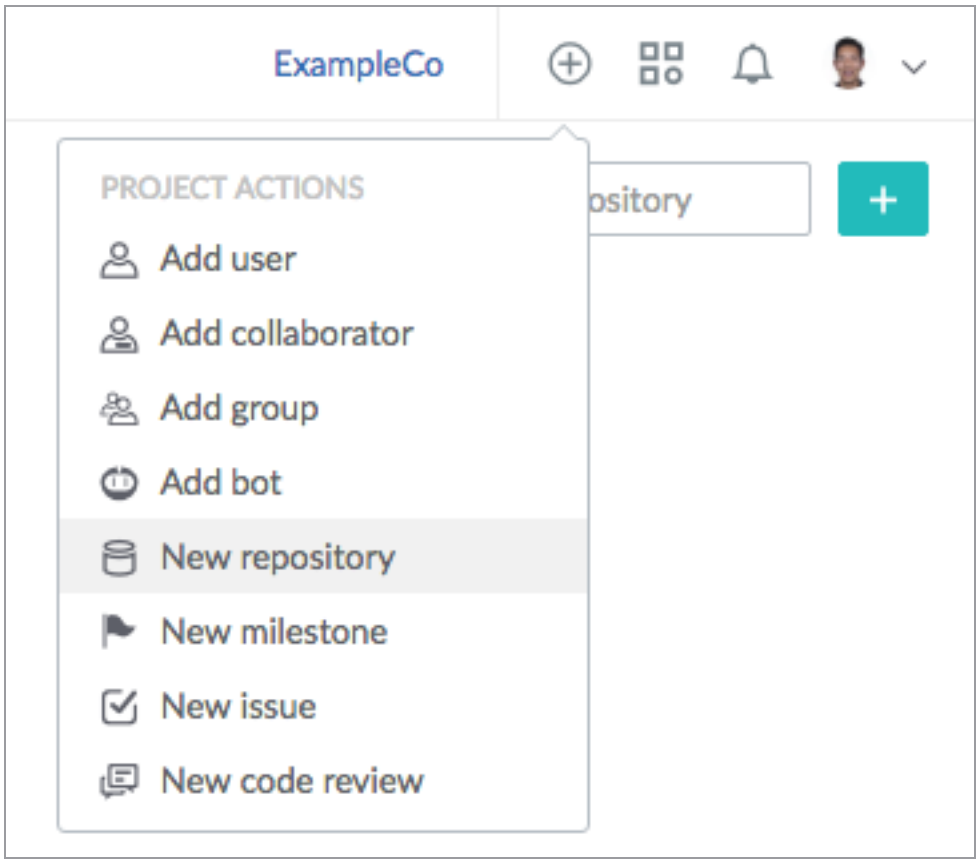
To clone the same repository with HTTP:

```
hg clone http://[HELIX TEAMHUB
USERNAME]@helixteamhub.cloud/acme/projects/foo/repositories/mercurial/bar
```

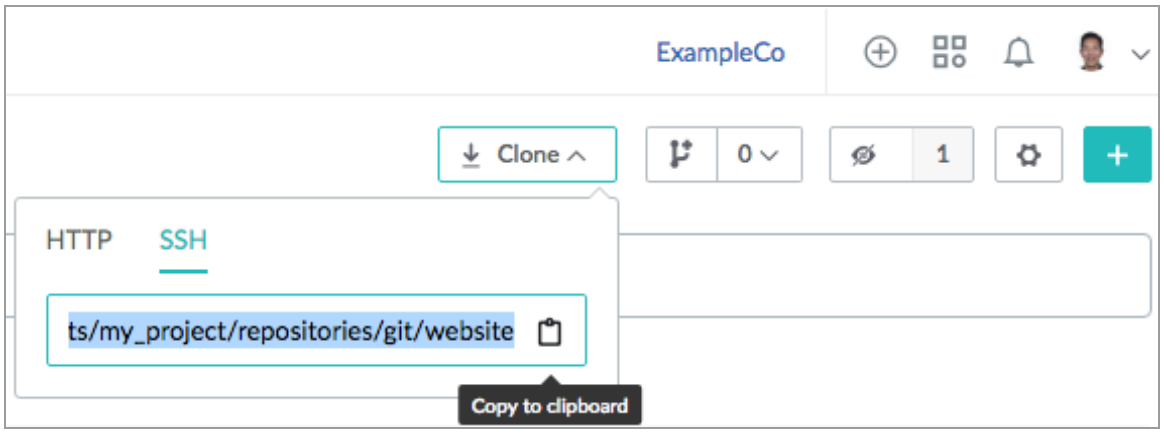
Migrating existing Mercurial (HG) repositories to Helix TeamHub

The following steps are needed in order to migrate existing Mercurial repositories to Helix TeamHub:

1. Create a new repository in Helix TeamHub project. You can accomplish this either through quick actions or from the repositories screen:



2. Clone the repository you wish to migrate:
   `hg clone <OLD-REPOSITORY-URL>`
3. Go to the repository directory:
   `cd <YOUR-REPOSITORY-NAME>`
4. Obtain the clone URL for the repository you created in Helix TeamHub from the repository view:



5. Push the repository to Helix TeamHub:

   ```
   hg push <HELIX-TEAMHUB-REPOSITORY-URL>
   ```

   If you copied the clone URL with SSH as an access method, you need to have your SSH key added to Helix TeamHub in order to push the changes.

6. Clone the repository from Helix TeamHub to a new directory. Alternatively, you can change the new clone URL to the `[paths]` section of `.hg/hgrc` file in the repository.

Version control with Subversion

Subversion is an open source version control system developed as a project of the Apache Software Foundation. In Helix TeamHub, you can create any number of Subversion repositories in your projects.

The below is the list of resources to help you learn Subversion:

- Official Subversion site
- Version Control with Subversion book

Setting up

Download and install the latest version of Subversion client and open your *Terminal*.

Configure your SSH keys under your Helix TeamHub Profile settings and start working with Subversion.

For example, to checkout the "bar" Subversion repository from the "foo" project in "acme" company with SSH, issue the following command in the Terminal:

```
svn checkout
svn+ssh://hth@helixteamhub.cloud/acme/projects/foo/repositories/subversion
/bar
```

To checkout the same repository with HTTPS:

```
svn checkout --username [HELIX TEAMHUB USERNAME]
https://helixteamhub.cloud/acme/projects/foo/repositories/subversion/bar
```

Instructions for using Subversion over SSH in Windows

This article goes through step by step instructions on how to use Subversion over SSH in Windows. We are going to use TortoiseSVN in combination with SSH keys generated with Putty.

Prerequisites

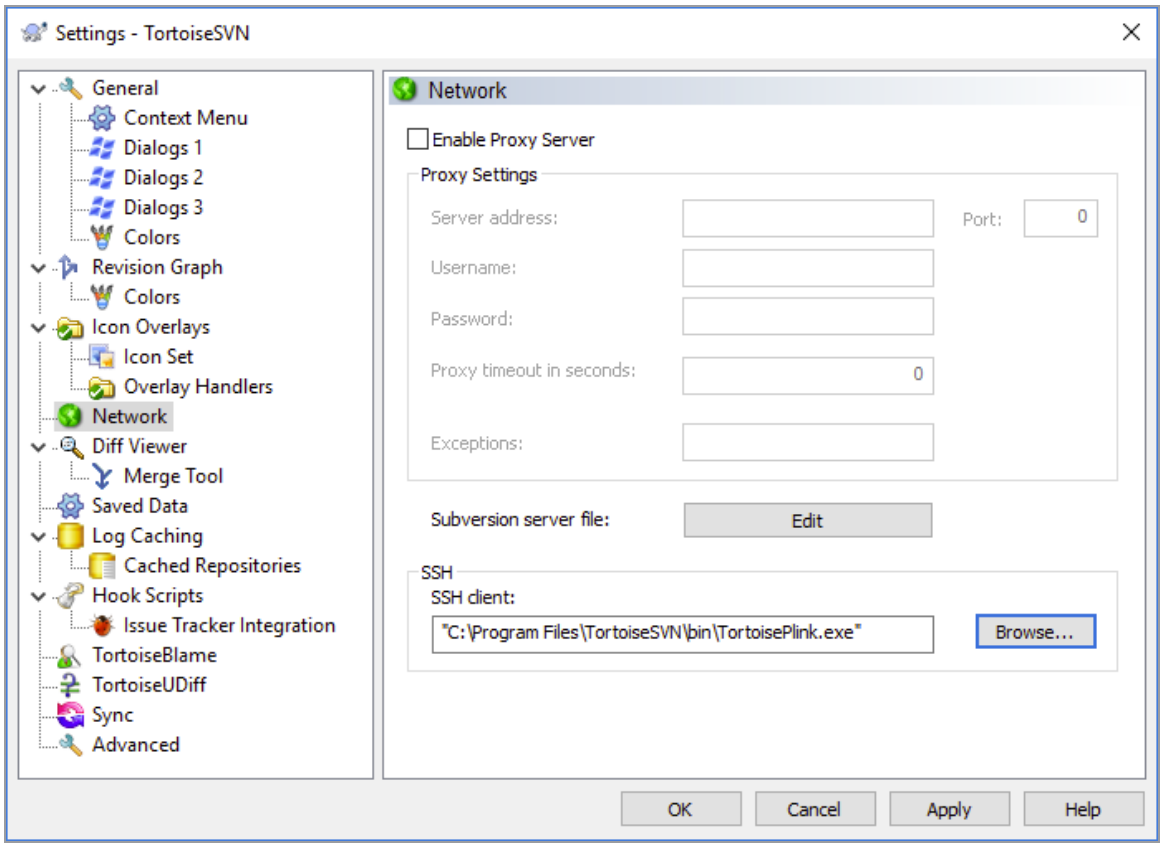
In order to use subversion over SSH in Windows we are going to use TortoiseSVN application. You can download TortoiseSVN from their official site. We are also using Putty to generate a SSH key and handle communication with Helix TeamHub. You can download Putty from their official site. Download and install the Windows MSI installer package for Putty to have everything needed.

In order for TortoiseSVN to find Putty executables, you need to add the Putty install directory path to the **`Path`** variable. You can do this from the control panel. If you are using the Windows MSI installer package for Putty, this is done automatically at least for version 0.67.

In addition to having the aforementioned software installed, you should have a Helix TeamHub project and Subversion repository under it.

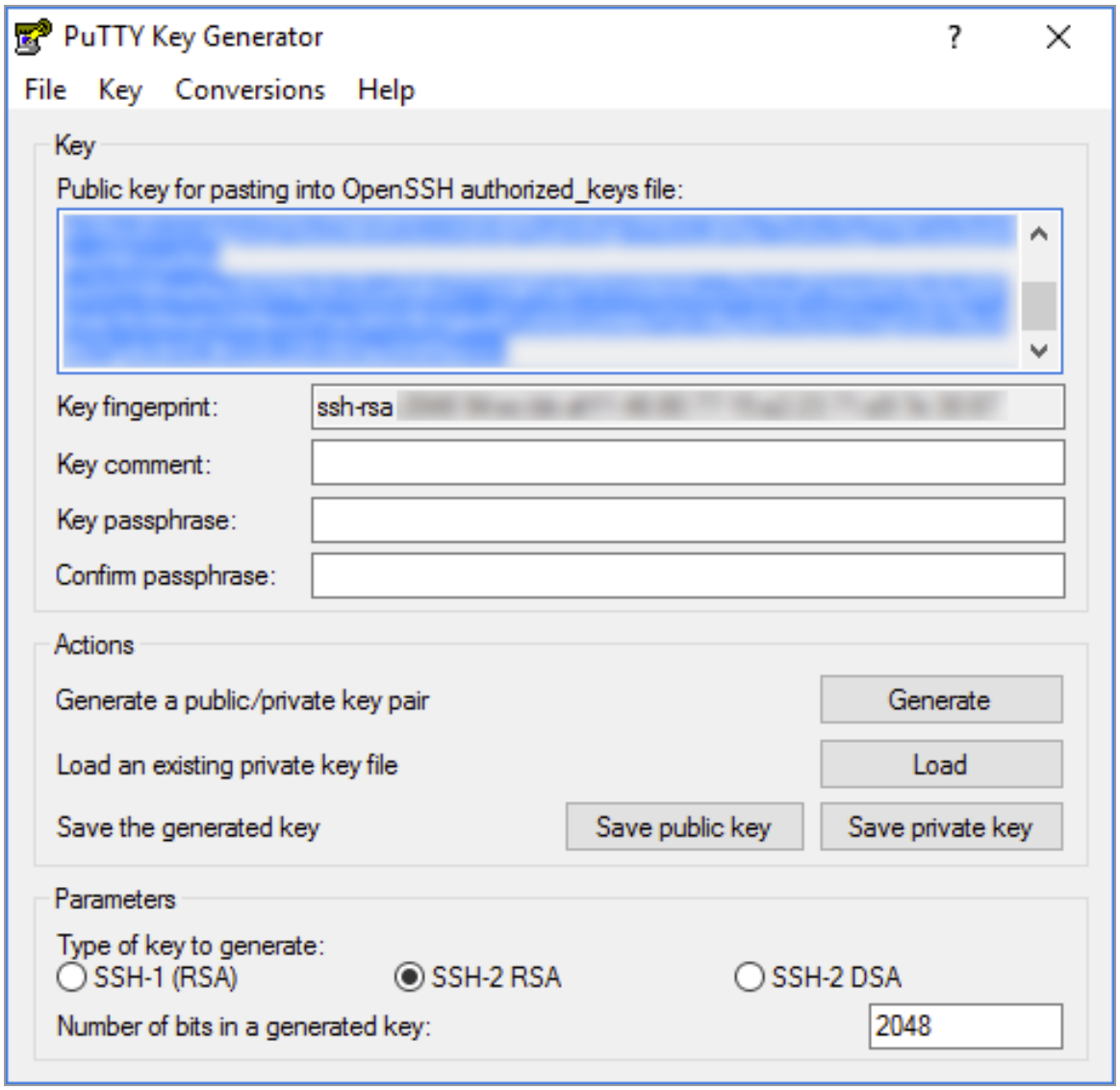
Configure TortoiseSVN to use SSH

Open TortoiseSVN settings from the start menu. Go to the Network tab, and setup **`TortoisePlink.exe`** as your SSH Client. You may find **`TortoisePlink.exe`** under **`bin`** directory from the TortoiseSVN installation directory, e.g. **`C:\Program Files\TortoiseSVN\bin`**.



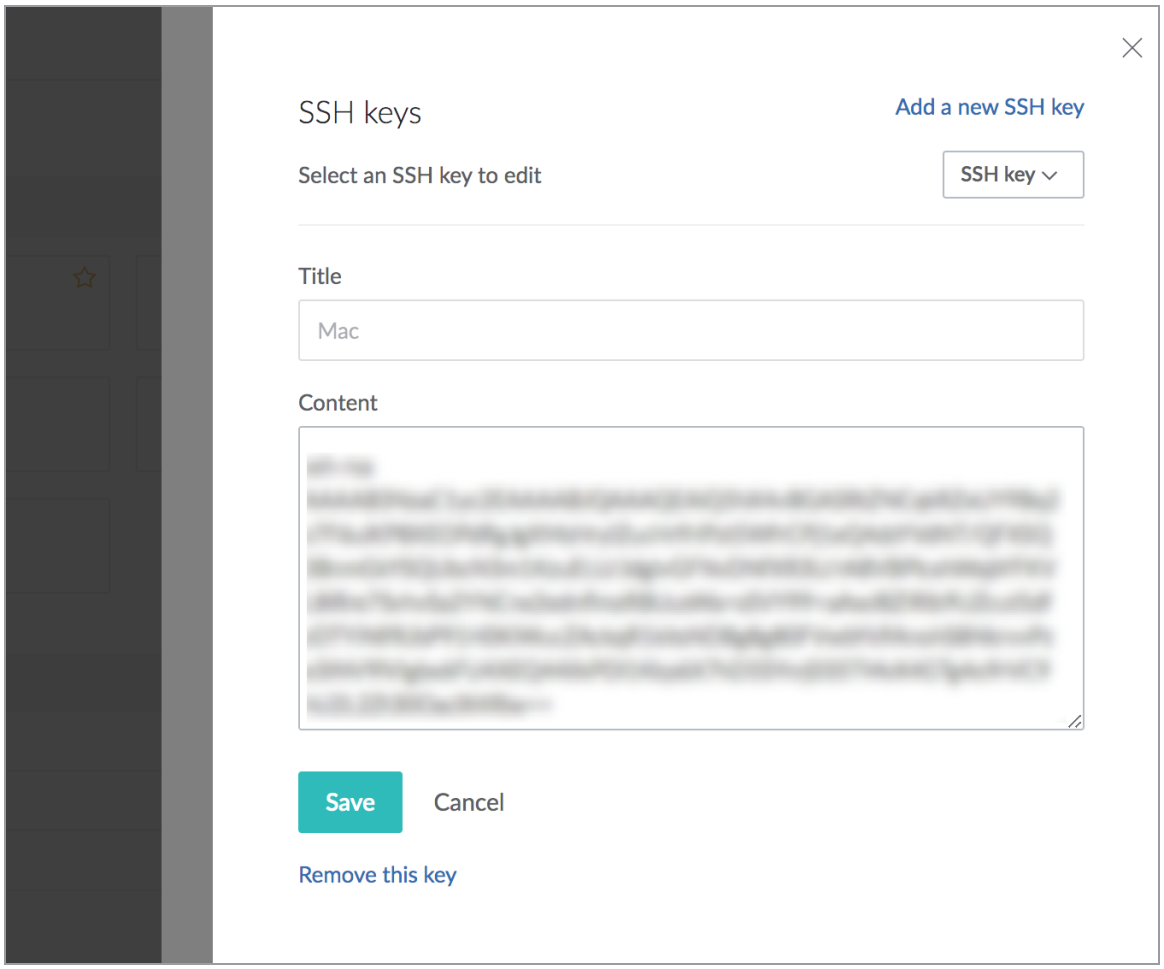
Generate SSH key with Putty

Generate the SSH key using PuttyGen. Use the default settings (SSH-2 RSA, 2048 bits). Save the private key to a directory of your choosing. Copy the **`Public key for pasting into OpenSSH authorized_keys file`** key into clipboard or to a file of your choosing. You need the public key in the next step.



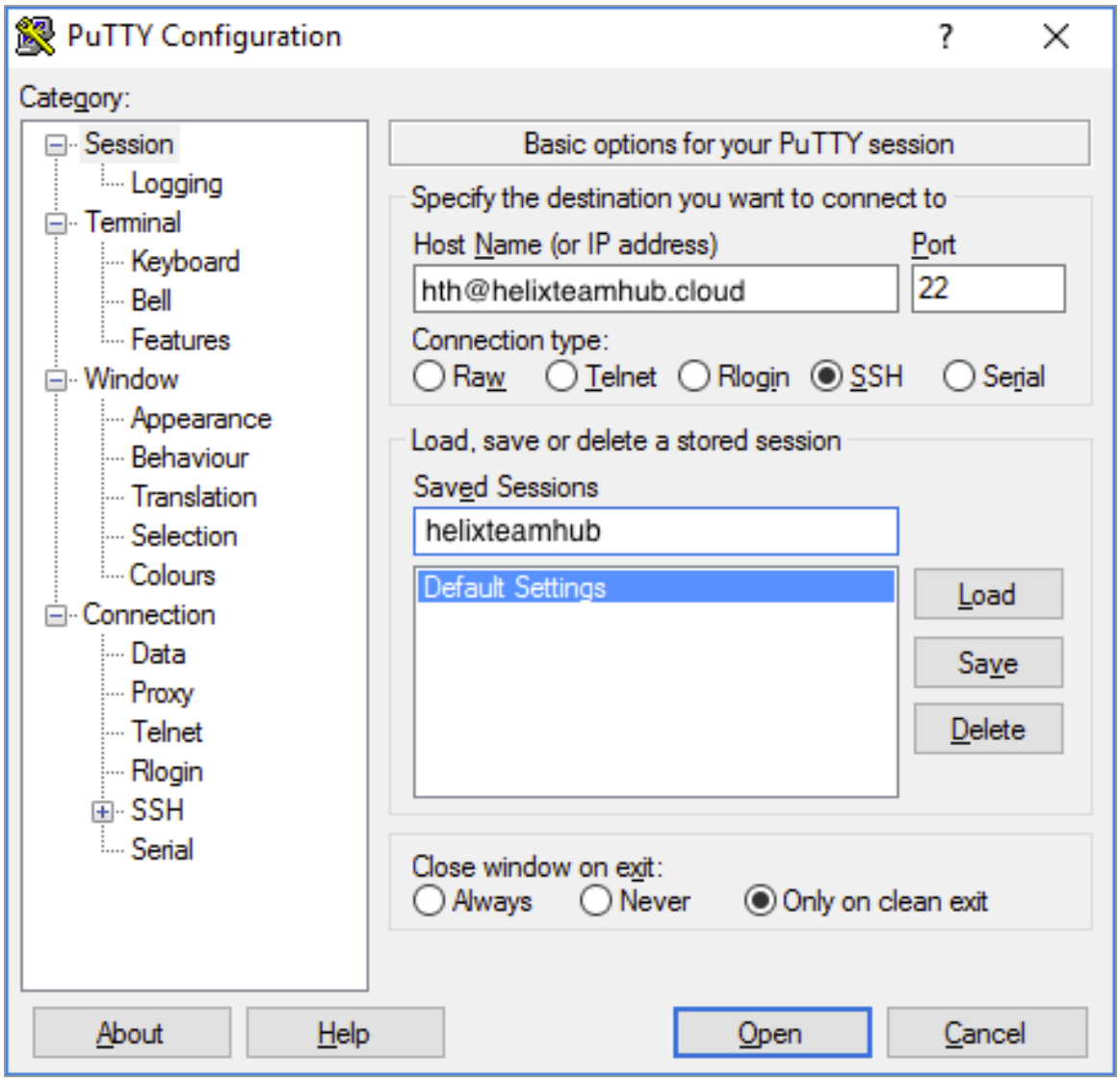
Copy the public key to Helix TeamHub user profile

In Helix TeamHub, SSH keys are managed under user preferences. You may find user preferences by clicking the avatar icon top-right of the screen. Add public key to your preferences with a describing name.



Make a PuTTY Session for the SSH Key

Open Putty from the start menu and copy the `hth@hostname` part of the SVN repository SSH clone URL to the `hostname` field. Omit the `svn+ssh://` prefix and the path suffix. Write a descriptive name to the saved sessions field, and press `save`. If you are using the Helix TeamHub cloud instance for example, the configuration might look like the below.



Next go to the `Connection -> SSH -> Auth` from the left sidebar, and add the private key you saved to `Private key file for authentication`. Go back to the `Session` from the left sidebar, select session you saved earlier and press `save` again.

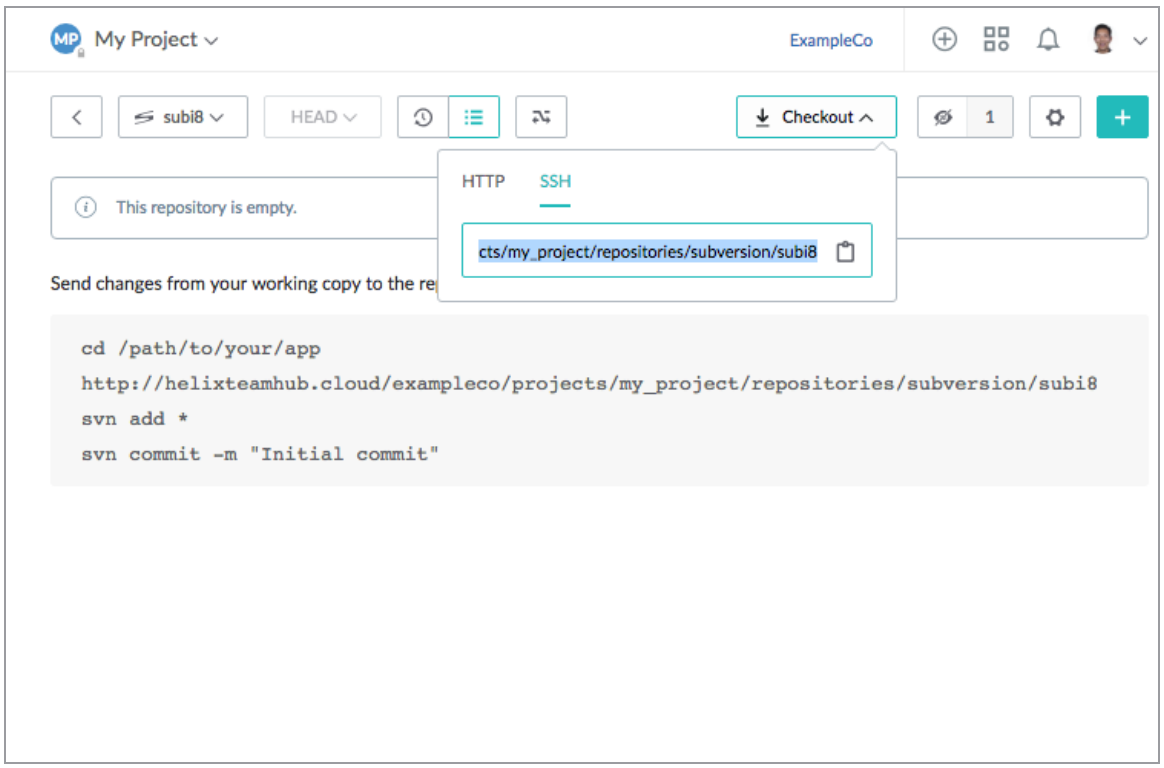
Run Pageant to Activate the SSH Key

Start Pageant from the start menu and access it from the system tray. Add the private key you generated earlier, and click close.

> **Note**
> Pageant needs to be running when you are using TortoiseSVN with SSH protocol.

Checkout the repository

Next you can browse to a directory you wish to checkout your repository. You can copy-paste the clone url as it is directly from the Helix TeamHub web client. With the first checkout using a new SSH key, TortoiseSVN will ask to trust the host key.



Instructions for Migrating a Subversion (SVN) repository to Helix TeamHub

This article walks you through how to migrate an existing Subversion repository to Helix TeamHub.

Prerequisites

In order to migrate your existing Subversion repository to Helix TeamHub by yourself, you need at least Subversion 1.7.X series. If your Subversion repository size is above 2GB or contains more than 5000 commits, contact us or your local IT support to get assistance in your migration.

Get full dump from existing repository

> **Warning**
> If you are using Windows, we recommend that you create the dump file using the native Windows command line interpreter (`cmd.exe`). Creating the dump file using PowerShell can result in data

corruption.

First step is to create a dump file from your existing repository. If you have shell access to the Subversion server, you can use **`svnadmin dump`** as follows:

```
svnadmin dump /path/to/repository > svn-repo-dump.dump
```

where **`/path/to/repository`** is the repository path in the local disk and **`svn-repo-dump.dump`** is the dump file name.

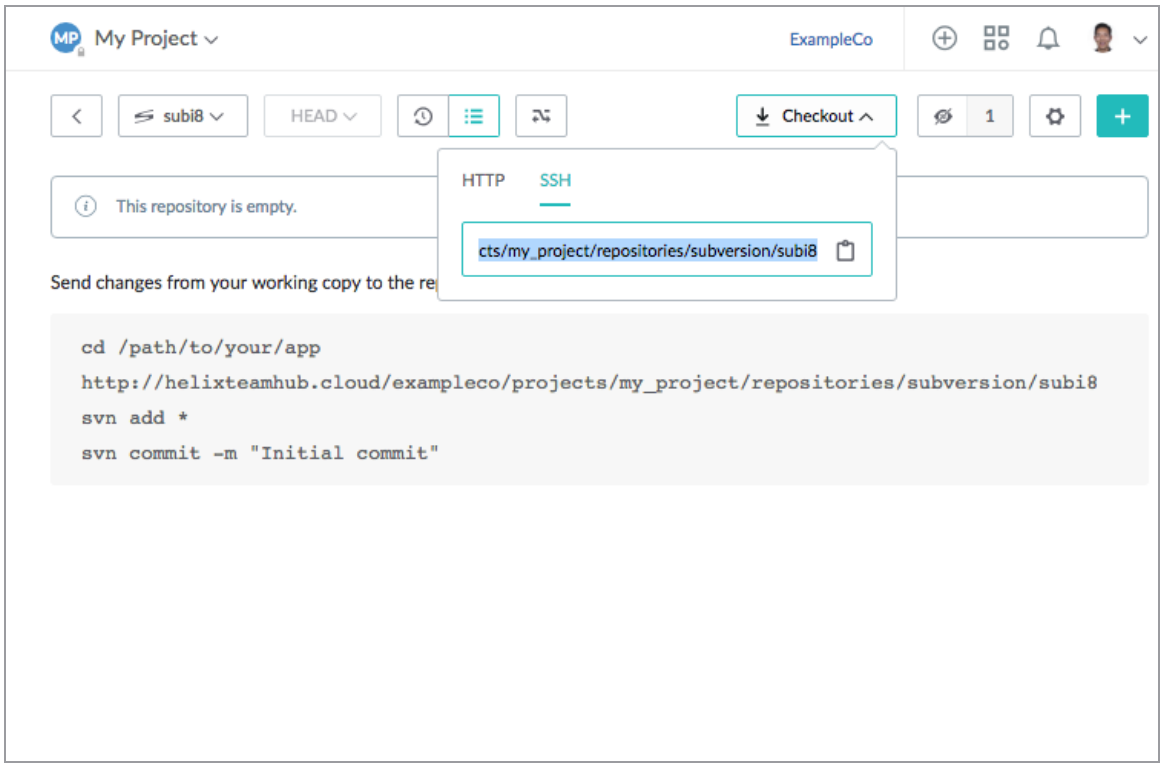
If you do not have shell access to the Subversion server, you need to use **`svnrdump dump`** to dump the repository. This would be accomplished as follows:

```
svnrdump dump <REPOSITORY-CHECKOUT-URL> > svn-repo-dump.dump
```

where **`<REPOSITORY-CHECKOUT-URL>`** is the repository checkout url. You may need to add **`--username <NAME>`** and **`--password <PASSWORD>`** options to **`svnrdump`** command if your Subversion server uses authentication and authorization.

Create new project and repository in Helix TeamHub

1. Login to your account in Helix TeamHub
2. Create a new project or navigate to a existing project
3. Create a new Subversion repository. If you want to keep the original repository UUID, enter the UUID now.
4. Copy the checkout URL from the repository view to clipboard



Do not checkout the repository or commit any changes to it.

Migrate the data

Use **`svnrdump load`** to load the dump file you got from the existing repository to the freshly created Helix TeamHub repository as follows:

```
svnrdump load <HELIX-TEAMHUB-REPOSITORY-CHECKOUT-URL> < svn-repo-dump.dump
```

where **`<HELIX-TEAMHUB-REPOSITORY-CHECKOUT-URL>`** is the repository checkout URL you copied to clipboard and **`svn-repo-dump.dump`** is the dump filename.

If you need assistance in migrating your Subversion repository to Helix TeamHub cloud, contact our support. If you wish to migrate your Subversion repository to Helix TeamHub on-premises installation, contact your local IT support.

Version control with WebDAV

WebDAV (Web-based Distributed Authoring and Versioning) facilitates collaboration between users in editing and managing documents and files. In Helix TeamHub, you can create any number of WebDAV repositories in your projects to share files, or even host static web sites.

Setting up on Windows

Unfortunately, Windows support for WebDAV is poor and it may not always function correctly. We strongly recommend using one of the 3rd party clients, like Cyberduck or WebDAV Client.

Setting up on MacOS

Connect to server from the "Finder" menu and fill in the Helix TeamHub repository URL into "Server Address" field. Click "Connect" and you will find the new "drive" on your desktop.

Setting up on Linux

Select "Open Location..." from the global menu and fill in the Helix TeamHub repository URL in the location field. (You may need to replace the protocol "http(s)://" with "davs://" in the URL). Click "Open" and you will find the new "drive" on your desktop.

For example, to mount the "bar" WebDAV repository from the "foo" project in "acme" company to your desktop, use the following URL:

```
http://[HELIX TEAMHUB
USERNAME]@helixteamhub.cloud/acme/projects/foo/repositories/webdav/bar
```

Version control with Maven

Maven repositories can be used to store build artifacts and other binary files. In Helix TeamHub, you can create any number of Maven repositories in your projects.

The below is the list of resources to help you learn Maven:

- Official Maven site
- Maven in 5 Minutes

Setting up

Setup credentials in your `~/.m2/settings.xml` file. Password can also be encrypted with master password. See the documentation for more details.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>helixteamhub.cloud</id>
      <username>username</username>
```

```
      <password>password</password>
    </server>
  </servers>
</settings>
```

Configure repositories in your project's `pom.xml` file as required for resolving and deploying using the repository url:

```
<repositories>
  <repository>
    <id>helixteamhub.cloud</id>

<url>https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar</
url>
  </repository>
</repositories>

<distributionManagement>
  <repository>
    <id>helixteamhub.cloud</id>

<url>https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar</
url>
  </repository>
  <snapshotRepository>
    <id>helixteamhub.cloud</id>

<url>https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar</
url>
  </snapshotRepository>
</distributionManagement>
```

Importing and managing artifacts

Other existing artifacts can also be imported to Helix TeamHub with Maven Deploy Plugin:

```
cd /path/to/maven/repos

mvn deploy:deploy-file -
```

```
Durl=https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar \
                      -DrepositoryId=helixteamhub.cloud \
                      -DpomFile=junit/junit/3.8.2/junit-3.8.2.pom  \
                      -Dfile=junit/junit/3.8.2/junit-3.8.2.jar
```

It is also possible to access and manage artifacts with `curl`:

Upload file:

```
curl -i -u user:password -T file.jar
https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar/file.j
ar
```

Download file:

```
curl -i -u user:password
https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar/file.j
ar
```

Delete file/directory:

```
curl -i -u user:password -X DELETE
https://helixteamhub.cloud/acme/projects/foo/repositories/maven/bar/file.j
ar
```

Version control with Ivy

Ivy repositories can be used to store build artifacts and other binary files. In Helix TeamHub, you can create any number of Ivy repositories in your projects.

The below is the list of resources to help you learn Ivy:

- Official Ivy site
- Ivy Quick Start

Setting up

Setup credentials in `~/helixteamhub.properties` file:

```
hth.host=helixteamhub.cloud
hth.user=username
hth.password=password
```

Configure credentials and resolvers in your project's `ivysettings.xml` file using the repository url:

```
<ivysettings>
  <properties file="${user.home}/helixteamhub.properties" />
  <credentials host="${hth.host}" realm="Helix TeamHub repository"
username="${hth.user}" passwd="${hth.password}"/>

  <settings defaultResolver="hth"/>
  <resolvers>
    <ibiblio name="hth" m2compatible="true"
root="https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/bar"
/>
  </resolvers>
</ivysettings>
```

Chained resolvers can be used to access multiple repositories:

```
<settings defaultResolver="hth-chain"/>
<resolvers>
  <chain name="hth-chain">
    <ibiblio name="hth" m2compatible="true"
root="https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/bar"
/>
    <ibiblio name="ibiblio" m2compatible="true" />
  </chain>
</resolvers>
```

Url resolver can be used for specifying custom pattern:

```
<url name="hth">
  <ivy
pattern="https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/ba
r/[organisation]/[module]/[revision]/ivy.xml" />
  <artifact
pattern="https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/ba
r/[organisation]/[module]/[revision]/[artifact]-[revision].[ext]" />
</url>
```

Importing and managing artifacts

Ivy install task can be used for importing from another repository. It is also possible to access and manage artifacts with `curl`:

Upload file:

```
curl -i -u user:password -T file.jar
https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/bar/file.jar
```

Download file:

```
curl -i -u user:password
https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/bar/file.jar
```

Delete file/directory:

```
curl -i -u user:password -X DELETE
https://helixteamhub.cloud/acme/projects/foo/repositories/ivy/bar/file.jar
```

Version control with Docker

Docker is a software containerization platform that enables you to introduce a lightweight layer of abstraction of operating-system-level virtualization. Docker isolates features of Linux to allow independent *containers* to run within a single Linux instance. A container image is a stand-alone and executable package that consists of code, runtime, system tools, system libraries and settings.

Helix TeamHub provides the Docker Registry so that you can store and distribute Docker images within your organization.

The following resources may help you learn Docker:

- Official Docker site
- Try Docker interactively in 15 min
- Docker documentation

Setting up

1. Download and install the latest version of the Docker client.
2. Open your *terminal* and run the following command to log in to start communicating with the Helix TeamHub Docker Registry:

   ```
   docker login helixteamhub.cloud
   ```

   Your username consists of 2 parts: the *username* and the *company short name* from TeamHub. These parts must be concatenated by the `+` sign. For example: `admin+hth`
3. Use the TeamHub UI to create a repo. This is required to be able to push images.
4. When you have a repo, tag your image with its address.

   To tag a new build, run:

```
docker build -t helixteamhub.cloud/<COMPANY-SHORT-NAME>/projects/<PROJECT-NAME>/repositories/docker/<REPOSITORY-NAME>
```

If your image already exists, tag it by running:

```
docker tag <IMAGE> helixteamhub.cloud/<COMPANY-SHORT-NAME>/projects/<PROJECT-NAME>/repositories/docker/<REPOSITORY-NAME>
```

By default, your images are tagged as **`latest`**.

Remember, you can use as many tags as you need, as follows:

```
docker tag <IMAGE> helixteamhub.cloud/<COMPANY-SHORT-NAME>/projects/<PROJECT-NAME>/repositories/docker/<REPOSITORY-NAME>:<TAG-NAME>
```

5. To push your image to the Helix TeamHub Docker Registry, run:

```
docker push helixteamhub.cloud/<COMPANY-SHORT-NAME>/projects/<PROJECT-NAME>/repositories/docker/<REPOSITORY-NAME>
```

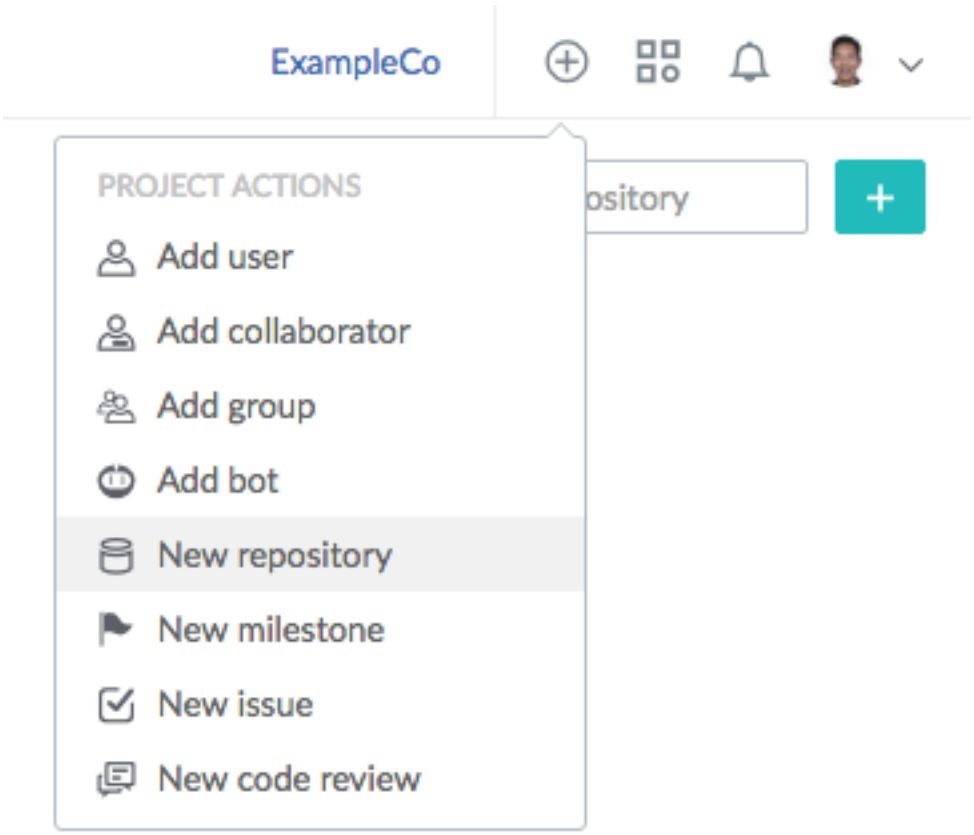
6. To pull an image (only available for authorized users), run:

```
docker pull helixteamhub.cloud/<COMPANY-SHORT-NAME>/projects/<PROJECT-NAME>/repositories/docker/<REPOSITORY-NAME>
```

Migrating existing Docker images to Helix TeamHub

To migrate existing Docker images to Helix TeamHub, perform the following steps:

1. Create a new repository in the TeamHub project, either from the **Repositories** view or the quick actions button:



2. Pull the image you want to migrate:

```
docker pull <OLD-IMAGE>
```

3. Tag your image with an appropriate name (remember to include all tags):

```
docker tag <OLD-IMAGE>:<OLD-TAG-NAME>
helixteamhub.cloud/<COMPANY-SHORT-NAME>/projects/<PROJECT-
NAME>/repositories/docker/<REPOSITORY-NAME>:<TAG-NAME>
```

4. Log in to the TeamHub Docker Registry using **`<HTH-USERNAME>+<COMPANY-SHORT-NAME>`** as your username:

```
docker login helixteamhub.cloud
```

5. Push the image to TeamHub:

```
docker push helixteamhub.cloud/<COMPANY-SHORT-
NAME>/projects/<PROJECT-NAME>/repositories/docker/<REPOSITORY-
NAME>
```

Repository settings

If you have admin access to a repository, you can configure the repository settings. The settings available depend on the repository type.

To access repository settings:

1. Navigate to a project.
2. Navigate to the **Repositories** view.
3. Navigate to the repository you want to configure.
4. Click the gear icon to open the **Repository settings** form.

This form lets you configure settings for branches, code reviews, and maintenance.

> **Note**
> With Helix authentication, certain restrictions apply to TeamHub functionality. For details, see "Limitations with Helix authentication" on page 11.

Branch settings

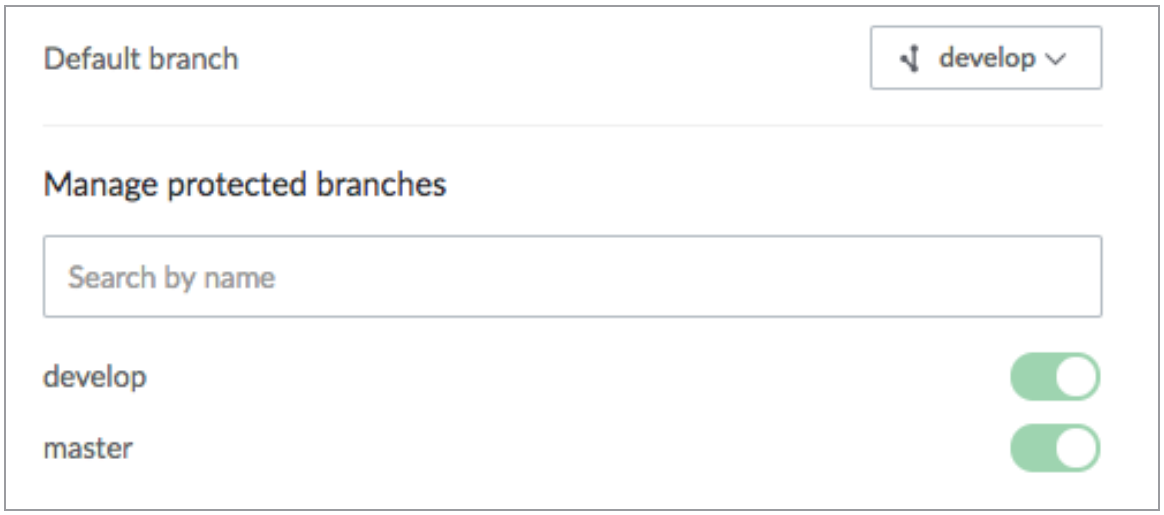
The **Branches** tab includes the following settings:

- **Default branch:** Sets the default branch shown in the code browser. For Git repositories, it also controls which branch gets checked out locally when you clone the repository.
- **Manage protected branches:** Lists branches that have been configured as protected. Manage branch protection by finding a branch by name and clicking the protected/unprotected toggle.

  > **Note**
  > Helix TeamHub supports managing protected branches for Git and Helix Git repositories.

  Protecting a branch has the following effects:

  - TeamHub rejects all force pushes to the branch.
  - TeamHub prohibits deletion of the branch.
  - Only users with admin or master role have permission to push changes to the branch.



Code Review settings

The **Code Reviews** tab includes the following settings:

- **Default base branch:** Controls the default destination branch to use when creating code reviews.
- **Default merge method:** Defines the default behavior when merging a code review, as follows:
  - **Commit and merge:** Creates a merge commit to retain the full history of changes but makes the history more verbose and complex.
  - **Rebase and merge:** Rebases the feature branch on top of the destination branch and then does a fast-forward merge to the destination branch, thus avoiding the creation of an explicit merge commit.
  - **Squash and merge:** Combines a set of commits into a single commit to the destination branch.
- **Delete head reference on merge by default:** Specifies whether to delete or keep the feature branch after merging the code review by default.
- **Allow administrators to force merge:** Project and repository admins are allowed to force merge a code review when requirements are not met. For details, see "Force merging" on page 92.
- **Default reviewers:** Specifies individual reviewers or groups that are automatically assigned to code reviews created for the repository.

  Adding a group instead of individual reviewers is beneficial, for example, when any one person in the group is sufficient to fulfill the review requirements.

  To add default reviewers or groups:

  1. Click **Manage**.
  2. In the **Manage default reviewers** form, do any of the following:

- On the **Project Members** tab, in the search field, enter the name of the reviewer you want to add, or select the reviewer from the list.
- On the **Project Groups** tab, in the search field, enter the name of the group you want to add, or select the group from the list.

3. Click **Update**.

To remove default reviewers or groups:

1. Click **Manage**.
2. In the **Manage default reviewers** form, do any of the following:
   - On the **Project Members** tab, click the name of the reviewer you want to remove.
   - On the **Project Groups** tab, click the name of the group you want to remove.
3. Click **Update**.

- **Reviewers may be set by admin or master roles only:** Defines whether managing reviewers is limited only to *admin* and *master* roles.
- **Approvals:** Defines the default number of approvals required for a code review before it can be merged.
- **Enforce approvals:** Defines whether to enforce approvals for all the code reviews of the repository.
- **Reset approvals when new changes are submitted:** Defines whether reviewers need to approve new changes submitted after approval.
- **Enforce approval reset:** Defines whether to enforce approval reset for all code reviews in the repository.
- **Require successful build:** Defines whether a successful build is required by default for a code review before it can be merged.
- **Enforce successful build requirement:** Defines whether to enforce the build configuration for all code reviews of the repository.
- **Require all tasks to be resolved:** Defines whether to resolve all task comments by default before a code review can be merged.
- **Enforce tasks requirement:** Defines whether to enforce the configuration of task comments for all code reviews of the repository.
- **Automatic code review merge:** Defines whether to merge a code review automatically when the configured criteria are met (such as build successful, code review approved, task comments resolved). Only available with code reviews that are not part of a multi-repo code review.
- **Enforce automatic code review merge:** Defines whether to enforce the configuration of automatic merge for all code reviews of the repository. Only available with code reviews that are not part of a multi-repo code review.

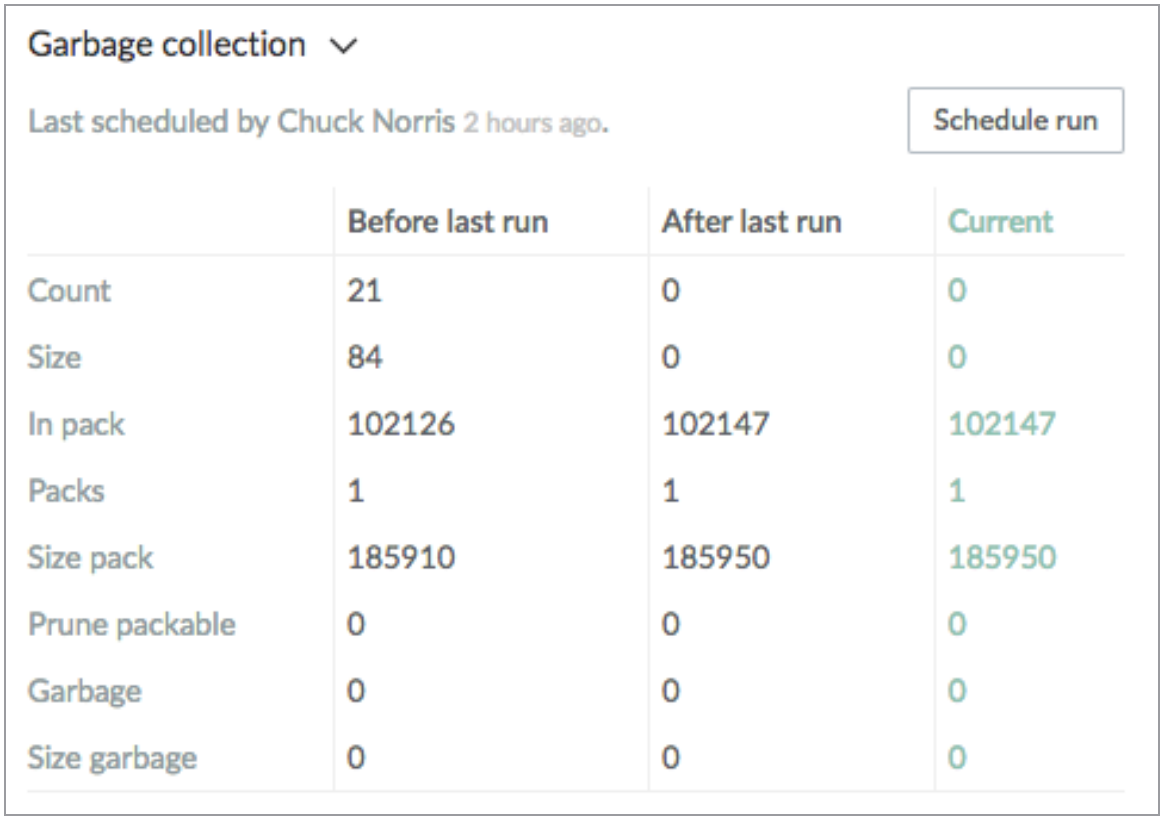
Maintenance settings

The **Maintenance** tab includes the following:

Garbage collection

Garbage collection: Lets you run garbage collection to clean up unused references from the repository and optimize its size and performance. To run garbage collection, click **Schedule run**. When garbage collection is complete, this tab displays the statistics.

> **Note**
> The **Garbage collection** option is only available for native Git repositories.



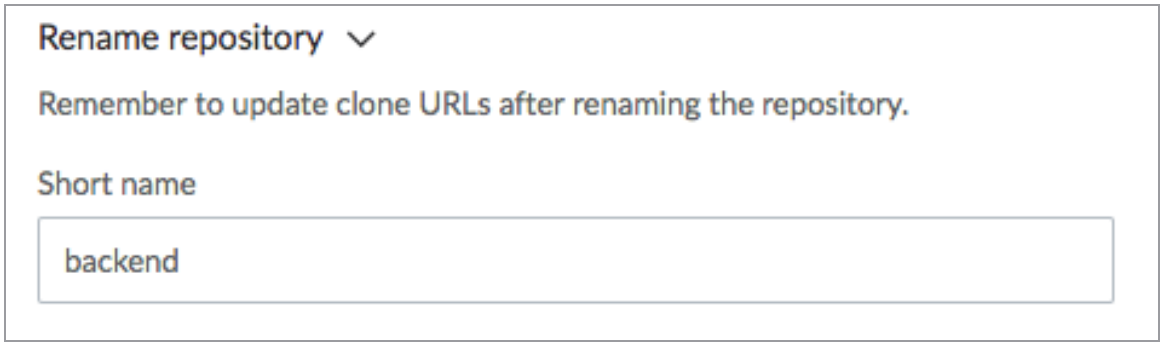
	Before last run	After last run	Current
Count	21	0	0
Size	84	0	0
In pack	102126	102147	102147
Packs	1	1	1
Size pack	185910	185950	185950
Prune packable	0	0	0
Garbage	0	0	0
Size garbage	0	0	0

Rename repository

Rename repository: Lets you rename a repository after it has been created.

> **Note**
> Renaming a repository also affects clone URLs. Make sure to update the configuration of existing

clones with the changed URL.



Set custom UUID

Set custom UUID: Lets you set a custom UUID for a repository after it has been created.

Note
The **Set custom UUID** option is only available for Subversion repositories.

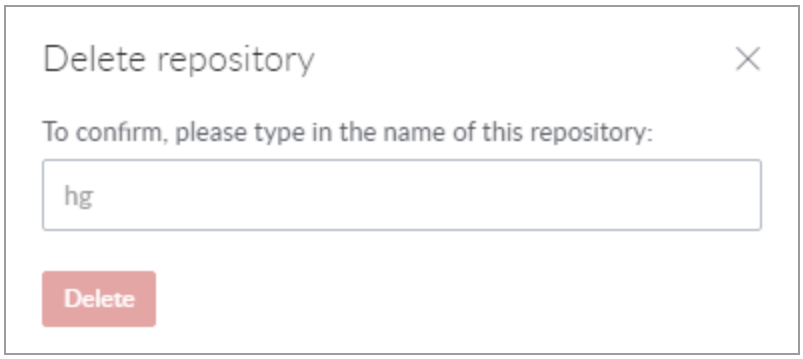
Delete repository

Delete repository: Lets you delete the repository.

When you delete a repository:

- The repository is deleted from Helix TeamHub.
- Code reviews associated with the repository are deleted.
- Helix Git repository data is not deleted from the Helix server.

When requested, confirm the delete by entering the repository name and clicking **Delete**.



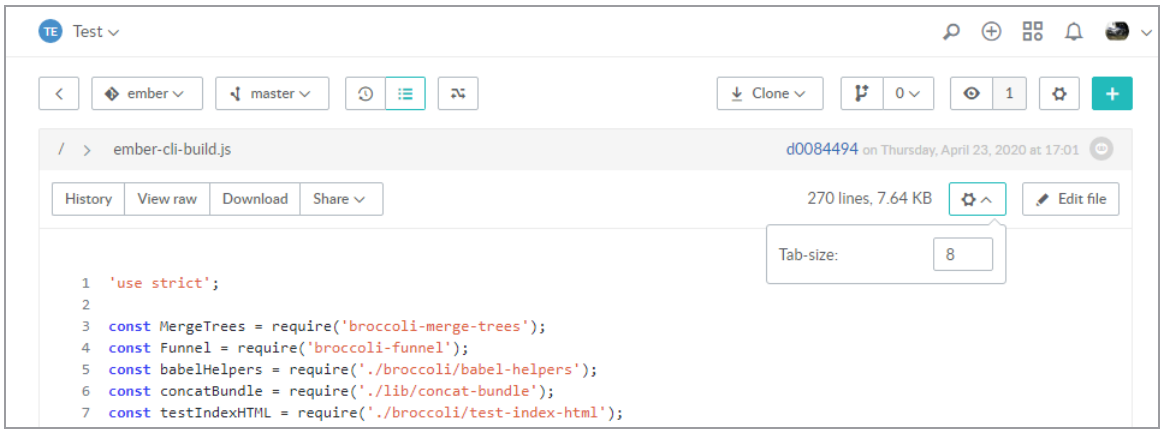
Code browser views

> **Note**
> Helix TeamHub supports editing and committing from the UI for Mercurial, WebDAV, Git, and Helix Git repositories.

TeamHub provides a browser view of code stored in a particular repository. The browser view presents the code in a more accessible, readable form. It also lets you edit the code and commit your changes to the repository.

View code

- Click a file name.

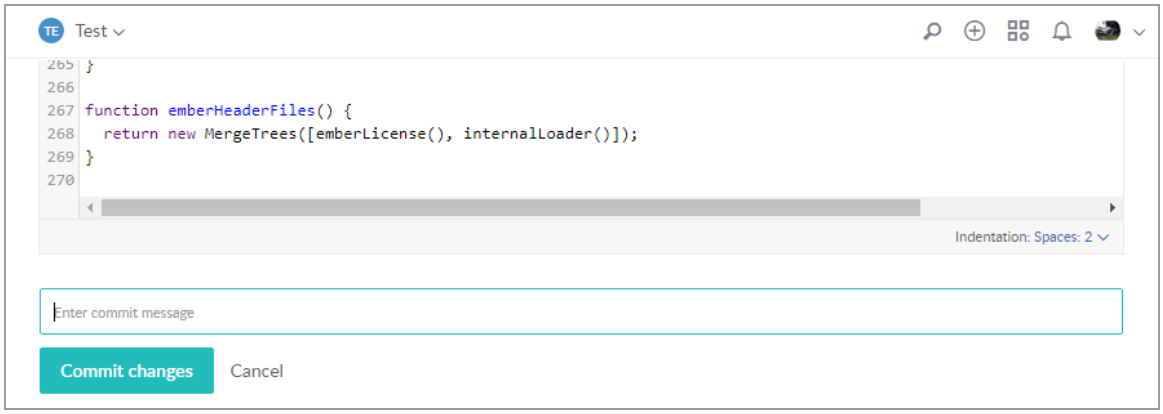


The number of spaces used to represent a tab can be set between 1 and 8. To change the number of spaces used, set **Tab-size** from the gear icon above the file (not available for Microsoft Internet Explorer).

Edit code

1. In the top right, click **Edit file**. The view switches to edit mode.

2. Make any required changes.
3. **Optional:** Enter a comment to be submitted with the commit.
4. Click **Commit changes** at the bottom of the code page to save your changes and submit them to the repository.



TeamHub may encounter difficulties displaying file content in browser view if the file:

- Is empty
- Is too long/big
- Contains unreadable content

In these cases, TeamHub displays the following message:

```
This file is empty or its contents cannot be displayed. Click here
to download the file.
```

On the **Raw** tab, you can always view file content in plain code view.

```
#! /usr/bin/env ruby

# This script takes two arguments:
# 1. The name of an export - ex. EJSON
# 2. The name of a package - ex. ejson
# It makes sure that if the export appears somewhere in package source code, the
# name of the package appears somewhere in package.js for that package.

root = File.join(File.dirname(__FILE__), "..", "..");

Dir.chdir(root)

file_list = `git grep -lw '#{ARGV[0]}' packages/`.lines
package_list = file_list.map do |filename|
  filename.split("/")[1]
end

package_list = package_list.uniq

package_list.each do |p|
  unless File.open("packages/#{p}/package.js").read.include? ARGV[1]
    puts "'#{ARGV[0]}' appears in #{p} but '#{ARGV[1]}' not in package.js. Files:"
    puts `git grep '#{ARGV[0]}' packages/#{p}/`
    puts ""
  end
end
```

Tabs

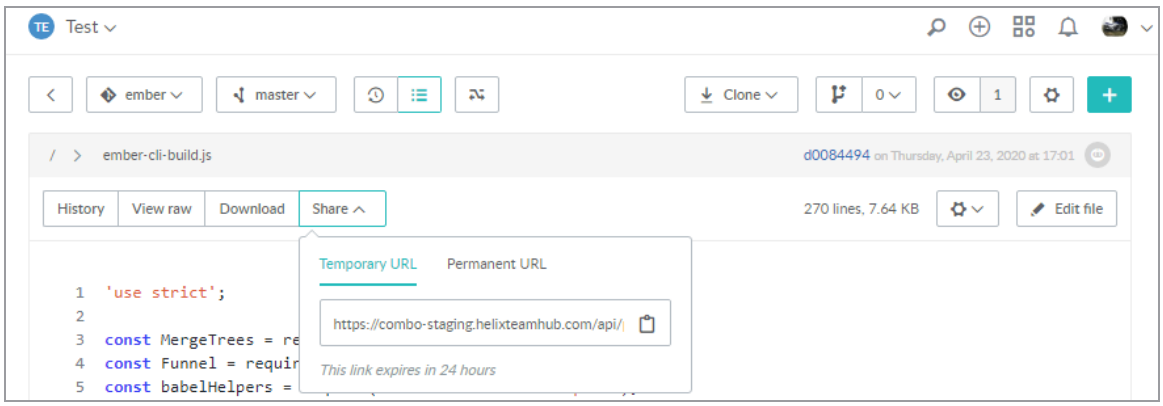
- The **History** tab provides the complete commit history for the file.

- The **View raw** tab displays the file content in plain code view.

```
#! /usr/bin/env ruby

# This script takes two arguments:
# 1. The name of an export - ex. EJSON
# 2. The name of a package - ex. ejson
# It makes sure that if the export appears somewhere in package source code, the
# name of the package appears somewhere in package.js for that package.

root = File.join(File.dirname(__FILE__), "..", "..");

Dir.chdir(root)

file_list = `git grep -lw '#{ARGV[0]}' packages/`.lines
package_list = file_list.map do |filename|
  filename.split("/")[1]
end

package_list = package_list.uniq

package_list.each do |p|
  unless File.open("packages/#{p}/package.js").read.include? ARGV[1]
    puts "'#{ARGV[0]}' appears in #{p} but '#{ARGV[1]}' not in package.js. Files:"
    puts `git grep '#{ARGV[0]}' packages/#{p}/`
    puts ""
  end
end
```

- Click the **Download** tab to download the file.
- The **Share** option lets you copy a **Temporary URL** that expires after 24 hours or a **Permanent URL** that you can share with others.

> **Tip**
> File sharing must be enabled for the company, see "Feature settings" on page 169.



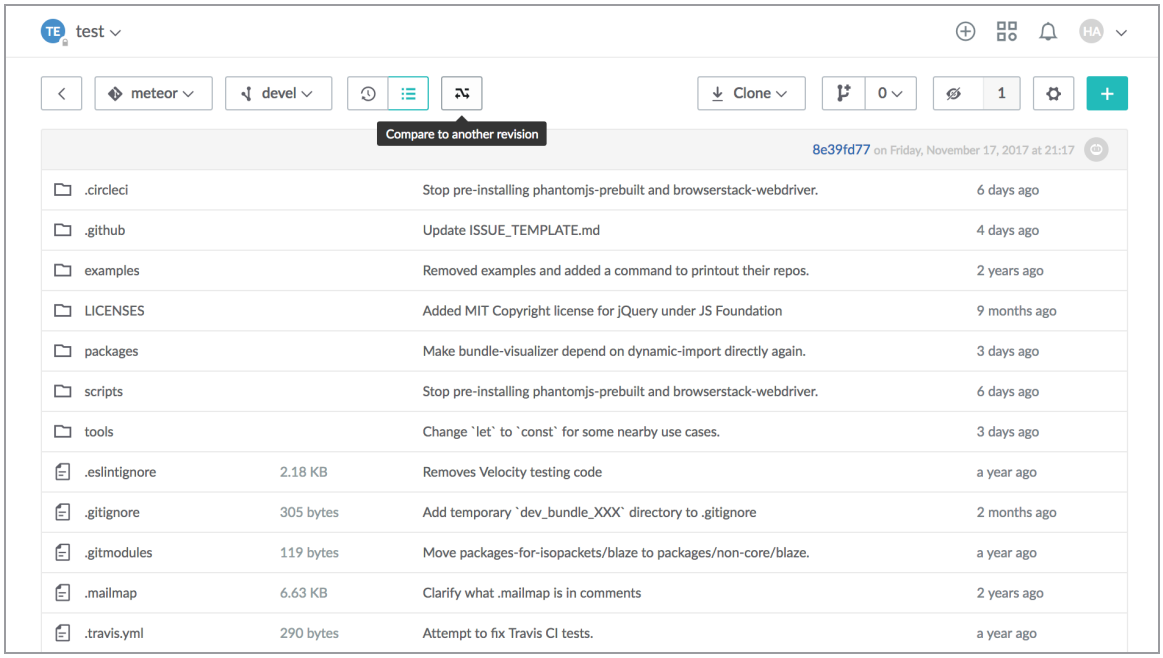
Compare view

> **Note**
> Helix TeamHub supports compare view for Mercurial, Subversion, Git, and Helix Git repositories.

Helix TeamHub lets you compare two revisions of a branch. By default, TeamHub compares a revision to the default base branch. If a difference between revisions is significant, TeamHub displays the following message to indicate that code changes are hidden: `This diff is too large to display and has been truncated. You may view the entire diff locally.`

To compare two revisions:

1. Select the revision that you want to compare and click the **Compare to another revision** button, as shown in the following figure.

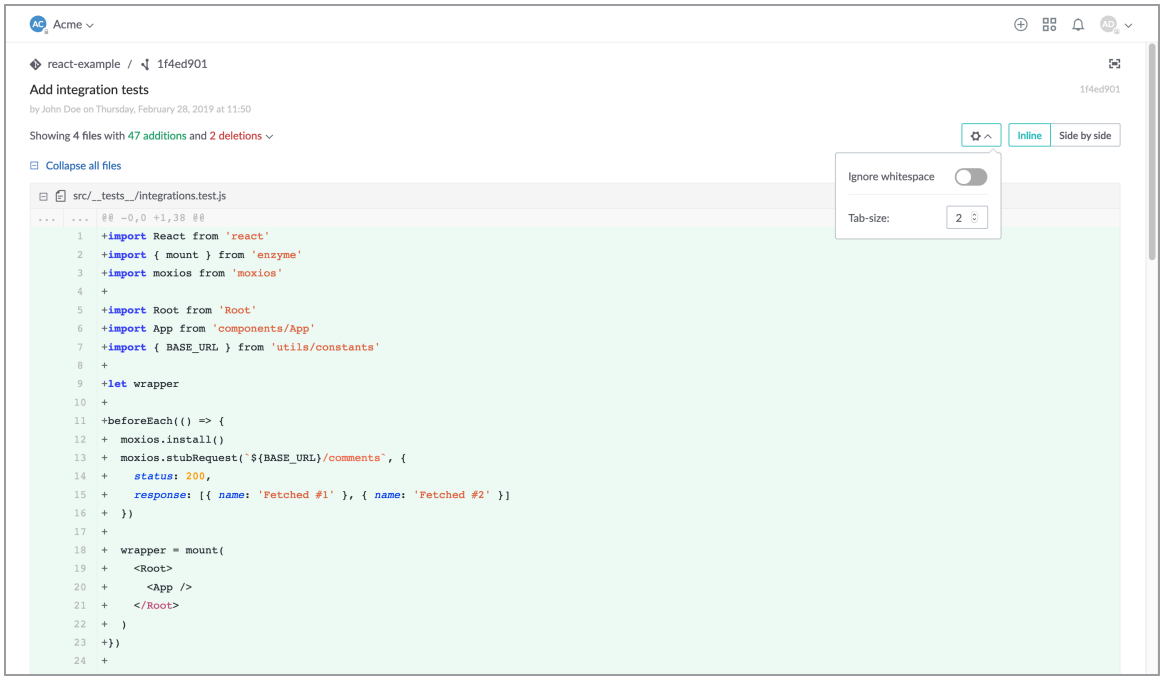


2. If you prefer to compare to a base other than the default base branch, select a different base from the **compare** list.
3. Click the plus icon next to a file name to expand the changes, or click **Expand all** at the top of the list to expand all changes at once.

   > **Note**
   > This operation may be slow for larger changesets.

   By default:

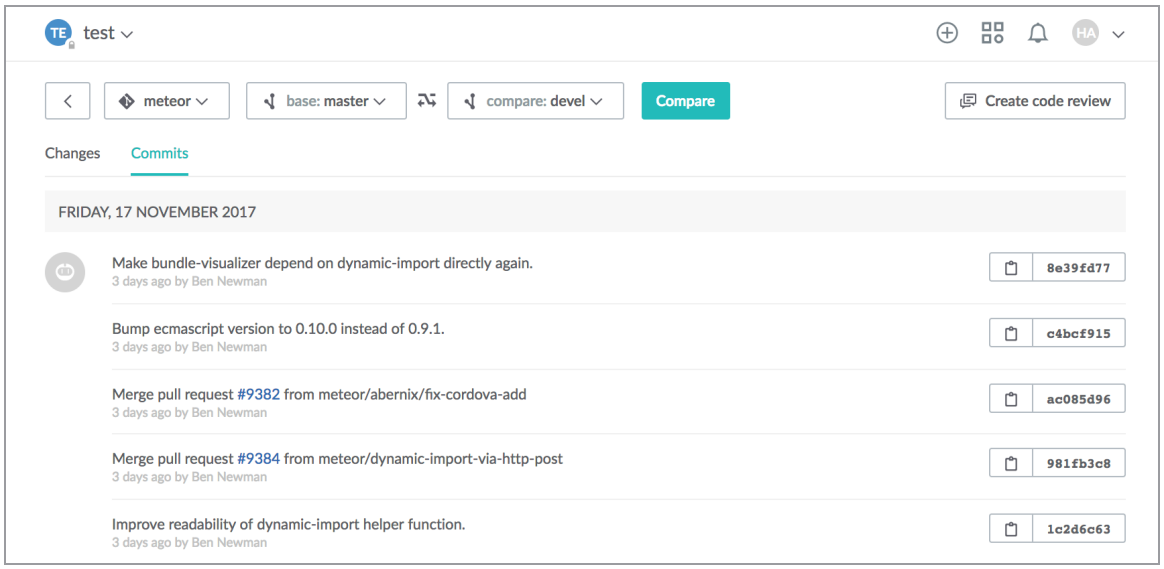
- **Inline** view shows diffs inline. If you prefer to view changes side by side, click **Side by side** above the list.



- Whitespace changes are displayed. To hide whitespace changes, switch on **Ignore whitespace** from the gear icon above the list.
- The number of spaces used to represent a tab can be set between 1 and 8. To change the number of spaces used, set **Tab-size** from the gear icon above the list (not available for Microsoft Internet Explorer).

Commits

In addition to viewing raw changes between revisions on the **Changes** tab, you can see commit difference on the **Commits** tab.



Branching

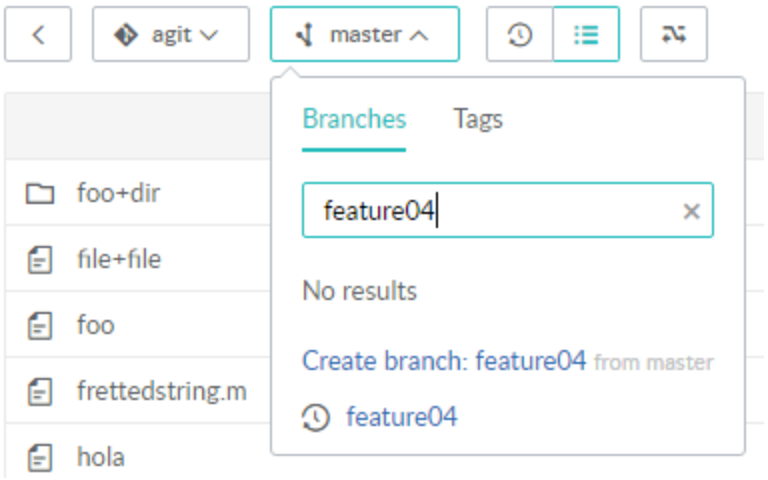
> **Note**
> Helix TeamHub supports branching from the UI for Git and Helix Git repositories.

Branching refers to creating a branch from a repository. Branches remain part of the original repository and allow you to make changes that can be merged into the repository when the work is complete. Branching is only available for supported repositories.

For information on how to merge your changes into the parent repository, see the "Merge options" on page 91 section under **Code Reviews**.

To create a branch:

1. Navigate to the project.
2. Navigate to the repository you would like to add a branch to.
3. In the repository tree view, click the **Branch** dropdown. The dropdown lists any existing branches.
4. Enter a unique name for your new branch in the **Search** box. TeamHub displays **No results** and offers to create the new branch for you.



5. Click the **Create branch <*branchname*>** text. TeamHub displays the newly created branch.

   Now you can now start working in the new branch.

Forking

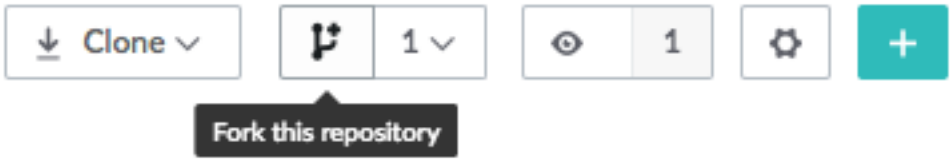
> **Note**
> - Helix TeamHub supports forking for Mercurial and native Git repositories.
> - Creating a fork requires read privileges to the parent repository and the privileges to create a repository in the target project. For more information, please see the project & repository roles section.

Forking refers to creating a copy of a repository for your own use. Forks allow you to do independent development because any changes you make to the fork do not affect the original (parent) repository. The **Fork** icon is only available for supported repositories.

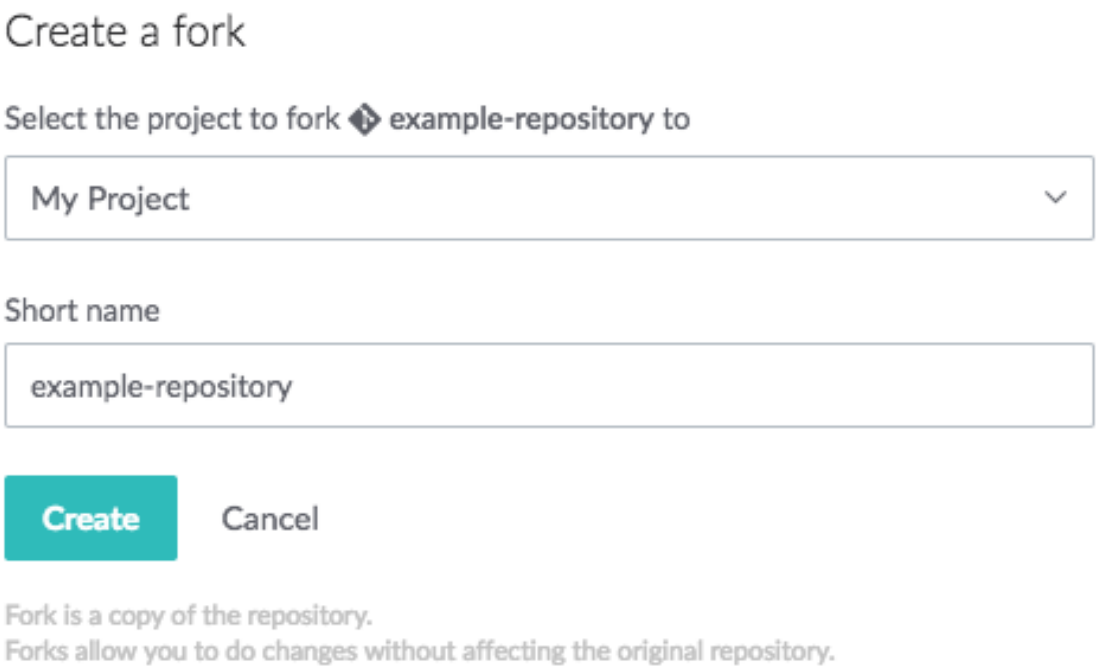
For information on how to merge your changes back into the parent repository, see the pull requests section under **Code Reviews**.

To create a fork:

1. Navigate to the project.
2. Navigate to the repository you would like to fork.
3. In the repository tree view, click the **Fork** icon. The dropdown list next to the button lists any existing forks.



4. In the **Create a fork** form, select the project into which you would like to fork the repository.
5. (Optional) Specify a different name for the fork. By default, the fork has the name of the parent repository.



6. Click **Create**. TeamHub displays the newly created repository.

   Now you can clone the repository and start making changes.

Milestones

Milestones indicate significant achievements in a project. Each milestone can contain multiple issues. The default states are **Open**, **WIP** (work in progress), or **Closed**.

You can prioritize the importance of the issues in a milestone. The default priorities are **Low**, **Medium**, or **High**.

Helix TeamHub lets you configure states and priorities with custom values.

Add or edit a milestone

1. At the project scope, in the left pane, click **Milestones**.
2. In the **Milestones** view, do one of the following:
   - To add a milestone, click the plus icon  .
   - To edit a milestone, move your pointer over the milestone to display the gear icon on the right. Then click .
3. In the milestone form, for a new milestone, enter the title of your milestone.
4. Optionally, provide the following information:
   - A short name
   - A description

   By default, TeamHub uses the title as the short name.
5. Optionally, modify the **States**, **Done states**, and **Priorities** fields as needed.

   You can delete any of the default values and type new values. If you enter a new **Done state** that is not yet available in the **States** field, TeamHub adds it automatically.
6. Click **Save**.

Delete a milestone

1. Move your pointer over the milestone you want to delete to display the gear icon on the right. Then click .
2. At the bottom of the milestone form, click **Delete this milestone**.
3. When prompted for confirmation, click **Yes**.

   TeamHub removes the milestone.

Issue tracking

Issues are associated with milestones. Before you can create an issue, you need to create at least one milestone.

The **Issues** view provides a list view of all issues. You can:

- Filter this view by milestone, state, priority, label, due date, or assignee.
- Switch between a list view and a card view that shows progress similar to a Kanban board.
- Add an issue.
- Click the title of an existing issue to view and edit issue details.

Add an issue

1. At the project scope, in the left pane, click **Issues**.
2. In the **Issues** view, click the plus icon  .
3. In the **New issue** form, do the following:
   - **Title:** Provide a descriptive title.
   - **Description:** Optional, enter more details that describe the issue.
   - **Milestone:** Select the milestone with which you want to associate the issue.
   - **State:** Select the initial state of the issue. By default, this is **OPEN**.
   - **Priority:** Select the priority level. By default, this is **LOW**.
4. **Optional:** assign one or more users to the issue by selecting them from the **Assignees** drop down. Click **Assign to me** to assign the issue to yourself. To delete an assignee from the issue, click the **x** next their name.
5. **Optional:** do one of the following to apply labels to the issue:
   - If the project has labels associated, click **Select labels** and then select the label you want to apply.
   - If the project does not have any labels associated, click **Create new labels** and proceed with *step a* below.

     If you want to add additional labels, click **Manage labels** and proceed with *step a* below.

     > **Note**
     > Only users with *admin* or *manager* roles can create and manage labels.

     a. In the **Manage labels** form, enter a label name. If the label exists, TeamHub displays it below the field. If the label does not exist, proceed to the next step.

b. Click **Add <label name> as a new label**. The new label appears with a colored background below the search field.

c. To modify the label's background color, click the drop-down arrow next to the label name and select a different color.

d. To close the **Manage labels** form, click the **x** at the top right or click anywhere in the window, outside of the form.

6. **Optional:** select a due date for the issue from the **Due date** drop down. If required, specify a time on the selected due date. The default time is 12:00 am on the selected date.
7. **Optional:** to associate the issue with an existing branch or a to create a new Git branch for the issue:
   a. Click **Link branches to this issue**.
   b. In the **Linked branches** form, from the **Repository** list, select a repository.
   c. From the **Branches list**, do one of the following to link a branch to the issue:
      - To link to an existing branch, select the branch from the list.
      - To link to a new Git branch, search for a branch name that doesn't already exist in the repo and click **Create branch** when prompted.
   d. Click **Save**.
8. Click **Create** or, to add more issues, click the drop-down arrow and select **Create and add another issue**.

Edit an issue

You can edit an issue to add comments and labels, modify the description, due date, or linked branches, change the assignee, add an attachment, or change the state, priority, or milestone.

To add a comment:

- On the **Discussion** tab, enter a comment in the text box and click **Comment**.

  Comments support markdown formatting.

To apply or add a label:

1. In the right pane, next to **Labels**, click the pencil icon .
2. Do one of the following:
   - If the project already includes the label you want to apply, click the label name to assign it to the issue. Then click the green check mark to save the selection.
   - To add a new label:

> **Note**
> Only users with *admin* or *manager* roles can create and manage labels.

a. Click **Manage labels**.
b. In the **Manage labels** form, enter a label name.
c. Click **Add <*label name*> as new label**.
d. To close the form, click the x icon at the top right or anywhere outside of the form.

   The label you created is now selected.
e. Click the green check mark to save this selection.

To assign an issue to yourself:

- In the right pane, move the pointer over **Assignee** and click **Assign to me**.

To assign an issue to someone else:

1. In the right pane, next to **Assignee**, click the assignee icon .
2. In the **Manage assignees** form, click the name of an assignee or enter the assignee name in the search field.
3. Click **Save**.

To delete an assignee from the issue:

1. Click the **x** next the name of the assignee you want to delete from the issue.
2. Click **Save**.

To specify or change the due date:

1. In the right pane, next to **Due date**, click the pencil icon .
2. In the **Set due date** form, select a date.
3. **Optional:** specify a time. The default time is 12:00 am on the selected date.
4. Click **Save**.

To specify a linked branch:

1. In the right pane, next to **Linked branches**, the pencil icon .
2. In the **Linked branches** form, select a repository and a branch.
3. Click **Save**.

To add an attachment:

1. Click the paperclip icon in the top right.
2. In the **Open** dialog, browse to the file you want to attach and click **Open**.

To change the priority, state, or milestone:

- From the respective drop-down list at the top, select a different priority, state, or milestone.

Code Reviews

A code review (CR) 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.

Helix TeamHub supports code reviews for Mercurial, Git, and Helix Git repositories. With Helix authentication, TeamHub also supports code reviews that span multiple repositories in the same project, referred to as multi-repo code reviews (MRCR).

This section includes the following information:

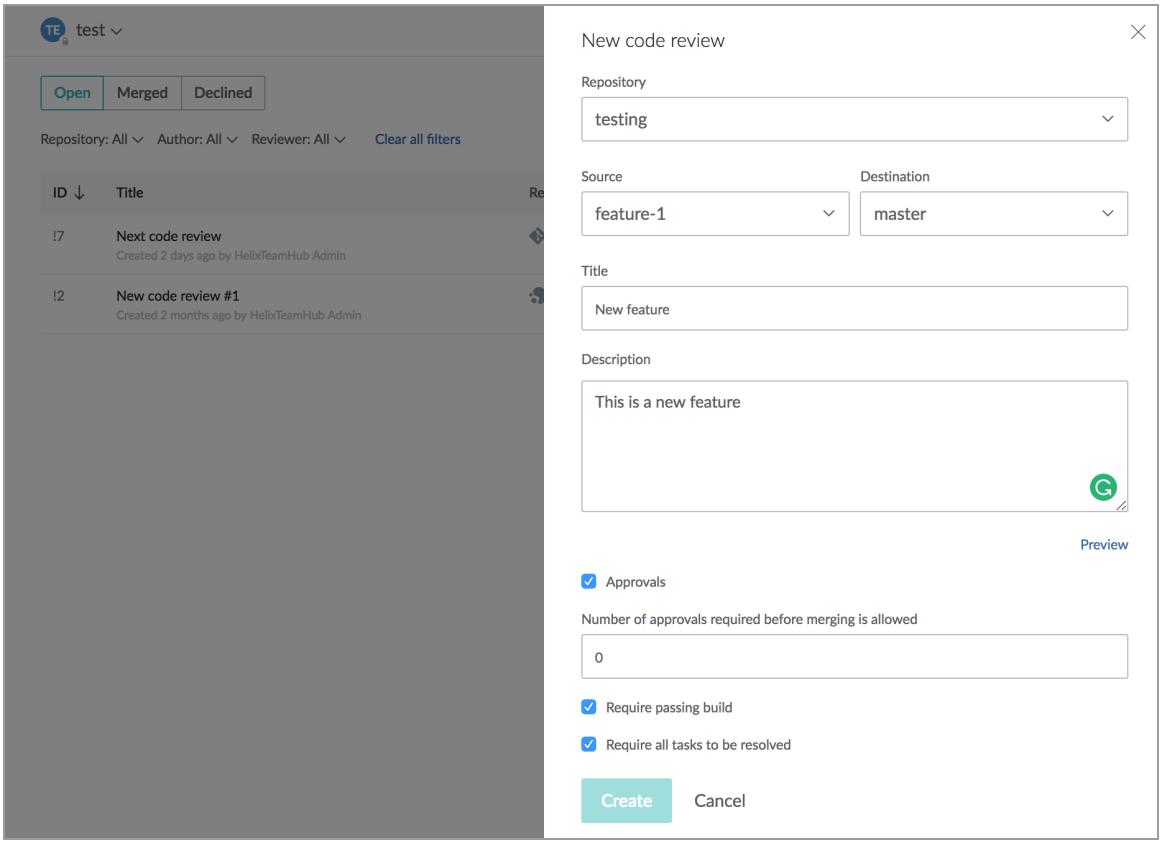
Creating a CR inside a repository

> **Note**
> Helix TeamHub supports code reviews for Mercurial, Git, and Helix Git repositories.

When you have a feature branch, you might want to review it before incorporating it into the base branch. To achieve this, you can create a code review.

To create a new code review:

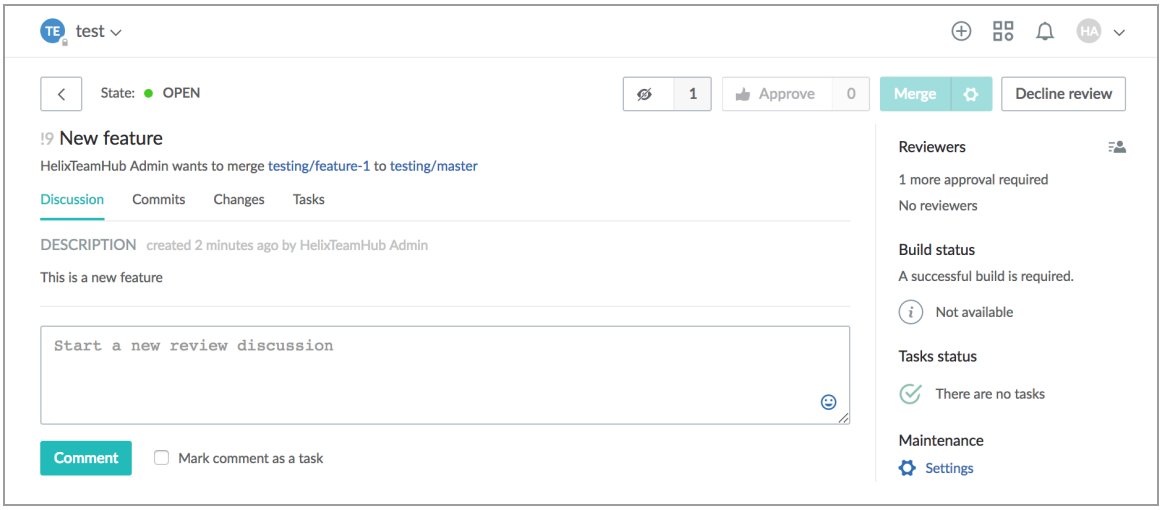
1. In the **Code Reviews** view, on the **Code Reviews** tab, click the plus icon  to the right of the **Search** field to create a new code review.
2. In the **New code review** form, select the repository that contains your feature branch.



3. Select the source (your feature branch) and destination (the base branch).
4. Enter a title for your code review.
5. (Optional) Enter a description for your code review.
6. (Optional) Select any of the following:

- **Approvals** to specify the **Number of approvals required before merging is allowed**. This prevents merging if the code does not receive the specified number of approvals.
- **Automatic merge** to merge the code review automatically when the configured criteria are met (such as build successful, code review approved, and task comments resolved).
- **Reset approvals when new changes are submitted** to require reviewers to approve new changes submitted after approval.
- **Require passing build** to prevent merging if there is no passing head build.
- **Require all tasks to be resolved** to prevent merging unless all tasks created during the review process have been resolved.

7. Click **Create**.



You can modify the settings of your code review at any time.

Creating a multi-repo code review

Note

- Multi-repo code reviews require Helix server 2018.1 and later.
- Helix TeamHub only supports Multi-repo code reviews for Helix Git repositories.

With Helix authentication, Helix TeamHub supports combining changes done in several repositories into a single code review, referred to as Multi-Repo Code Review (MRCR). All repos must reside in the same project. Multi-repo code reviews are helpful when you need to make sure that code changes in different repositories get merged at the same time. For example:

- You develop different parts of a feature in a feature-branch based workflow. All related code changes, made in different repositories, depend on each other. By combining them into an MRCR,

you can merge them from the development branches back to the corresponding master branches at once.

- You start developing the frontend part of a feature in one repository and create a single code review. You then realize that you also need to implement backend changes in a different repository to make things work. Merging both changes separately would put the system temporarily into an inconsistent state. Therefore, you want to merge atomically. You can create a new MRCR and loop in the existing single code review.
- You already have an MRCR in place for a complex feature. Last minute, you realize that you should merge related documentation changes at the same time as the feature code changes. You edit the existing MRCR to tie in the review for the feature documentation.

> **Note**
> TeamHub respects the merge settings for individual code reviews before it allows merging a multi-repo code review. For information on modifying default code review settings, such as approvals, passing builds, and task requirements, see "Merge options" on the facing page and "Code Review settings" on page 69.

To create a new Multi-Repo Code Review:

1. In the **Code Reviews** view, on the **Multi-Repo Code Reviews** tab, click the plus icon  to the right of the **Search** field to create a new multi-repo code review.
2. In the **New Multi-Repo Code Review** form, select the following for each set of code changes that you want to include:
   - The repository that contains your feature branch
   - The source (your feature branch)
   - The destination (the base branch)

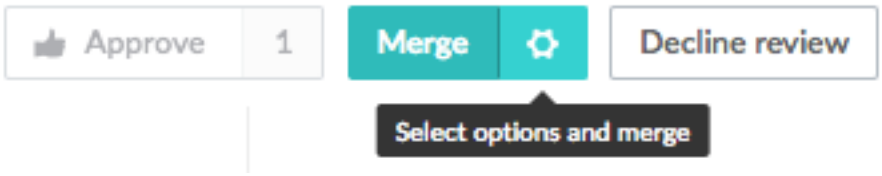
   You can use any combination of branches, with the restriction that you cannot select the same destination branch twice for the same repository. For example, you cannot merge from branch A to master and branch B to master on the same repository.
3. If required, click **Add another repository** and repeat step 2.
4. To add an existing code review:
   a. Click **Add existing code review**.
   b. Select the required code review from the list.

   > **Tip**
   > You can limit the list of existing code reviews by entering a name in the search field.
5. Enter a title for your code review.
6. (Optional) Enter a description for your code review.
7. Click **Create**.

You can edit this review at any time to remove or delete a repository, or to add or delete an existing review.

Merge options

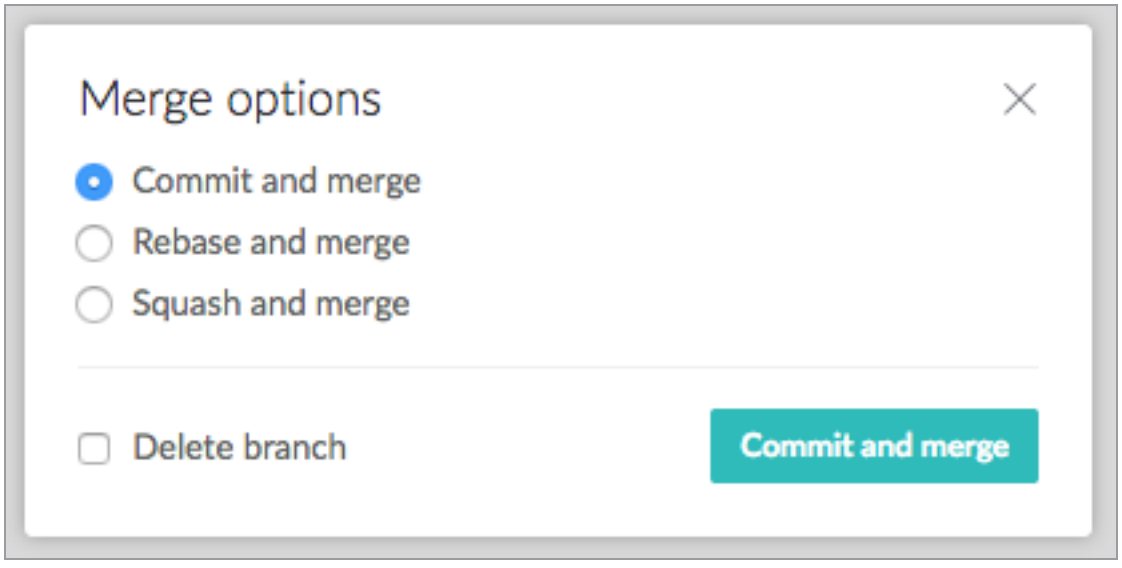
For single code reviews, you can access merge options by clicking the cogwheel icon next to the primary merge action.



Helix TeamHub supports the following methods for merging a code review:

- **Merge commit:** Helix TeamHub retains the full history of changes without fast-forwarding. For Git repositories, this means that the `--no-ff` flag is applied when merging. This is the default method unless a different method is configured for the destination repository.
- **Rebase and merge:** Helix TeamHub rebases all commits individually. It also fast-forwards the source branch to the newly rebased head, therefore avoiding the creation of an explicit merge commit. Use rebase when you want to have a clean history free of separate merge commits.
- **Squash and merge:** Squash and merge combines a set of commits into a single commit. Use squash and merge if you want to keep the history minimal. A real life example for using this option would be a feature or bug fix branch with multiple work-in-progress commits that you want to combine into a final commit.

You can also select the option to delete the source branch after a merge.



Note
You cannot merge individual reviews that are part of a multi-repo code review. You can only merge the multi-repo code review when all included reviews are ready for merge.

When you merge a multi-repo code review and conflicts occur, the merge fails. In this case, you need to manually resolve the conflict from the command line.

Deleting a branch

Select the **Delete branch** check box to delete the source branch after merging.

Note
You might not always be able to delete the source branch due to insufficient privileges (project or repository role), for example if the source branch is protected or if you are merging a fork. In this case, the check box is disabled.

Configuring defaults

Repository administrators have the ability to configure and enforce default code review settings for each repository. See the Repository settings section for more information.

Force merging

Normally, the **Merge** button is unavailable when the code does not meet the merge requirements: 

However, if the **Force Merge** feature is enabled for a repository and the current user is an administrator, the **Merge** button remains enabled. In this case, it displays in red to indicate that TeamHub enforces merging and ignores unmet merge requirements: 

Force merging is available to project and repository admins only.

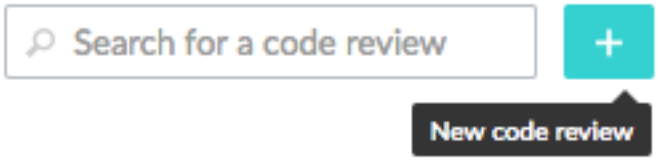
Pull requests (code reviews from forks)

Note
Helix TeamHub supports forking for Mercurial and native Git repositories.

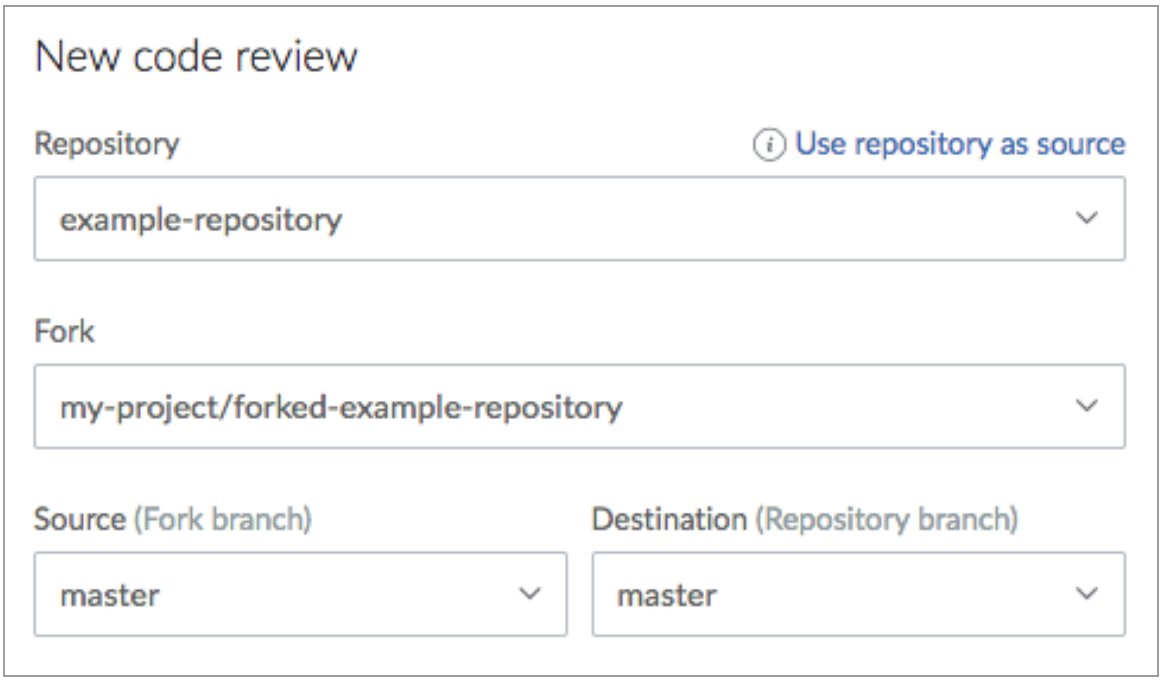
Pull requests are a way of merging the changes in a fork back to the parent repository. In Helix TeamHub, pull requests are simply code reviews whose source and destination branches belong to separate repositories in different projects. They have the same functionality as regular code reviews.

You create pull requests like any other code review. To create a new pull request, do the following:

1. Navigate to the project containing the parent repository (the repository into which you want to merge the changes).
2. Inside the project, browse to the **Code Reviews** view and click the **New Code Review** button (the plus icon).



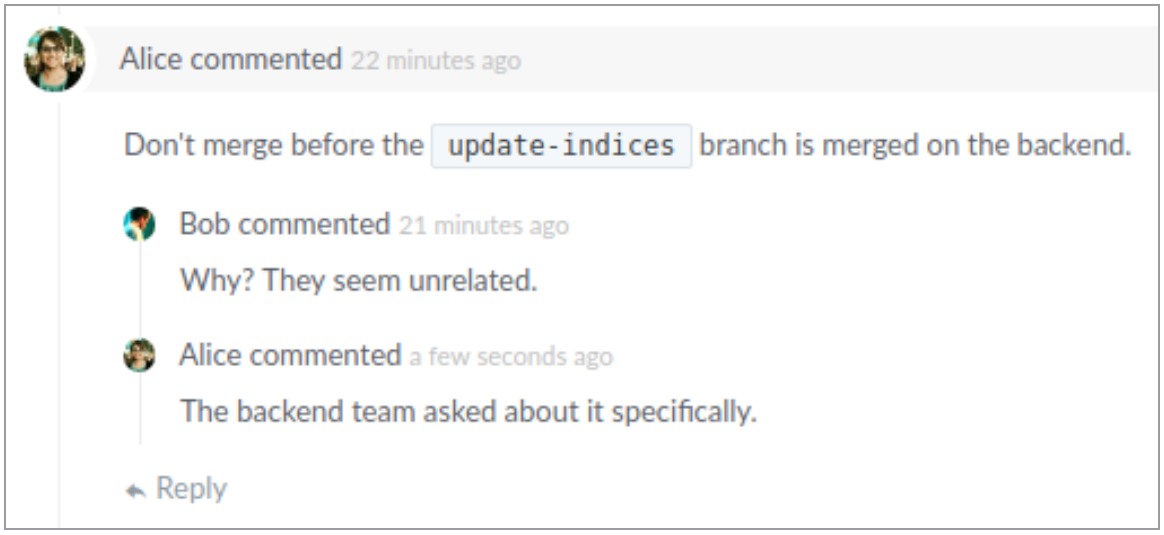
3. In the **New Code Review** form, select the repository from which the fork was created. If forks are available, a link appears on the right.
4. Click the link to display the **Fork** dropdown list and select the project containing the forked repository. The contents of the **Source** dropdown will now reflect the forked repository.
5. Select the source containing the changes you would like to merge.
6. From the **Destination** dropdown, select the target into which you would like to merge the changes.



7. Fill in the rest of the form and click **Create** to create a pull request.

Commenting in code reviews

Comments are a way to collaborate on code reviews. They allow team members and external collaborators to discuss code reviews, request changes, suggest solutions, and so on.



TeamHub organizes discussions in threads. For details, see Replying to Comments.

You can edit comments, but you cannot delete them.

Users watching the CR receive email notifications about new comments and replies.

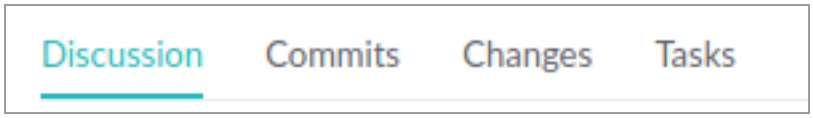
On the **Changes** tab of a code review, you can comment on specific lines of code, letting involved parties scope their discussions to specific fragments of code. For details, see Code Line Comments.

By default, the **Changes** tab shows differences inline. If you prefer to see them next to each other, click the **Side by side** option at the top right of the tab. You can also hide whitespace differences by selecting the **Ignore whitespace changes** check box.

You can use comments as tasks to indicate that some work has to be done before a code review should be merged. For details on this feature, see Tasks.

Reading comments

You can see comments on the **Discussion** tab of a code review.

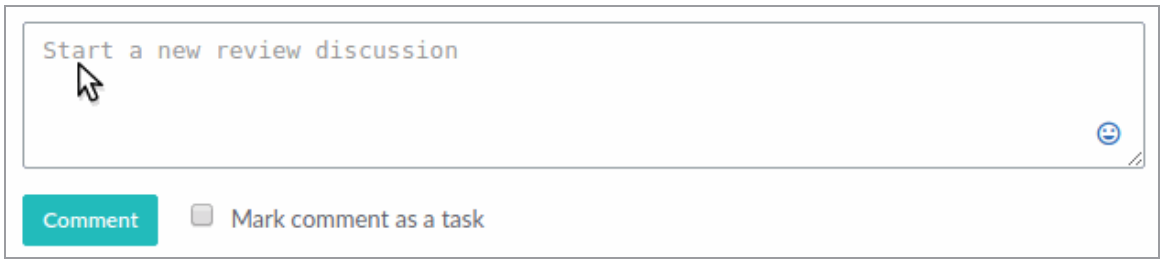


The tab lists comments along with other types of activity, such as commits and approvals. Comments appear chronologically, with newer comments at the bottom of the list. Replies appear grouped under respective parent comments, sorted chronologically.

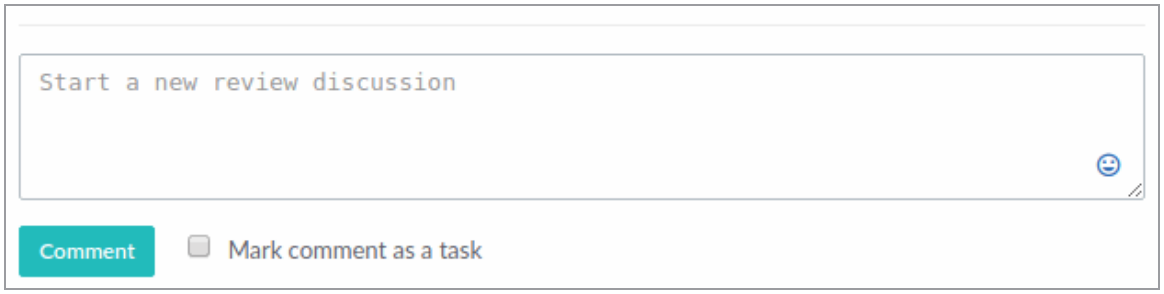
Creating comments

To create a new code review comment, do the following:

1. Type your comment in the text field at the bottom of the **Discussion** tab. As you type, the **Preview** link becomes available below the field.
2. (Optional) Click the **Preview** link to see what the comment will look like when submitted. To continue editing, click the **Editor** link.

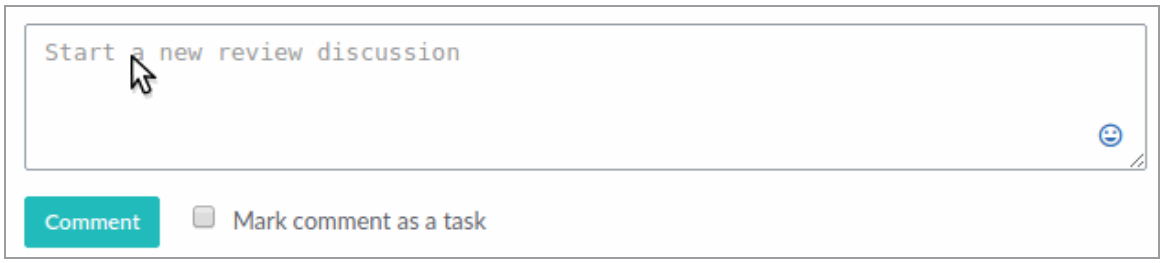


3. Click **Comment** to submit your comment.



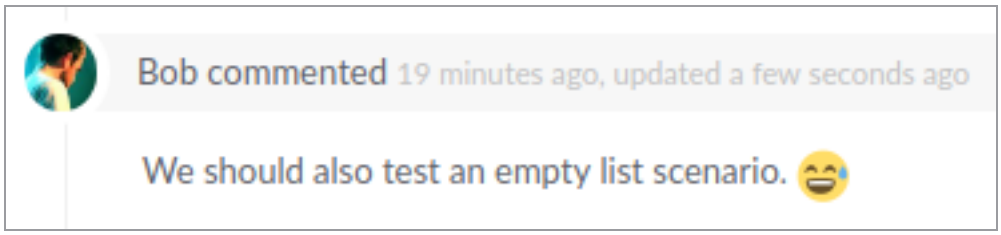
Formatting comments

Use Markdown syntax to format your comments.



Inserting emoji

Comments support emoji.

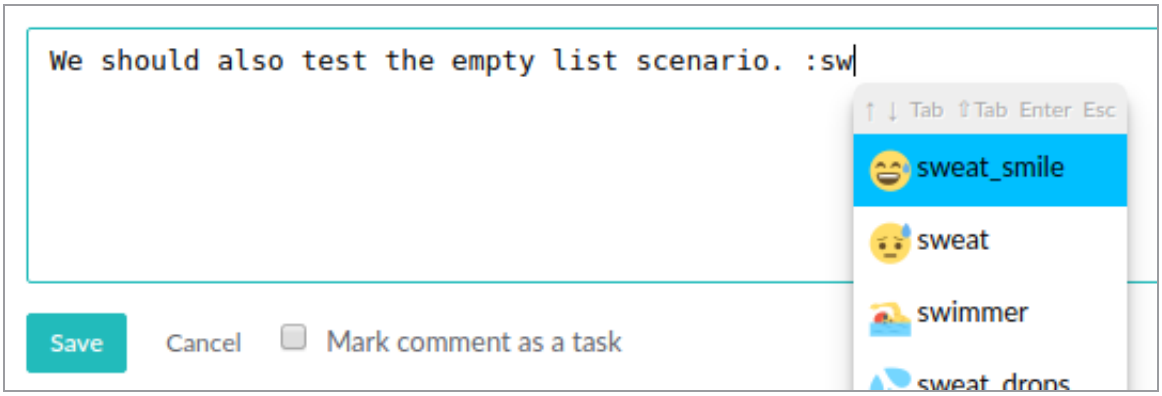


You can insert an emoji by:

- Typing emoji code surrounded by colons
- Using the emoji picker popup, either in a comment, a reply to a comment, or as a reaction to a comment

To insert emoji manually:

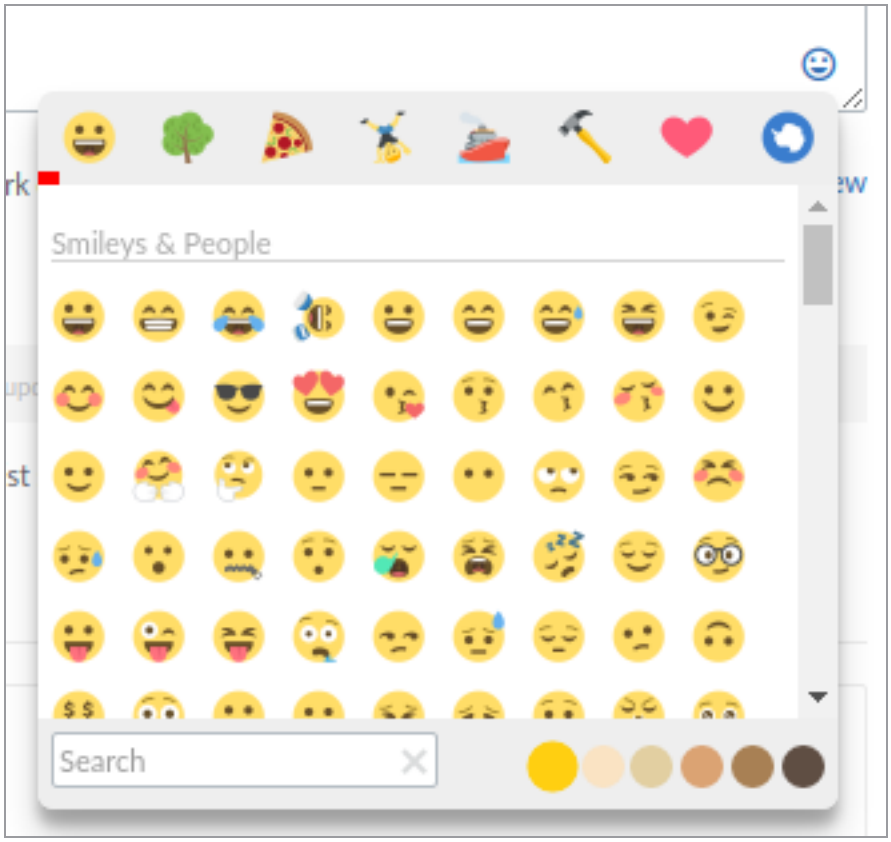
1. Type a colon and at least two characters of an emoji code, for example `:fo`. An autocomplete popup appears. You can keep typing to narrow down the list of emoji.
2. Select an emoji with the mouse pointer or keyboard. You can tab and shift-tab to navigate the list, press `Enter` to select an emoji, or press `Esc` to dismiss the popup.



To insert emoji from the emoji picker:

1. Do one of the following:
   - In the comment field, click the emoji icon ☺ at the bottom-right to open the emoji picker popup.
   - To react to an existing comment, move the pointer over the comment to display the emoji icon ☺ on the right. Then click the icon to open the emoji picker popup.
2. Browse all available emoji or filter the list using the text field at the bottom. The colored circles at the bottom right of the emoji picker popup allow changing the skin color of anthropomorphic emoji. It has no effect on the round-face smileys.

3. Select an emoji and click anywhere outside of the emoji picker to close the popup.



Inserting attachments

TeamHub lets you add attachments to a comment or response.

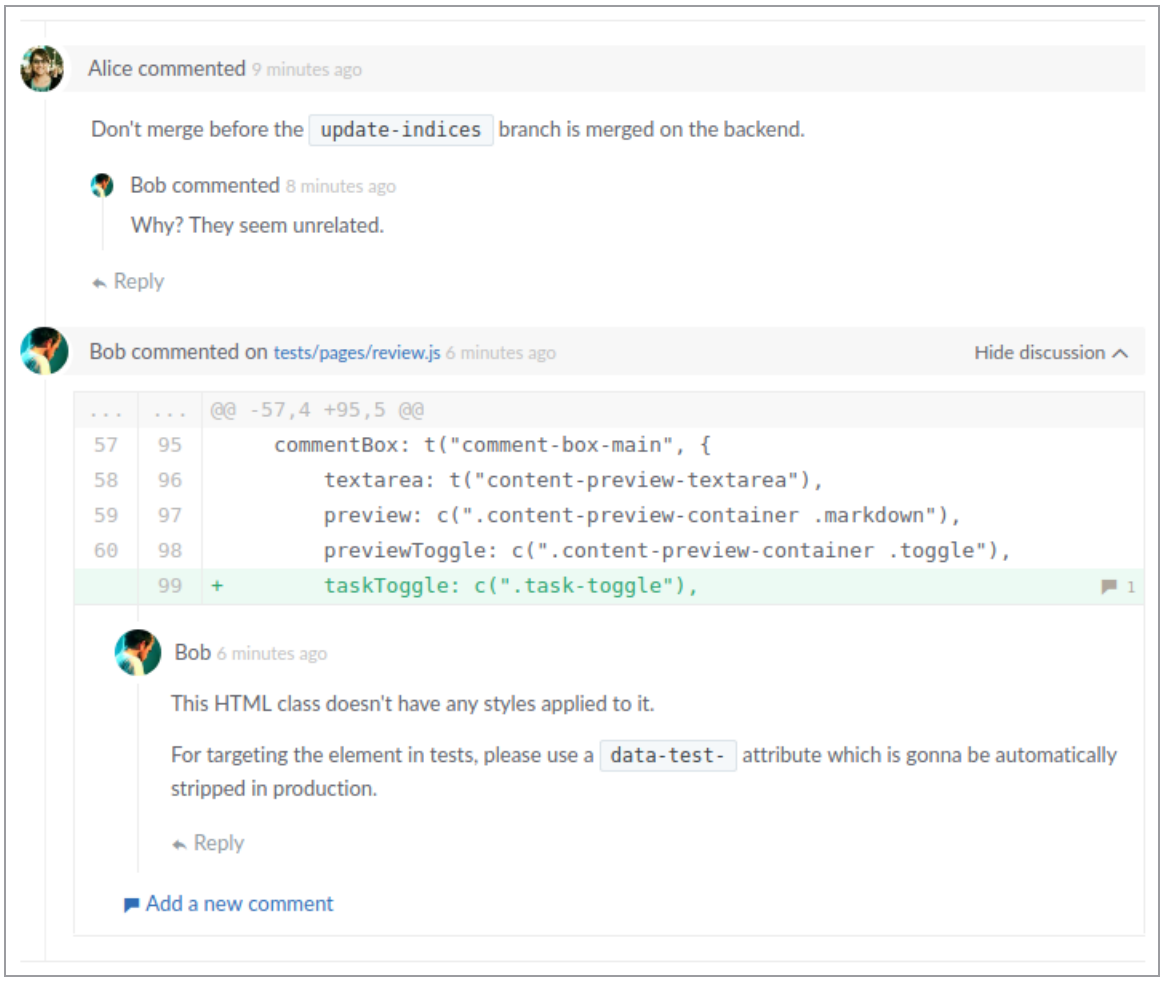
To add an attachment:

1. In the comment field, click the paperclip icon in the bottom right.
2. In the **Open** dialog, browse to the file you want to attach and click **Open**.

Replying to comments

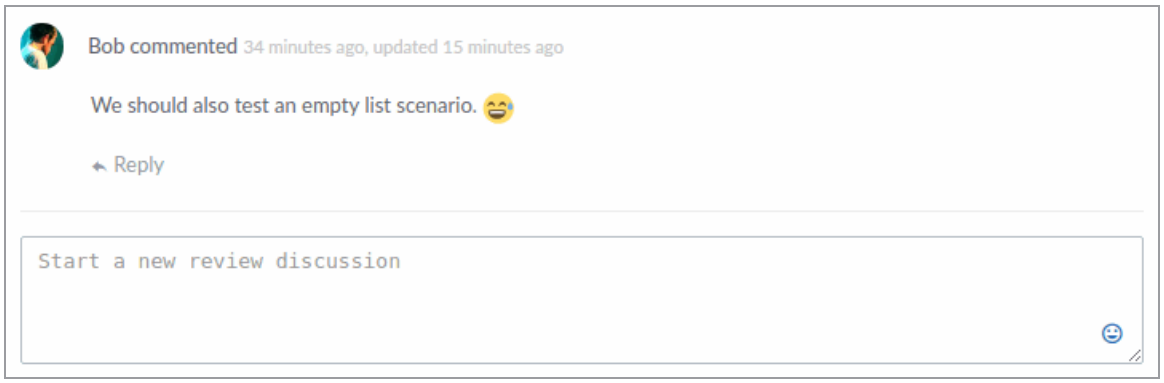
Each comment that you create via the comment form starts a new *thread*.

When you want to reply to a comment, do not create a new one. Instead, create a reply to the comment that you want to respond to. This lets the team have structured conversations, with unrelated matters kept separate:



To reply to a comment:

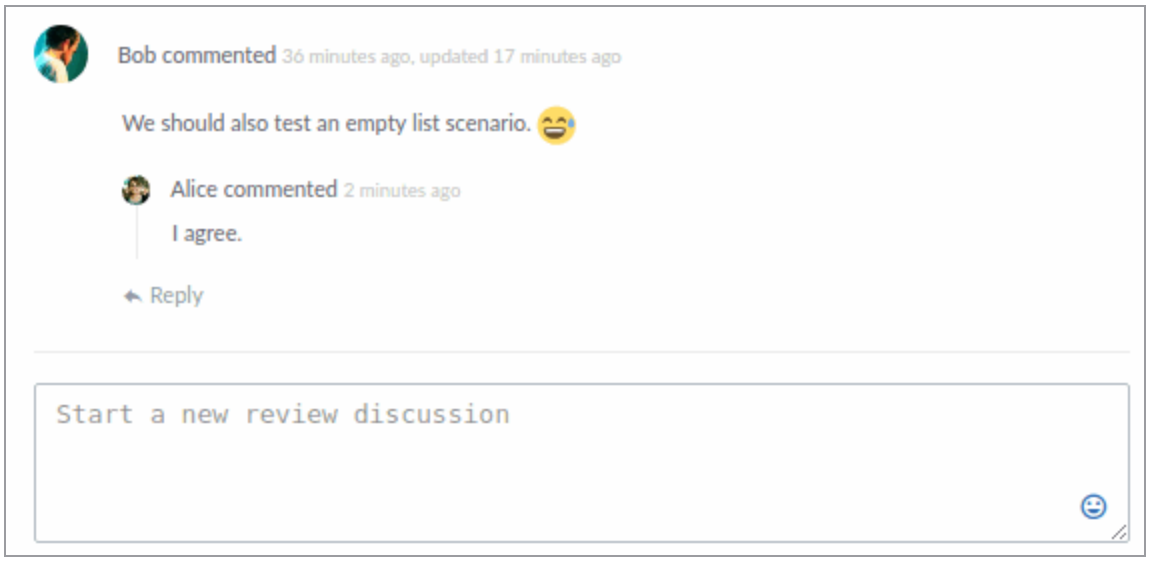
1. Click the **Reply** link below the comment. If a comment already has replies, the link is available under the last reply.
2. In the text field that opens, enter your response. The field is identical to the comment creation form, letting you insert text as well as emoji and attachments.
3. Click **Reply**.



Editing comments and replies

You can edit a comment or a reply by clicking the pencil icon.

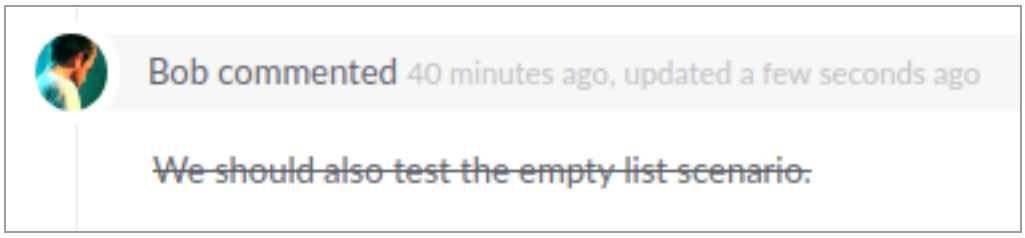
The icon is visible in the top-right corner of a comment when you hover your mouse pointer over the comment.



Deleting comments and replies

You cannot delete comments and replies. This is an intentional limitation aimed to prevent loss of information.

You can mark a comment as outdated, for example by applying formatting such as strikethrough or any other way you find reasonable. We recommend against editing out the text of outdated comments.



Email notifications

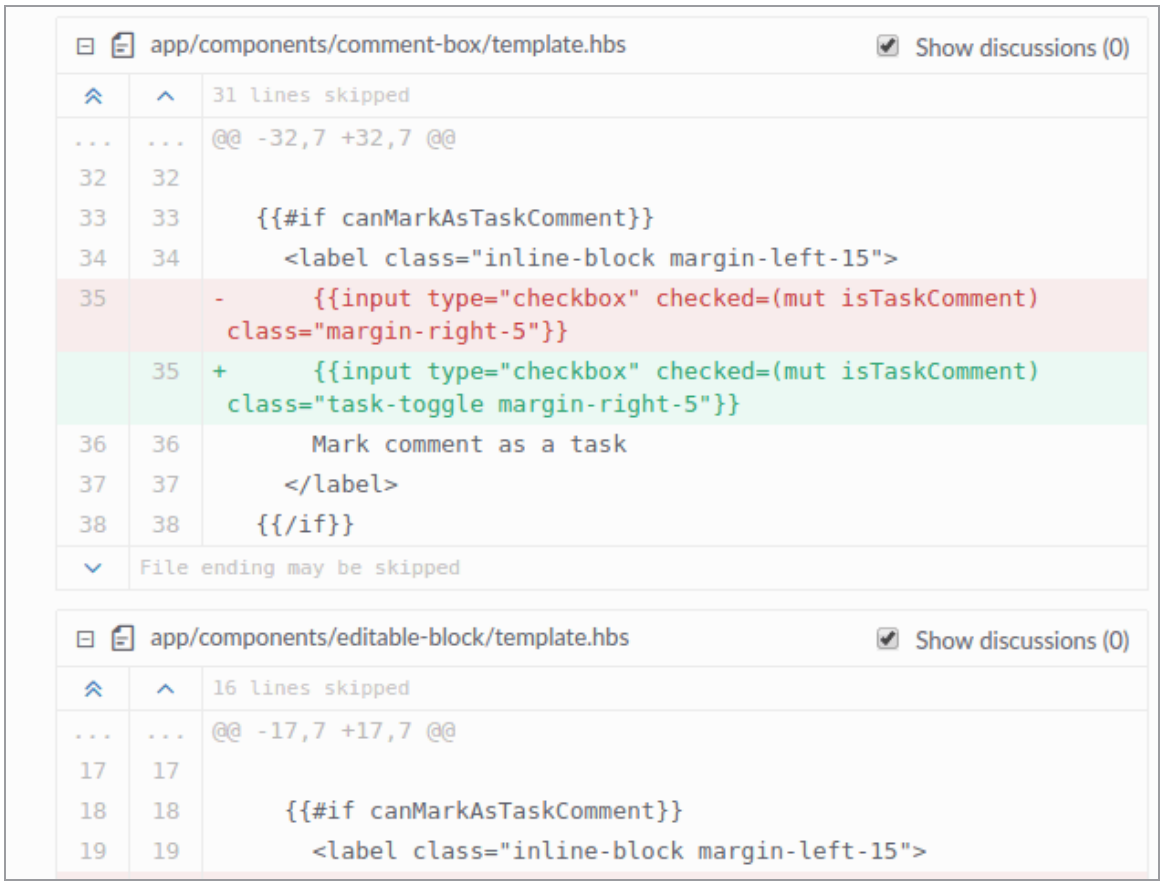
Users subscribed to the code review receive email notifications about new comments, along with other types of activity in a code review. Emails contain links to respective comments.

See Notifications for more info.

Code line comments

On the **Changes** tab of a code review, you can comment on individual lines of changesets. This is a convenient way to discuss code contributions, request changes, or suggest workarounds.

To create a line comment, hover the mouse pointer over a code line and click the comment button to the left.



Line comments behave just like normal comments: They are listed on the **Discussion** tab, you can reply to them, and so on.

You can start multiple comment threads per line. They will be grouped together on the **Discussion** tab under the same line of code. As a result, a new line comment may not appear at the bottom of the activity list, as you might expect as per chronological order.

When the source files are updated, line comments may become outdated: They are attached to lines that are no longer part of the code review. Outdated line comments disappear from the **Changes** tab, but you can still access them on the **Discussion** tab, where they are available in a collapsed view.

You cannot comment on lines that are not part of the code review. However, you can reveal such lines by clicking the expand button).

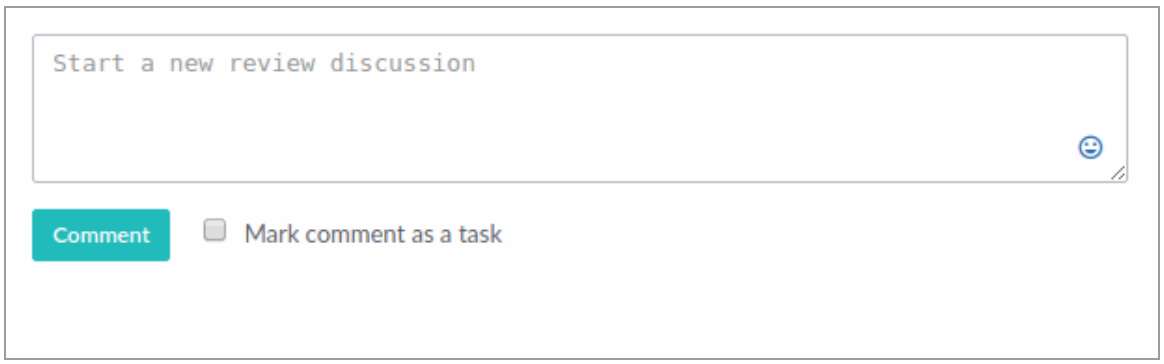
Code Review tasks

You can turn any comment into a task, and vice versa. Code review tasks let reviewers request changes that are required before the code review is considered ready for merge. You can configure a code review to disallow merging until all tasks have been resolved.

When all changes are done, you can mark the respective tasks as resolved.

Creating tasks

A task is simply a comment marked as such. You create a new task by creating a new comment and selecting the **Mark comment as a task** check box:



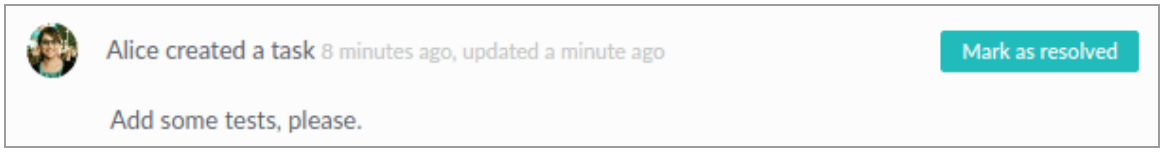
You can also turn existing comments into tasks or remove their task status.

Discussing tasks

You can discuss tasks by replying to them, just like with regular comments.

Resolving tasks

The **Mark as resolved/unresolved** button indicates the current task status:



Resolving indicates that the task is complete and does not require additional attention.

Resolve permissions

TeamHub handles resolve permissions as follows:

- Comment authors are able to resolve their own tasks.
- Company admins can resolve any task.
- Users with the *manage code reviews* permission can resolve tasks within the respective CRs. This permission is granted to:
  - Project members with admin, master, or developer role if repository authorization is disabled.
  - Repository members with admin, master, or developer role if repository authorization is enabled.

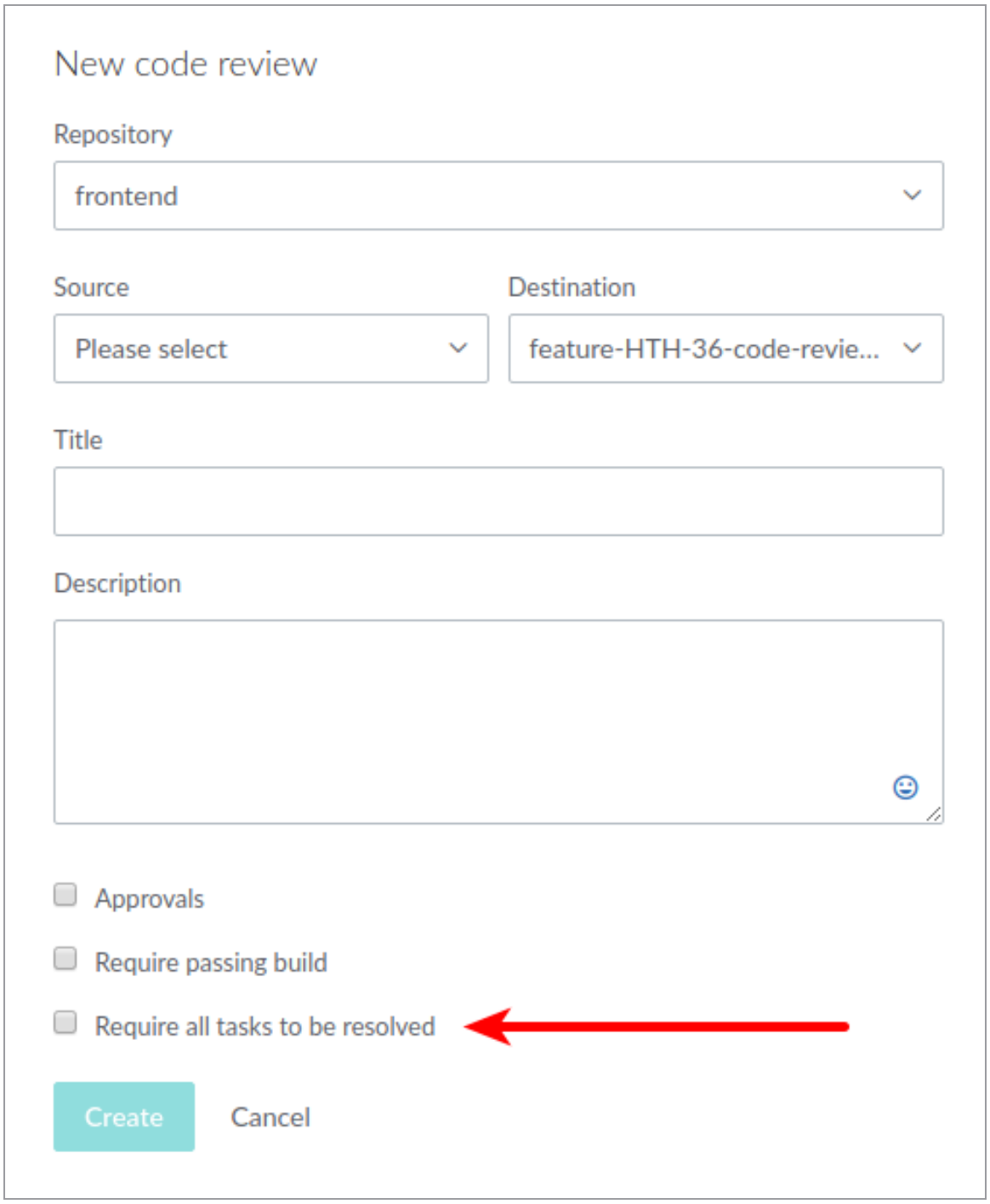
See Roles for more info.

Blocking the merge

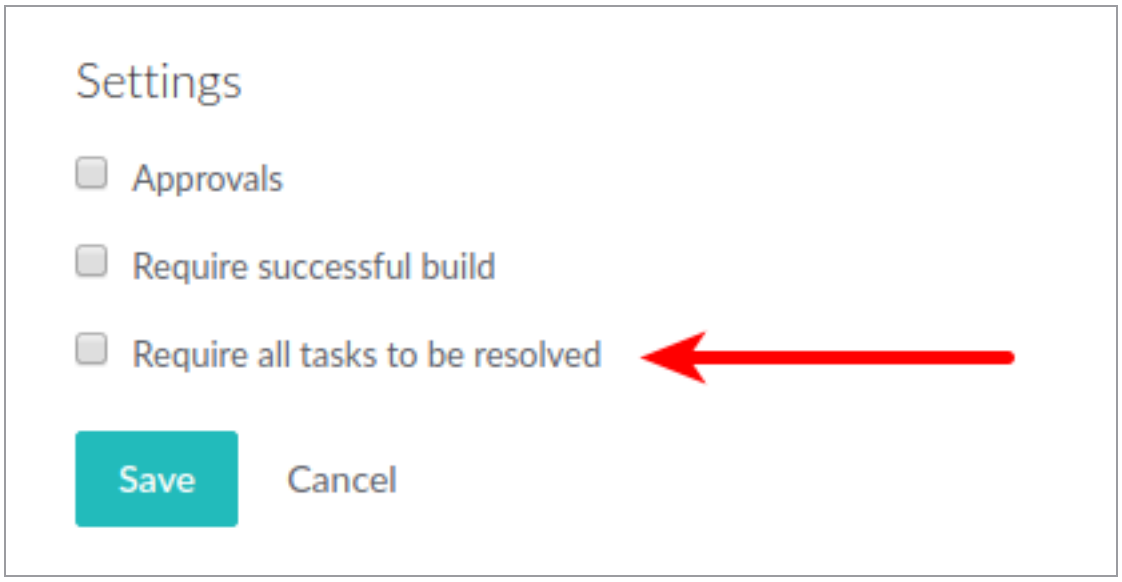
It is possible to configure a code review to prevent merging until all tasks are resolved. This helps ensure that low-quality code does not make it into the base branch.

For single code reviews, you can enable merge blocking by selecting the **Require all tasks to be resolved** check box in the **New code review** form or in the **Settings** form for the code review.

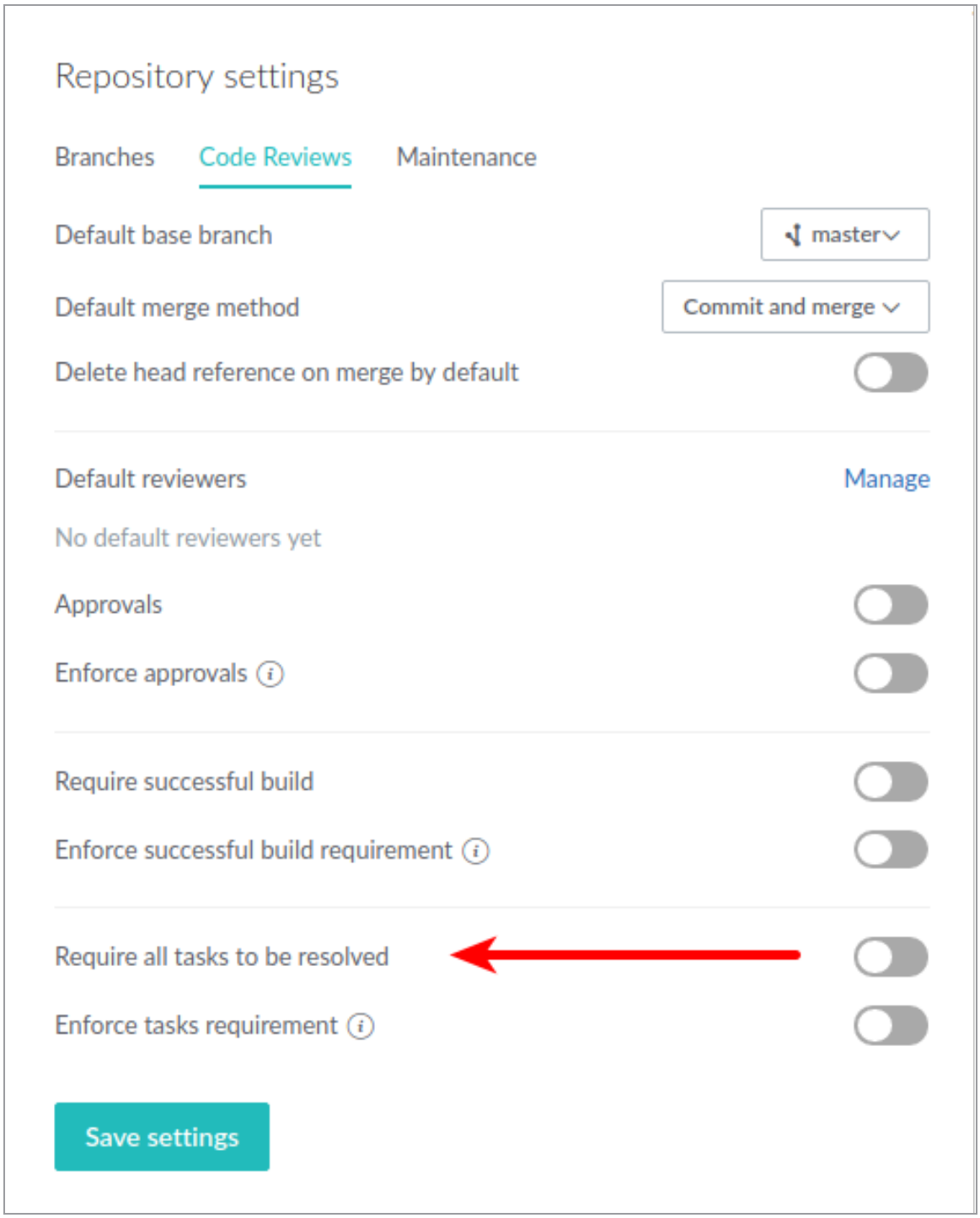
When creating a code review:



In the **Settings** form for the code review:

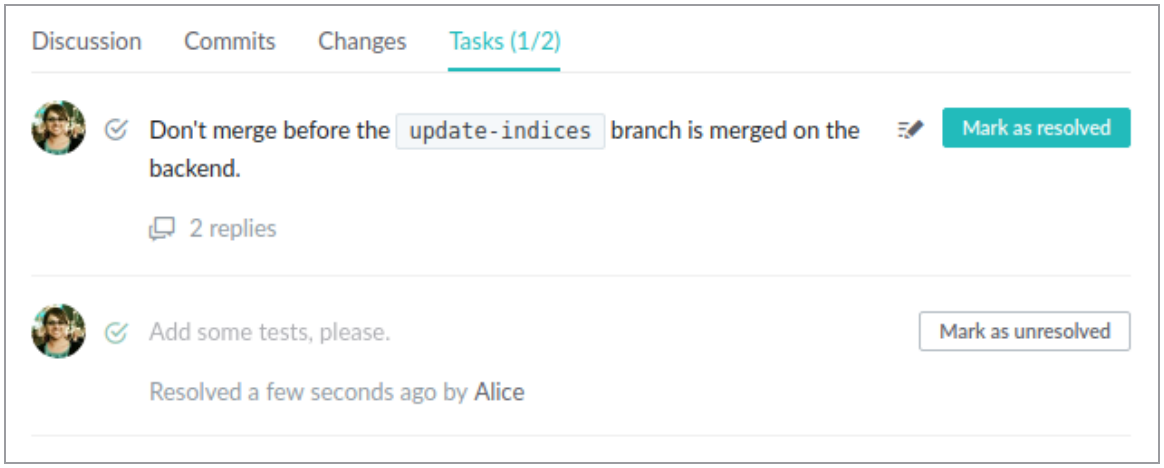


In addition, for single and multi-repo code reviews, you can configure the default behavior for unresolved tasks in the **Repository settings** form by enabling or disabling **Require all tasks to be resolved**:


Repository settings
Branches
Code Reviews
Maintenance
Default base branch
master
Default merge method
Commit and merge
Delete head reference on merge by default
Default reviewers
Manage
No default reviewers yet
Approvals
Enforce approvals
Require successful build
Enforce successful build requirement
Require all tasks to be resolved
Enforce tasks requirement
Save settings

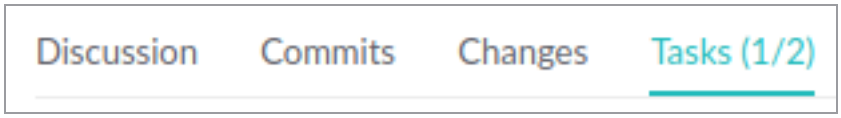
Reviewing the list of tasks

The **Tasks** tab lets you quickly review all tasks in a code review. It lists each task along with its state and number of replies. Click a task to proceed to its discussion.



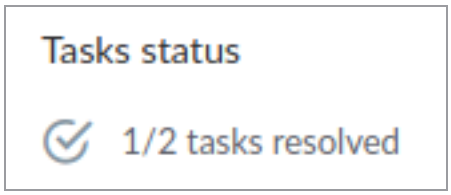
Reviewing the code review status

The label on the **Tasks** tab indicates the amount of resolved tasks and the total number of tasks.



The same information is available in the sidebar on the right. The color of the icon next to the number of tasks indicates whether all tasks have been resolved, as follows:

- Gray: One or more tasks are unresolved.
- Green: No task comments are available, or all comments have been resolved.



Configuring builds with Jenkins

The feature branch workflow requires a successful (green) build from Jenkins before you can merge the feature branch into the target branch in Helix TeamHub.

Initial setup

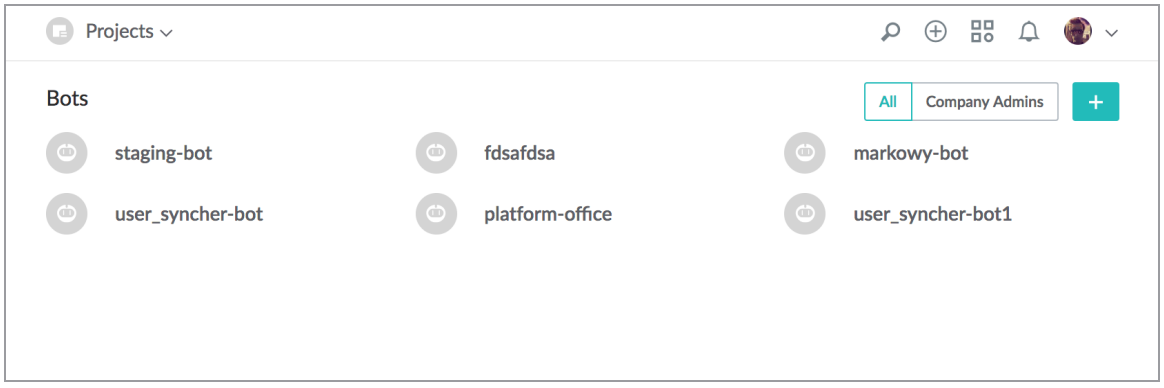
Perform the following steps for the initial setup:

1. Set up a TeamHub project and create a new Git or Mercurial repository. You also need to have some feature branches to the Git/Mercurial repository.
2. For Jenkins to execute the build, set up a Jenkins job to be triggered for the feature branches in this repository.
3. For Jenkins to notify TeamHub of whether the build succeeded or failed, do the following:
   a. Set up a TeamHub bot account for programmatic access.
   b. Set up a TeamHub Jenkins hook for triggering the Jenkins build.
   c. Configure the Jenkins job to build feature branches.
   d. Install and configure the Jenkins Helix TeamHub plugin.

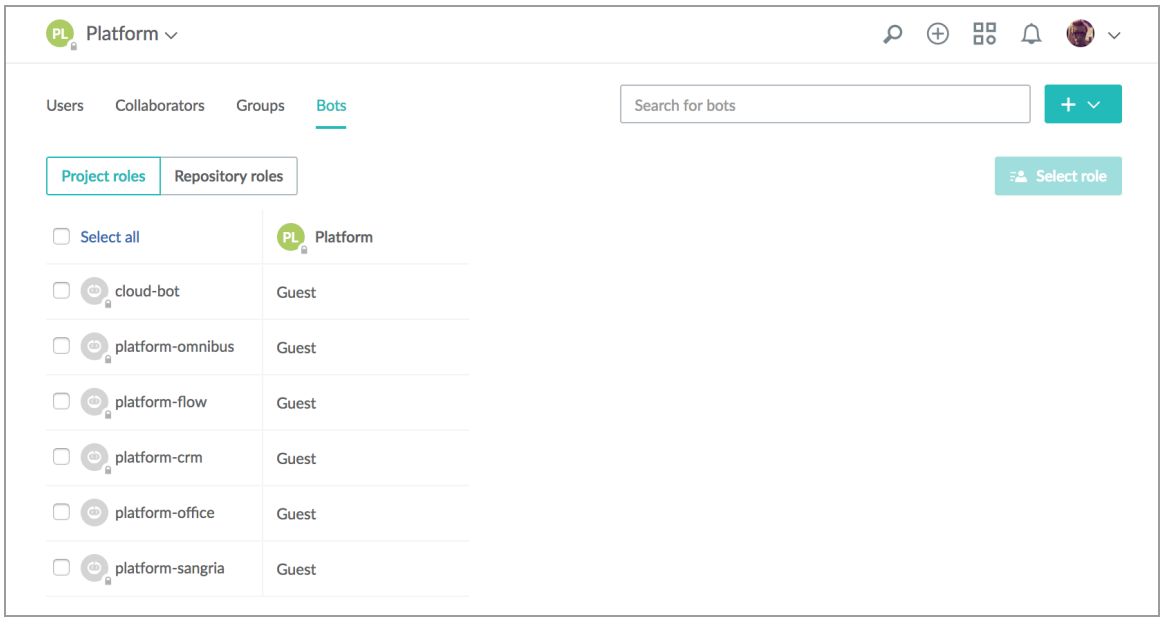
Set up a bot account for programmatic access

Helix TeamHub has a unique concept called bot accounts, or simply bots. Bots are used for external access to Helix TeamHub APIs and version control systems. Configuring a continuous integration server is an ideal example of using bots instead of your personal credentials.

Setting up a bot account in TeamHub is done through the bots UI. All users with access to TeamHub can create bots. When you create a bot, you become the owner of the bot, but you can share ownership of bot accounts between multiple users. As the following figure shows, you do not need to assign others as owners or members of the bot, but simply add the bot to the project.

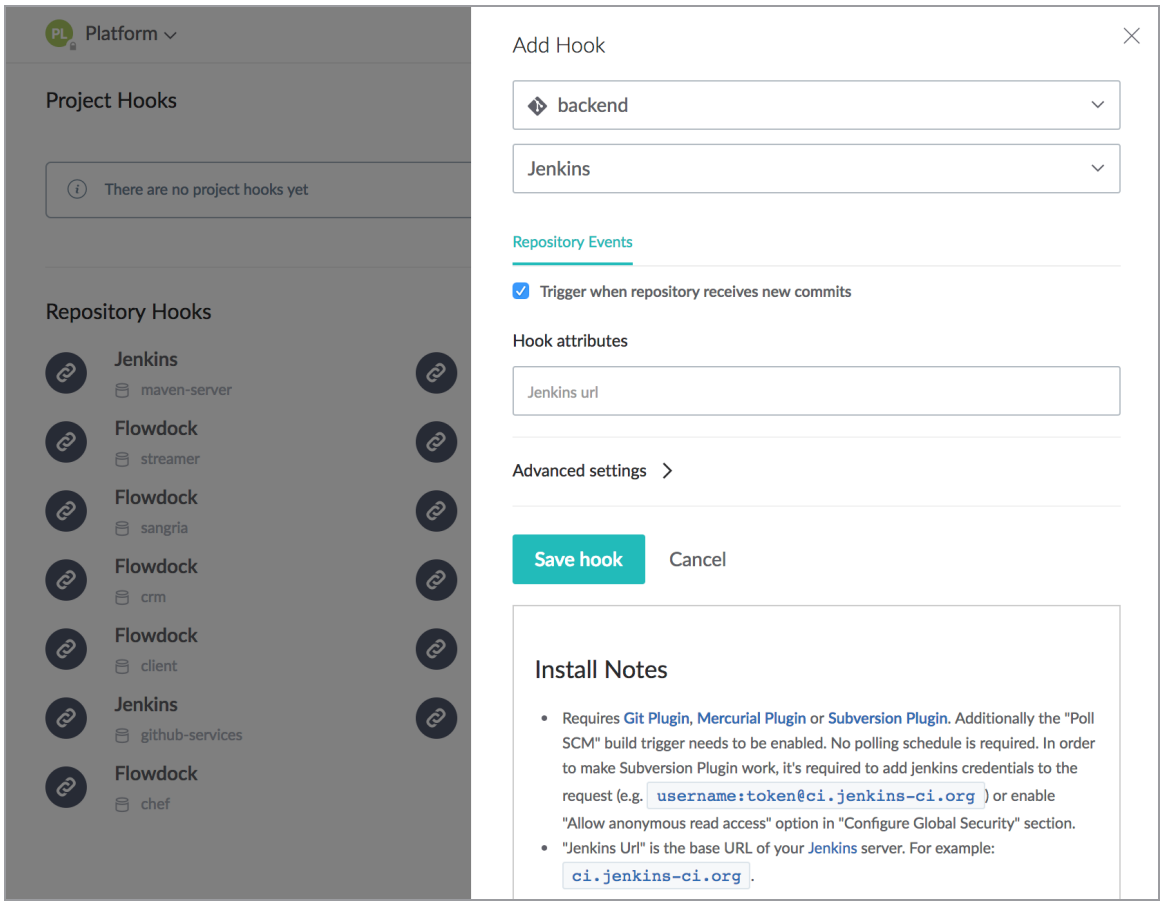


To add a bot to a given project, you use the project's **Team** view. For this purpose, you need to assign a guest role to the bot because it only requires read access to the code. Bots with guest access can publish build events. For a complete description of bot credentials, see "Bots & programmatic repository access" on page 22 in the *Helix TeamHub User Guide*.



Set up a Jenkins hook for triggering builds

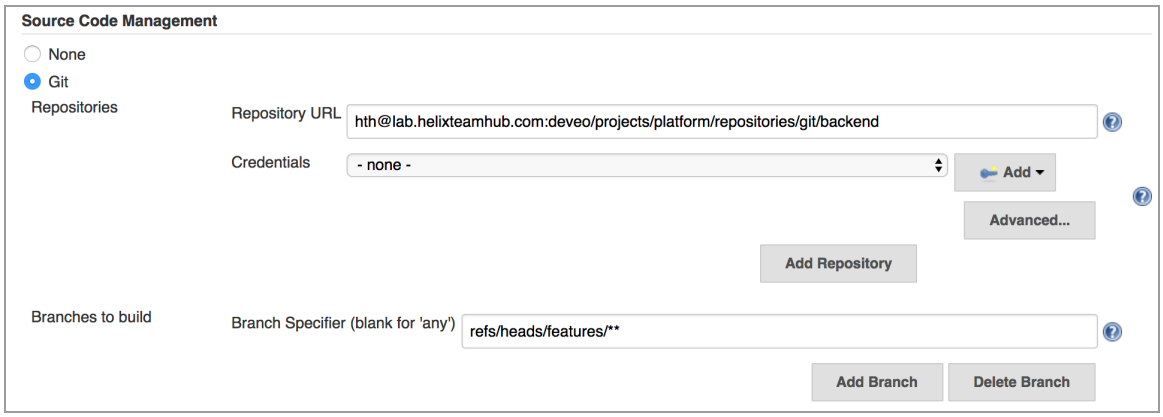
Next, you need to set up a Jenkins hook for triggering a build for each change in the feature branches. You manage hooks in the project's **Hooks** view. TeamHub supports over 80 hooks to different services. After you add the Jenkins hook to a Git repository, a commit hook posts a request to `http://yourserver/git/notifyCommit` after each change, in correspondence with Jenkins Git Plugin documentation.



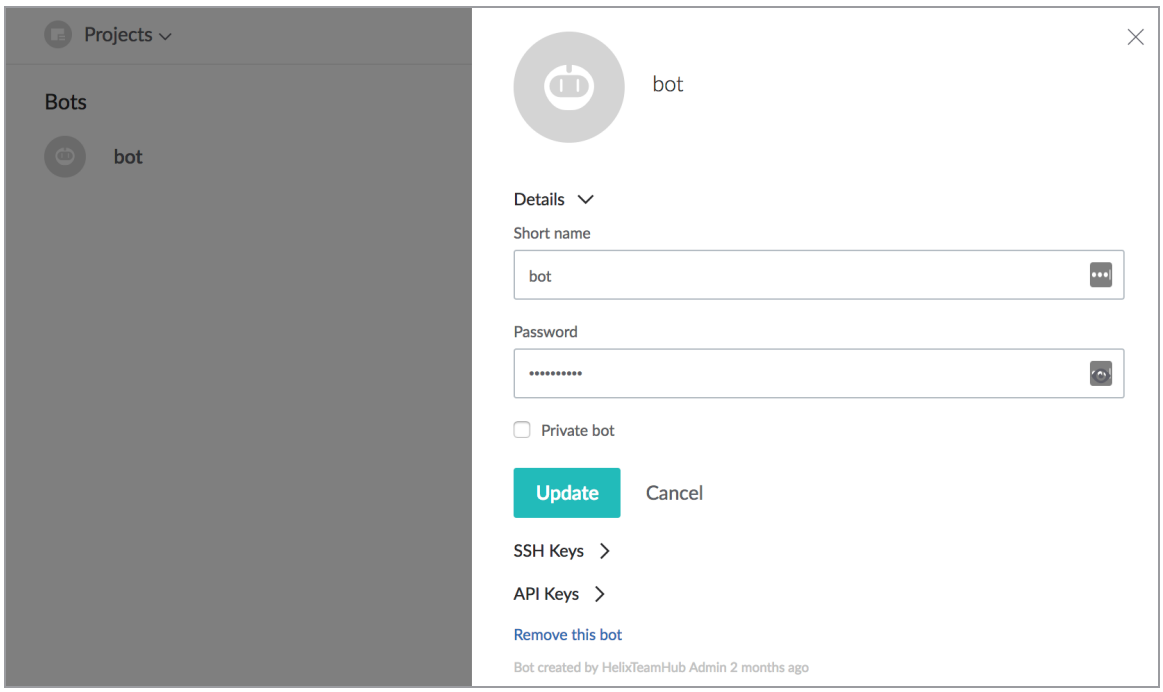
Configuring the Jenkins job

When setup on the TeamHub side is complete, you are ready to create a new job in Jenkins and configure necessary job-related settings. Perform the following steps:

1. Add the clone URL for the repository to the job configuration. TeamHub supports both SSH and HTTPS protocols for repository access. While SSH is typically preferred over HTTPS, in this example, you use HTTPS for simplicity.
2. Configure `branches to build` to `refs/heads/features/**`. This makes Jenkins run the job only upon changes to branches that are prefixed with features/, such as features/login, features/logging, features/foo, and so on.



3. Add your bot credentials to the configuration. In TeamHub, you can find them in the project's **Team** tab or in the company's **Bots** tab by clicking the cogwheel icon that appears when you hover over the bot name.



4. Enable the `Poll SCM` option in the Job configuration for the TeamHub Jenkins hook to start the execution of this job. However, you do not need to specify a polling schedule. Jenkins relies on this setting to determine which builds to start when changes occur.

Setting up the build steps is not covered here because those steps are project specific. For more information on configuring the build and test automation, refer to the documentation for the application in question.

Configure the TeamHub Jenkins plugin

You install the TeamHub Jenkins plugin from the Jenkins plugin manager. Make sure to configure it according to the plugin instructions. When you are done installing and configuring the plugin, a new post build action should be available in the Jenkins job configuration named `Helix TeamHub notification`.

For Jenkins to send build information successfully, you need to add the `Helix TeamHub notification` post build action and configure the account key of the bot account to it. This way, the plugin utilizes the correct credentials when creating the event. After setting the post build action and saving the Jenkins job settings, you are all set for proper testing.

To configure the plugin:

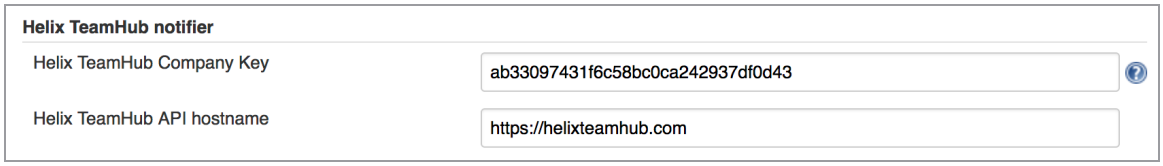
1. To allow Jenkins to send notifications to TeamHub, configure the API keys **for your Jenkins instance (Helix TeamHub Company Key** and **Helix TeamHub API hostname**), as follows.
   a. Look up the company key in TeamHub, as follows:

      Click the user list at the top right and select **User preferences**. Then click **API keys**.

   b. In Jenkins, select **Manage Jenkins > Configure System** to open the **Jenkins System Configuration.**

   c. Find the **Helix TeamHub notifier** section and provide the following:

      - **Helix TeamHub Company Key** field: The company key retrieved in step a
      - **Helix TeamHub API hostname** field: The hostname of your TeamHub instance



2. Specif which Helix TeamHub account each job should utilize, as follows:
   a. In Jenkins, select **Job > Configure**.

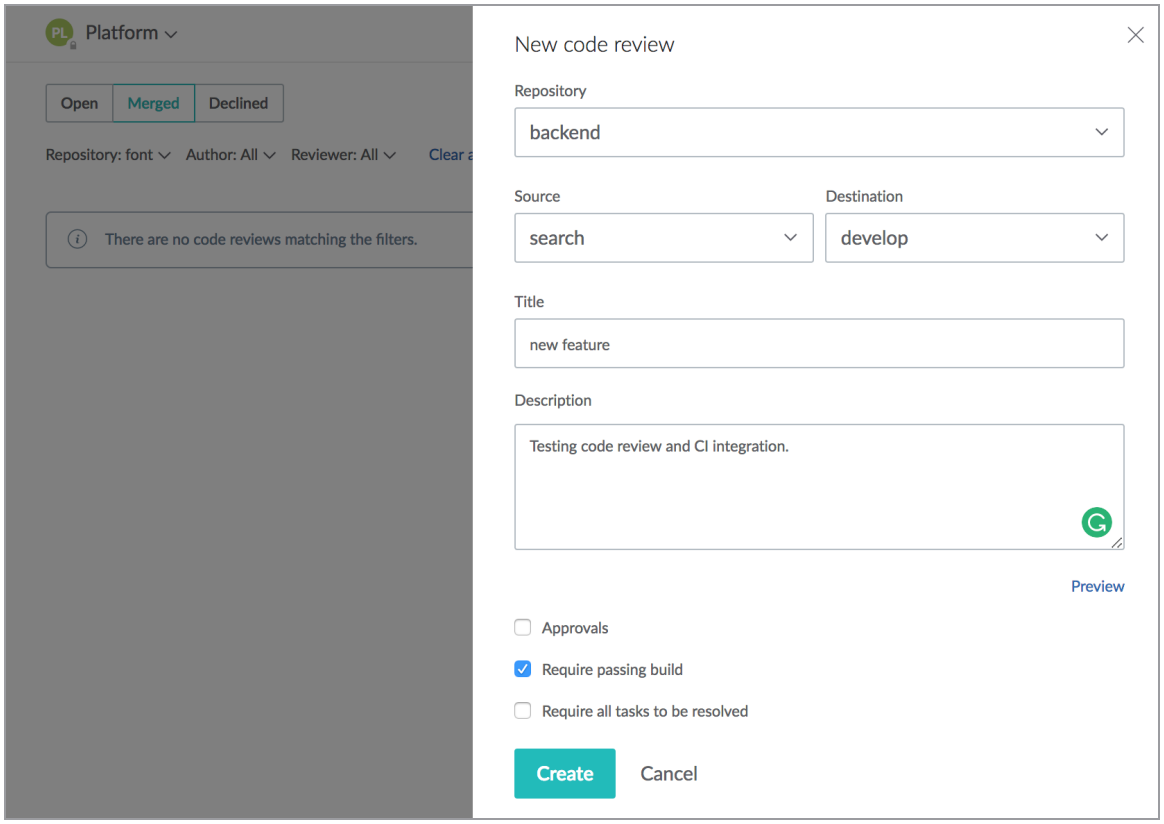
   b. In the **Job Configuration**, add a new **Helix TeamHub Notification Post-build Action** by providing the account key of the TeamHub account you want the job to utilize. This account is usually the bot account you created in "Set up a bot account for programmatic access" on page 109.

      To look up the bot account key, in TeamHub, in the **Bots** view, move the pointer over the bot you created for this purpose and click the gear icon that appears. In the details panel on the right, expand the **API Keys** section to view the **Account Key**.

Testing the setup

You can now test the setup by creating a feature branch named `features/new-feature`. Create a couple of commits to the branch and push it to Helix TeamHub. Once the branch is pushed, you can create a new code review with the branch against the master branch and select the `Require passing build` check box.

Selecting the `Require passing build` check box disables merging the changes through the TeamHub web interface until TeamHub receives a successful build notification for the given branch. However, you can still merge changes manually through the command-line, which is sometimes useful.



When you click the **Create** button in the **New code review** form to start the code review, you should see that the Build event is already green and that merging changes is possible. However, automated builds and verifications might take longer, and the successful or unsuccessful status only shows after the build is executed.


Platform
State: OPEN
1
0
Merge
Decline review
!683 Avoid stripping protocol prefix from links
Ville Vainio wants to merge sangria/feature-HTH-141-persist-protocol-prefix to sangria/develop
Discussion
Commits
Changes
Tasks
This code review has no tasks
Reviewers
1 more approval required
No reviewers
Build status
Build was successful
See details
Tasks status
There are no tasks
Maintenance
Settings

Code search

Note
Helix TeamHub supports code searching for Mercurial, Git, and Helix Git repositories.

The code is indexed from each repository's default branch. You can configure the default branch for each repository from the repository settings view.

Code search can be performed on the following levels:

- Company level (all repositories within the company)
- Project level (all repositories within one or multiple projects)
- Repository level (one or multiple repositories)

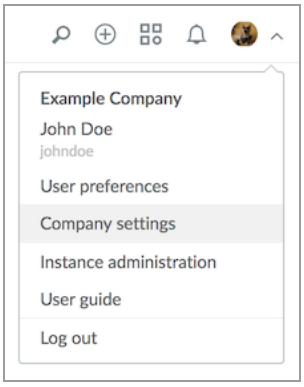
Enabling code search

Only user with company administrator privileges can enable code search.

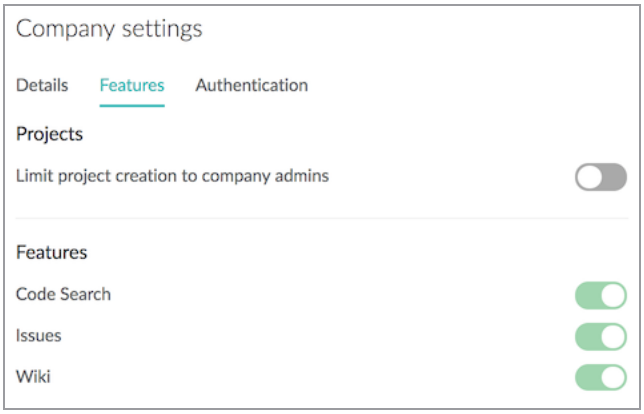
Note
On-premise customers will first need to setup and configure Elasticsearch to their Helix TeamHub instance. Refer to the administrator's guide on how to achieve this. If you require further assistance, please don't hesitate to contact support.

To enable code search:

1. Access the **Company Settings** view using the user dropdown list in the header or from the **My Dashboard** tab.



2. In the **Features** tab, turn on **Code Search** and click **Save**.



Once code search is enabled, TeamHub indexes all repositories within the company. Depending on the amount of data to process, the indexing operation might take a while. You can monitor the current progress on the **My Dashboard** tab. When done, the indicator on the right displays **Indexing complete**.

For information on other features that company admins can configure, see "Feature settings" on page 169.

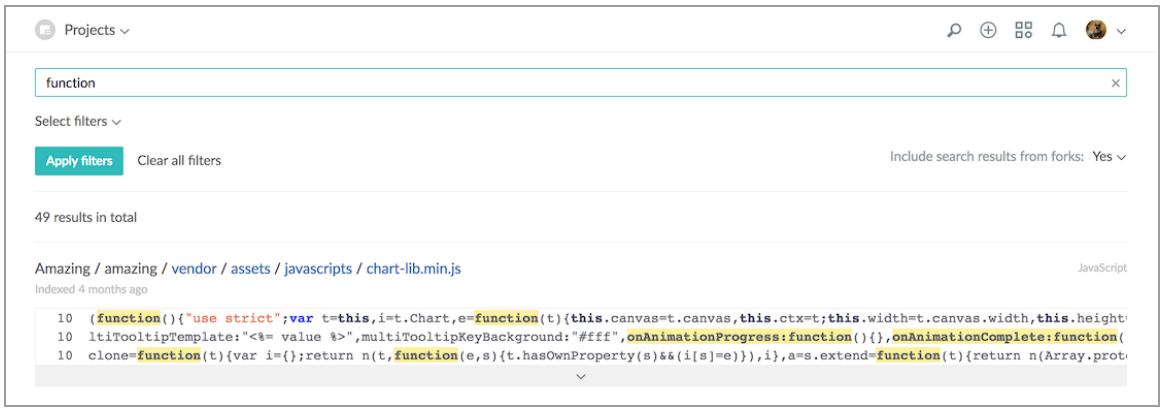
Searching

You can begin a code search from any view. To search the code:

> **Tip**
> You can limit code searches by path/filename, make advanced searches, and filter your search results, see "Limiting a code search" on the facing page, "Advanced searches" on page 119, and "Filtering search results" on page 120.

1. Click the search icon in the site header.
2. In the search field, enter your search term and press Enter.

TeamHub displays the search results in a separate view.



By default, TeamHub displays only the matching code lines.

3. To view the full search context, click the arrow icon beneath the result.
4. To see more results, use the pagination controls at the bottom of the page.

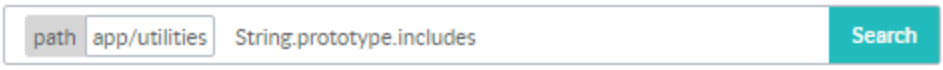
Limiting a code search

You can limit code searches to a path, a filename, or a filename in a path by entering your search term followed by the path, filename, or path and filename.

> **Tip**
> More than one path and filename can be entered for a search term.

Limit a code search to a path

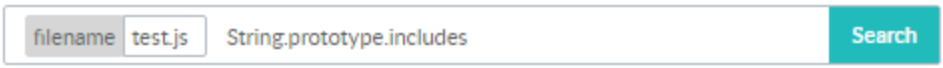
For example, **`String.prototype.includes path:app/utilities`** followed by Enter, TeamHub creates the path as a tag:



Wildcards are supported in the path.

Limit a code search to a filename

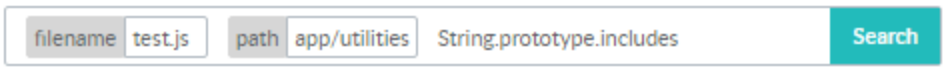
For example, **`String.prototype.includes filename:test.js`** followed by Enter, TeamHub creates the filename as a tag:



The filename is only searched if an exact match is found. Wildcards are not supported for filenames.

Limit a code search to a filename in a path

For example, **`String.prototype.includes filename:test.js path:app/utilities`** followed by Enter, TeamHub creates the filename as a tag and path as a tag:



The filename is only searched if an exact match is found and the filename is in the path. Wildcards are not supported for filenames.

Advanced searches

You can use special characters to make your results more relevant.

Search for an exact match

To search for an exact match, surround your search term with quotes **`"`**. For example, to find results that exactly match **`String.prototype.includes`** enter the following search term:

`"String.prototype.includes"`

> **Note**
> In most cases, wrapping your search term in quotes will result in exact matches for your search query. However, due to the way that Elasticsearch indexes and creates its "phrase query" for your search term, it can result in unexpected search results.
>
> **For example:** searching for **`"String.prototype.includes"`** could result in the following being displayed in the search results:
>
> - **`String.prototype.includes`** - expected
> - **`String.PrototypeObject.includes`** - not expected
> - **`StringTheory.prototype.includes`** - not expected

Search for term-x and term-y (AND)

To find results that contain **`term-x`** and **`term-y`**, separate your search terms with a space. For example, to find results that contain **`String`** and **`prototype`** enter the following search term:

```
String prototype
```

Search for term-x, term-y, or both (OR)

To find results that contain **`term-x`**, **`term-y`**, or both, separate your search terms with a pipe **`|`** character. For example, to find results that contain **`String`**, **`prototype`**, or both enter the following search term:

```
String | prototype
```

Search for term-x but exclude results that also contain term-y

To find results that contain **`term-x`** but exclude those that also contain **`term-y`**, use the minus **`-`** character in front of the term you want to exclude it. For example, to find results that contain **`String`** but not **`prototype`** enter the following search term:

```
String -prototype
```

Filtering search results

You can narrow down search results by specifying which projects and repositories to include.

Helix TeamHub applies the following filters automatically based on the view from which you invoke the search:

- Browsing a project sets the current project as the filter.
- Browsing repository views or the project's wiki sets the repository as the filter.

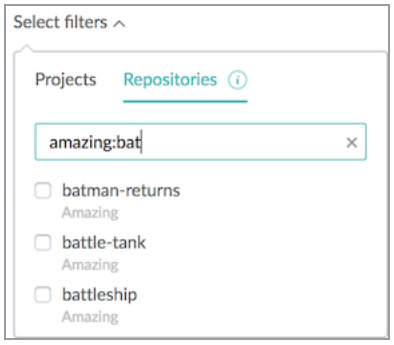
You can clear these filters in the results view.

Invoking a search from the company scope does not set any filters. Instead, TeamHub searches within all projects and repositories to which you have access. To make sure you are at the company scope level, click the company name or logo in the top left corner.

In addition to the project and repository filters, you can define whether to include or omit results from forked repositories using the dropdown list on the right.

The numbers next to the filters indicate the number of results for each project and repository. Note that by selecting a project, you are effectively searching in all repositories within that project. To restrict the search to specific repositories, first remove the project filter.

If you have multiple repositories with the same name in several projects, you can narrow down the search using the following syntax: `project:repository`. For example, to search repositories starting with `bat` in a project called `Amazing`, enter the following string in the search field: `amazing:bat`



To view a project's identifier, go to the **Projects** tab or move your pointer over the project name.

Webhooks

Webhooks let you notify external services when:

- A project has been created, deleted, or updated (company hook)
- A repository has been created (project hook)
- A tag/branch has been created or deleted
- New commits have been pushed to a repository (repository hook)

Helix TeamHub sends an HTTP POST request with a JSON payload to the configured endpoint URL. You can use any endpoint that is capable of processing the request.

A project's or repository's admin can configure project and repository hooks from the **Hooks** tab.

Adding a webhook

To create a webhook:

1. Do one of the following, depending on the type of hook you want to create:
    - For a company hook: Navigate to the **Hooks** tab in the company scope.

      To make sure you are at the company scope level, click the company name or logo in the top left corner.
    - For a project or repository hook: Navigate to the **Hooks** tab in the project scope.

      To make sure you are at the correct scope level, click **My Dashboard > <*project name*>** for the project scope or click **My Dashboard > <*project name*> > Repositories > <*repository name*>** for the repository scope.
2. Click the plus icon to open the **Add Hook** form.
3. From the service list, select **Webhook**. All hooks require the same parameters.
4. Set the parameters as follows:
    - **Url**: Provide the URL of the endpoint to which the payload is delivered.
    - **Content-type**: Specify whether the payload is serialized as **form (application/x-www-form-urlencoded)** (default) or **json (application/json)**.
    - **Secret**: Optionally, enter a string that is passed with the HTTP requests as an X-Hub-Signature header. The value of this header is computed as the HMAC SHA1 hex digest of the body, using the secret as the key.
    - **Insecure ssl**: Select if your endpoint is using an SSL certificate that cannot be verified (for example a self-signed certificate).

5. Optionally, if you want to override the default proxy configuration defined for the Helix TeamHub instance, under **Advanced settings**, select **Use custom proxy configuration**; then define a custom URL for the proxy or leave the field empty to disable the proxy altogether.
6. Click **Save hook**.

Company hook

Example JSON payload for a *new project created* webhook:

```
{
  "operation": "created",
  "type": "project",
  "subject": {
    "id": "chuck",
    "uuid": "a94ea07c-4590-4dc9-b397-c83ca5daf976"
  },
  "project": {
    "id": "test_project",
    "uuid": "c788fd2a-788c-4888-8673-90e027b1b849",
    "name": "Test project",
    "description": "Lorem ipsum",
    "visibility": "company"
  }
}
```

Project hook

A project webhook lets you notify external services when new repositories are created in Helix TeamHub.

Example JSON payload for a *new repository created* webhook:

```
{
  "operation": "created",
  "type": "repository",
  "subject": {
    "id": "chuck",
    "uuid": "a94ea07c-4590-4dc9-b397-c83ca5daf976"
  },
```

```
  "project": {
    "id": "test_project",
    "uuid": "c788fd2a-788c-4888-8673-90e027b1b849",
    "name": "Test project"
  },
  "repository": {
    "id": "website",
    "uuid": "ff8f33e9-d619-493e-872d-be7dd4a10235"
    "type": "git"
  }
}
```

Repository hooks

Repository hooks let you notify external services when new commits are pushed to a repository.

Example JSON payload for a *new commit pushed* webhook:

```
{
  "after": "67ec79c2cc2737eec07b649555b3da32c47d095b",
  "ref": "refs/heads/master",
  "before": "c58a421ed77556d217abc7638de9ba9b3589b36d",
  "compare": "",
  "forced": false,
  "created": false,
  "deleted": false,
  "project": {
    "uuid": "c788fd2a-788c-4888-8673-90e027b1b849",
    "name": "Test project",
    "url": "https://helixteamhub.com/example/code/diff/test"
  },
  "repository": {
    "uuid": "ff8f33e9-d619-493e-872d-be7dd4a10235",
    "name": "website",
    "type": "git",
    "url":
```

```
"https://helixteamhub.com/example/code/overview/test/repositories/websit
e",
    "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/website",
    "ssh_url":
"hth@helixteamhub.com:example/projects/test/repositories/git/website",
    "owner": {
      "uuid": "a94ea07c-4590-4dc9-b397-c83ca5daf976",
      "name": "chuck",
      "email": "chuck@example.com"
    }
  },
  "pusher": {
    "uuid": "a94ea07c-4590-4dc9-b397-c83ca5daf976",
    "name": "chuck",
    "display_name": "Chuck Norris"
  },
  "commit_count": 1,
  "commits": [{
    "distinct": true,
    "removed": [],
    "message": "Update readme",
    "added": [],
    "timestamp": "2015-01-30T12:17:56Z",
    "modified": ["readme"],
    "url":
"https://helixteamhub.com/example/code/diff/test/repositories/website/comm
its/67ec79c2cc2737eec07b649555b3da32c47d095b",
    "author": {
      "name": "Chuck Norris",
      "email": "chuck@example.com"
    },
    "id": "67ec79c2cc2737eec07b649555b3da32c47d095b"
  }]
}
```


Example JSON payload for a *new branch created* webhook:

```
{
  "after": null,
  "ref": "refs/heads/branch2",
  "before": "",
  "compare": "",
  "forced": false,
  "created": true,
  "deleted": false,
  "project": {
    "uuid": "5a7f9f16-2e2d-4582-b2f6-9cd866fe9b40",
    "name": "Test",
    "url": "https://helixteamhub.com/example/projects/test"
  },
  "repository": {
    "uuid": "02f347d4-5ec2-4189-b5d0-eb487d15cc2e",
    "name": "website",
    "type": "git",
    "url":
"https://helixteamhub.com/example/projects/test/repositories/website",
    "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/website",
    "ssh_url":
"hth@helixteamhub.com:example/projects/test/repositories/git/website",
    "owner": {
      "uuid": "bb501310-0292-4925-88e3-ba3b3d53a795",
      "name": "Chuck Norris",
      "email": "chuck@example.com"
    }
  },
  "pusher": {
    "uuid": "bb501310-0292-4925-88e3-ba3b3d53a795",
    "name": "Chuck Norris",
    "display_name": "chuck@example.com"
  },
  "commit_count": 0,
```

```
  "commits": []
}
```

Example JSON payload for a *tag deleted* webhook:

```
{
  "after": null,
  "ref": "refs/tags/v3.0",
  "before": "",
  "compare": "",
  "forced": false,
  "created": false,
  "deleted": true,
  "project": {
    "uuid": "5a7f9f16-2e2d-4582-b2f6-9cd866fe9b40",
    "name": "Test",
    "url": "https://helixteamhub.com/example/projects/test"
  },
  "repository": {
    "uuid": "02f347d4-5ec2-4189-b5d0-eb487d15cc2e",
    "name": "website",
    "type": "git",
    "url":
"https://helixteamhub.com/example/projects/test/repositories/website",
    "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/website",
    "ssh_url":
"hth@helixteamhub.com:example/projects/test/repositories/git/website",
    "owner": {
      "uuid": "bb501310-0292-4925-88e3-ba3b3d53a795",
      "name": "Chuck Norris",
      "email": "chuck@example.com"
    },
  },
  "pusher": {
    "uuid": "bb501310-0292-4925-88e3-ba3b3d53a795",
    "name": "Chuck Norris",
```

```
    "display_name": "chuck@example.com"
  },
  "commit_count": 0,
  "commits": []
}
```

Code review hooks

Some events are specific to code reviews:

- `code_review` - code review created/deleted
- `code_review_title` - title updated
- `code_review_description` - description updated
- `code_review_threshold` - threshold updated
- `code_review_require_build` - require build updated
- `code_review_require_task_comments` - require task comments updated
- `code_review_state` - state updated
- `reviewer` - reviewer joined/left/voted/unvoted
- `build` - build completed/failed

Example JSON payload a for *new code review created* webhook:

```
{
  "operation": "created",
  "type": "code_review",
  "code_review": {
    "uuid": "d9fb853c-cb2e-4f45-962a-b32f2d0f7948",
    "url": "https://helixteamhub.com/example/projects/test/reviews/7",
    "number": 7,
    "state": "open",
    "merge_state": "initial",
    "title": "sdf",
    "description": "",
    "created_at": "2017-11-15T10:35:07Z",
    "updated_at": "2017-11-15T10:35:07Z",
    "deleted_at": null,
    "require_build": false,
```

```
    "require_task_comments": false,
    "voting_threshold": 0,
    "base_branch": "test",
    "base_branch_type": "branch",
    "base_commit_id": "dd2b053486ab60ab1d3c4ac6c9b0669f85882894",
    "head_branch": "master",
    "head_branch_type": "branch",
    "head_commit_id": "8e399e9d20df2e6bb74778cfc135510aac65b1de",
    "creator": {
      "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
      "name": "admin",
      "display_name": "Admin"
    },
    "reviewers": [],
    "voters": [],
    "project": {
      "uuid": "372b0f0b-75bf-49c1-b9be-85aa47cdc6e7",
      "name": "test",
      "url": "https://helixteamhub.com/example/projects/test"
    },
    "repository": {
      "uuid": "45b71dd0-3ece-4862-9945-a29acf63cf45",
      "name": "lolo",
      "type": "git",
      "url":
"https://helixteamhub.com/example/projects/test/repositories/lolo",
      "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/lolo",
      "ssh_url":
"hth@helixteamhub.com:hth/projects/test/repositories/git/lolo",
      "owner": {
        "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
        "name": "admin",
        "email": "chuck@example.com"
      }
```

```
    }
  }
}
```

Example JSON payload for a *new build completed* webhook:

```
{
  "operation": "completed",
  "type": "build",
  "code_reviews": [
    {
      "uuid": "d9fb853c-cb2e-4f45-962a-b32f2d0f7948",
      "url": "https://helixteamhub.com/example/projects/test/reviews/7",
      "number": 7,
      "state": "open",
      "merge_state": "initial",
      "title": "sdf",
      "description": "",
      "created_at": "2017-11-15T10:35:07Z",
      "updated_at": "2017-11-15T10:35:07Z",
      "deleted_at": null,
      "require_build": false,
      "require_task_comments": false,
      "voting_threshold": 0,
      "base_branch": "test",
      "base_branch_type": "branch",
      "base_commit_id": "dd2b053486ab60ab1d3c4ac6c9b0669f85882894",
      "head_branch": "master",
      "head_branch_type": "branch",
      "head_commit_id": "8e399e9d20df2e6bb74778cfc135510aac65b1de",
      "creator": {
        "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
        "name": "admin",
        "display_name": "Admin"
      },
      "reviewers": [],
      "voters": [],
```

```
      "project": {
        "uuid": "372b0f0b-75bf-49c1-b9be-85aa47cdc6e7",
        "name": "test",
        "url": "https://helixteamhub.com/example/projects/test"
      },
      "repository": {
        "uuid": "45b71dd0-3ece-4862-9945-a29acf63cf45",
        "name": "lolo",
        "type": "git",
        "url":
"https://helixteamhub.com/example/projects/test/repositories/lolo",
        "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/lolo",
        "ssh_url":
"hth@helixteamhub.com:hth/projects/test/repositories/git/lolo",
        "owner": {
          "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
          "name": "admin",
          "email": "chuck@example.com"
        }
      }
    }
  ]
}
```


Code review comments hooks

A separate group of events is specific to code review comments:

- **`code_review_comment`** - comment created
- **`code_review_comment_reply`** - comment reply created
- **`code_review_line_comment`** - line comment created
- **`code_review_line_comment_reply`** - line comment reply created

Example JSON payload for a *new code review comment created* webhook:


```
{
  "operation": "created",
```

```
"type": "code_review_comment",
"code_review": {
  "uuid": "d9fb853c-cb2e-4f45-962a-b32f2d0f7948",
  "url": "https://helixteamhub.com/example/projects/test/reviews/7",
  "number": 7,
  "state": "open",
  "merge_state": "initial",
  "title": "sdf",
  "description": "",
  "created_at": "2017-11-15T10:35:07Z",
  "updated_at": "2017-11-15T10:35:07Z",
  "deleted_at": null,
  "require_build": false,
  "require_task_comments": false,
  "voting_threshold": 0,
  "base_branch": "test",
  "base_branch_type": "branch",
  "base_commit_id": "dd2b053486ab60ab1d3c4ac6c9b0669f85882894",
  "head_branch": "master",
  "head_branch_type": "branch",
  "head_commit_id": "8e399e9d20df2e6bb74778cfc135510aac65b1de",
  "creator": {
    "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
    "name": "admin",
    "display_name": "Admin"
  },
  "reviewers": [],
  "voters": [],
  "project": {
    "uuid": "372b0f0b-75bf-49c1-b9be-85aa47cdc6e7",
    "name": "test",
    "url": "https://helixteamhub.com/example/projects/test"
  },
  "repository": {
    "uuid": "45b71dd0-3ece-4862-9945-a29acf63cf45",
```

```
      "name": "lolo",
      "type": "git",
      "url":
"https://helixteamhub.com/example/projects/test/repositories/lolo",
      "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/lolo",
      "ssh_url":
"hth@helixteamhub.com:hth/projects/test/repositories/git/lolo",
      "owner": {
        "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
        "name": "admin",
        "email": "chuck@example.com"
      }
    }
  },
  "comment": {
    "id": "5a0c1e34222b8037f6a37433",
    "type": "code_review",
    "author": {
      "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
      "name": "admin",
      "display_name": "Admin"
    },
    "content": "abc",
    "created_at": "2017-11-15T11:00:04Z",
    "updated_at": "2017-11-15T11:00:04Z",
    "task_comment": false,
    "task_status": null,
    "task_resolved_at": null,
    "task_resolver": null
  }
}
```

Example JSON payload for a *new code review line comment created* webhook:

```
{
  "operation": "created",
```

```
"type": "code_review_line_comment",
"code_review": {
  "uuid": "d9fb853c-cb2e-4f45-962a-b32f2d0f7948",
  "url": "https://helixteamhub.com/example/projects/test/reviews/7",
  "number": 7,
  "state": "open",
  "merge_state": "initial",
  "title": "sdf",
  "description": "",
  "created_at": "2017-11-15T10:35:07Z",
  "updated_at": "2017-11-15T10:35:07Z",
  "deleted_at": null,
  "require_build": false,
  "require_task_comments": false,
  "voting_threshold": 0,
  "base_branch": "test",
  "base_branch_type": "branch",
  "base_commit_id": "dd2b053486ab60ab1d3c4ac6c9b0669f85882894",
  "head_branch": "master",
  "head_branch_type": "branch",
  "head_commit_id": "8e399e9d20df2e6bb74778cfc135510aac65b1de",
  "creator": {
    "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
    "name": "admin",
    "display_name": "Admin"
  },
  "reviewers": [],
  "voters": [],
  "project": {
    "uuid": "372b0f0b-75bf-49c1-b9be-85aa47cdc6e7",
    "name": "test",
    "url": "https://helixteamhub.com/example/projects/test"
  },
  "repository": {
    "uuid": "45b71dd0-3ece-4862-9945-a29acf63cf45",
```

```
      "name": "lolo",
      "type": "git",
      "url":
"https://helixteamhub.com/example/projects/test/repositories/lolo",
      "https_url":
"https://helixteamhub.com/example/projects/test/repositories/git/lolo",
      "ssh_url":
"hth@helixteamhub.com:hth/projects/test/repositories/git/lolo",
      "owner": {
        "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
        "name": "admin",
        "email": "chuck@example.com"
      }
    }
  },
  "comment": {
    "id": "5a0c1e34222b8037f6a37433",
    "type": "code_review",
    "author": {
      "uuid": "9db5061d-42a9-4a1b-8115-0f673c24516f",
      "name": "admin",
      "display_name": "Admin"
    },
    "content": "abc",
    "created_at": "2017-11-15T11:00:04Z",
    "updated_at": "2017-11-15T11:00:04Z",
    "line_number": 1,
    "computed_line_number": 1,
    "line_type": "added",
    "outdated": false,
    "diff_handle": [
      { "type": "range", "content": "@@ -0,0 +1,1 @@", "old_start_line":
0, "new_start_line": 1 },
      { "content": "+d213sf", "type": "added" }
    ],
```

```
    "task_comment": false,
    "task_status": null,
    "task_resolved_at": null,
    "task_resolver": null
  }
}
```

Restricting hook execution

TeamHub executes all active webhooks by default. You can restrict the execution of repository hooks by using case-insensitive regular expressions on reference and path. When set, the hook executes only if the *reference pattern* matches a branch or a bookmark, and/or if any of the changed paths matches the *path pattern*.

For examples, to execute a hook only:

- When `master` or `develop` branches are updated:

  Set reference pattern as `^(master|develop)$` and leave path pattern empty.
- When javascript files are changed:

  Leave reference pattern empty and set path pattern as `.*\.js$`.
- On feature branches changing paths matching `tests`:

  Set reference pattern as `^feature` and path pattern as `tests`.

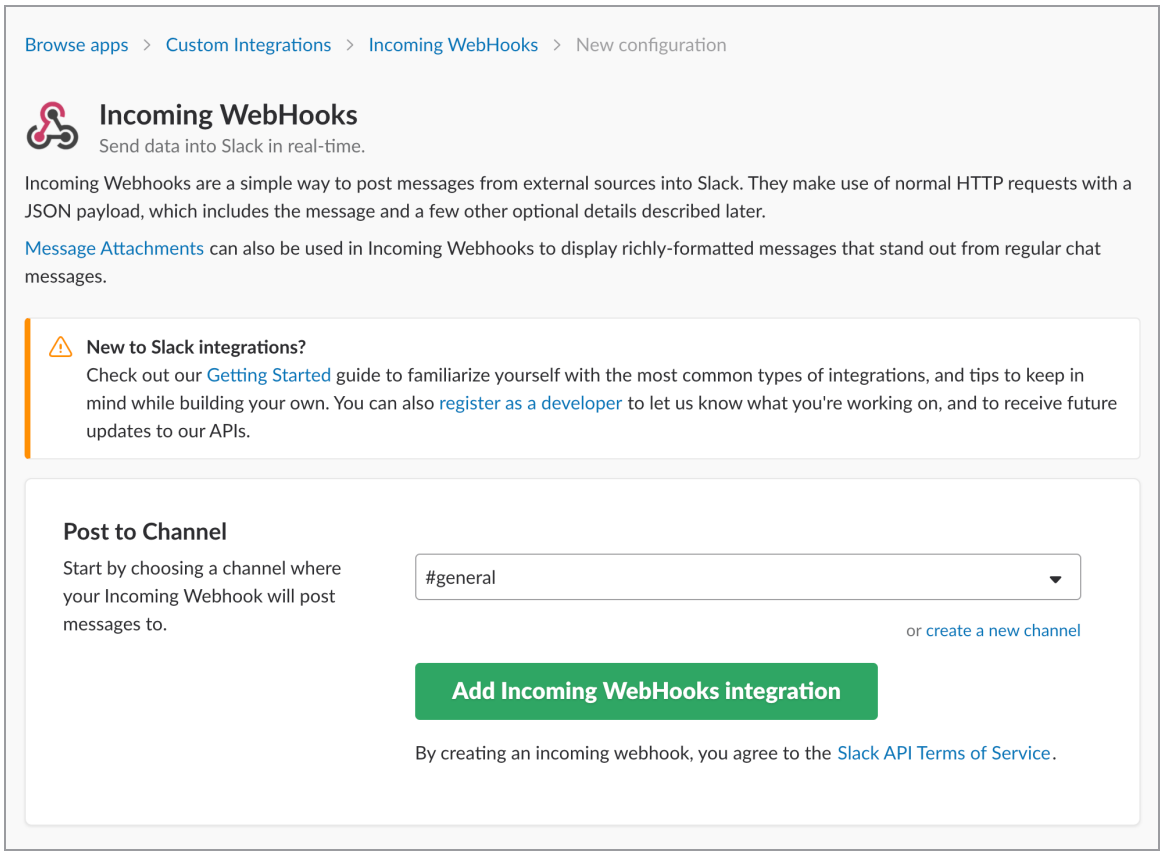
Slack

Slack is a team communication and collaboration tool for both small businesses and large enterprises. Slack integrates with many products available on the market. Adding apps to Slack lets your team stay coordinated and work faster within the context of your conversations. Helix TeamHub harnesses the power of Slack productivity and offers communication over the dedicated webhook.

Setting up a webhook

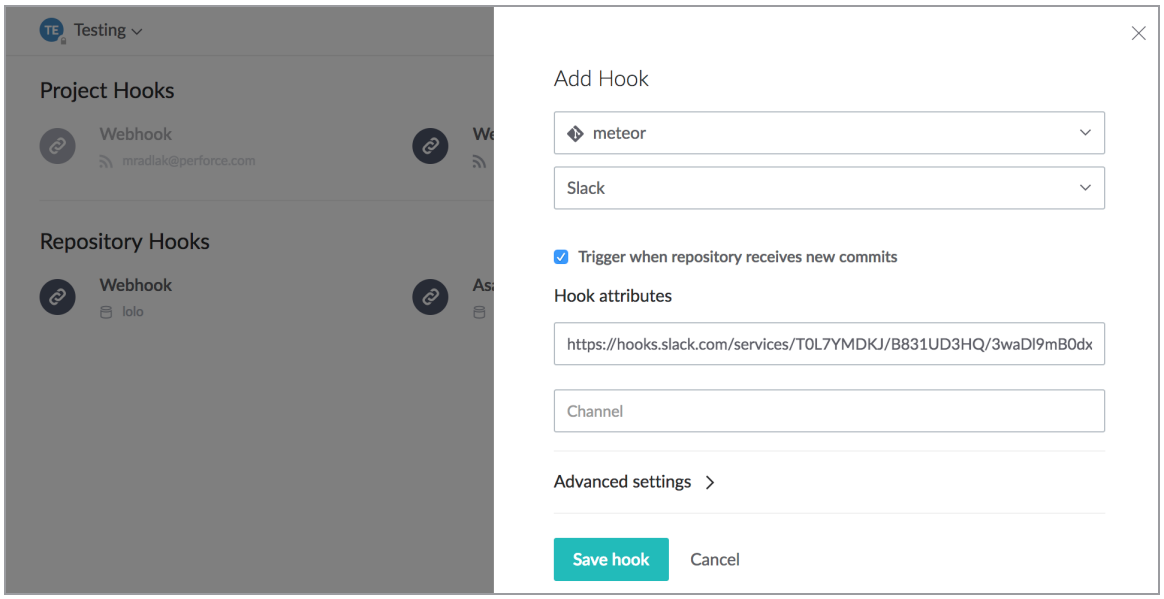
To set up a Slack webhook:

1. Read the official Slack documentation on Incoming Webhooks.
2. In Slack, add a new configuration for incoming webhooks:



a. Select the Slack channel to which the new Incoming Webhook should post messages.
b. Click **Add Incoming WebHooks integration** to proceed.
c. Copy the **Webhook URL**.

3. In Helix TeamHub, in the **Add Hook** tab, click the plus icon to add a new hook.
4. In the **Add Hook** form, do the following:
   a. Select the required repository.
   b. Select **Slack** as the hook.
   c. Under **Hook attributes**, in the **Webhook url** field, paste the **Webhook URL** that you copied from Slack.
   d. (Optional) Specify the channel to which to direct calls.
5. Click **Save hook**. Details of new commits are now posted to the configured slack channel.



JIRA

JIRA is a well-known issue tracking tool by Atlassian. Many large organizations use it for tracking work at multiple levels. JIRA offers limitless possibilities for configuring workflows and levels of work to match even the most complicated needs enterprises might have.

By comparison, issue tracking in Helix TeamHub takes on a much simpler approach. TeamHub strives to solve even the most complex problems with the simplest solution possible.

This section describes how to set up a JIRA hook to use JIRA smart commits.

Smart commits

Helix TeamHub supports interoperability between the code commits made to Helix TeamHub's Git, Subversion or Mercurial repositories and tickets that are maintained in JIRA. The interoperability between TeamHub and JIRA happens through smart commits, which allow processing JIRA issues with commit messages. Smart commits allow developers to perform actions such as transitioning or commenting on JIRA issues by embedding specific commands into commit messages.

With Helix TeamHub smart commits, you can:

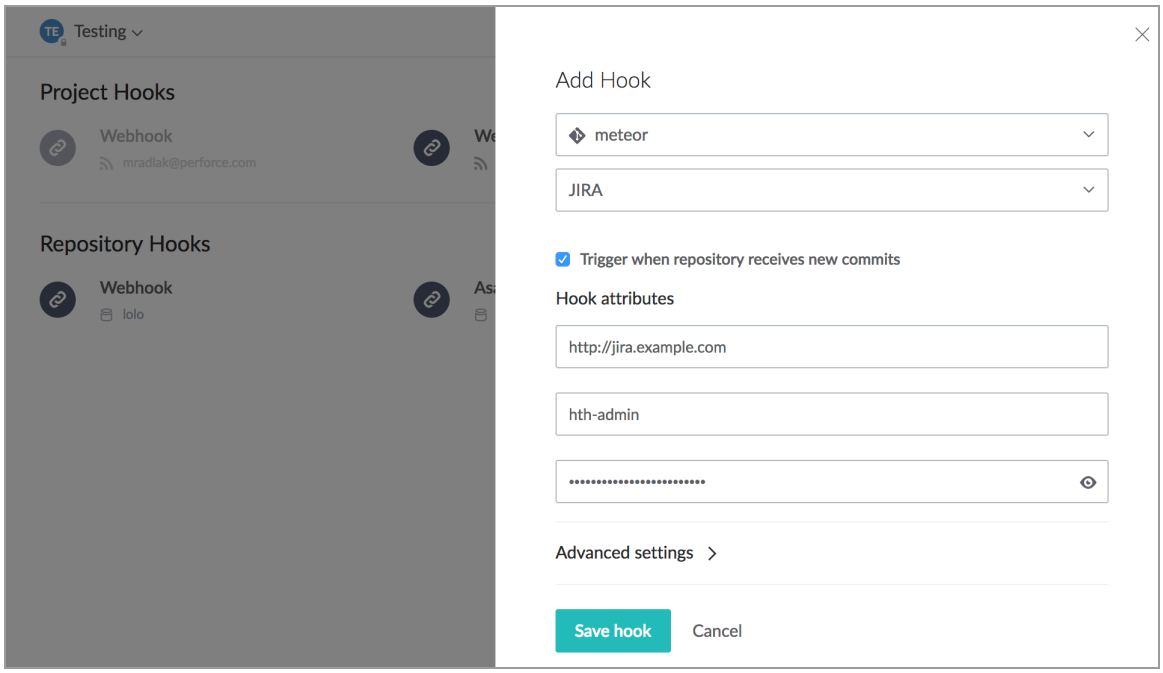
- Comment on issues.
- Record time tracking information against issues.
- Transition issues to any status defined in the JIRA project's workflow.

You can also:

- Add, set, or remove a version and fixed version, components, and labels for a given issue.
- Set the assignee, description, priority, reporter, summary, or resolution for an issue.

Setting up smart commits

Smart commits work through a post-commit hook that you add and configure against a given repository and JIRA server from the **Hooks** tab in a TeamHub project. You can add hooks to any version control system repository.



To add a JIRA hook, you need the following information:

- The URL of the JIRA server
- Credentials to edit the project that the smart commits reference.

  For different JIRA and TeamHub projects, different credentials might be required.

Smart commit commands basics

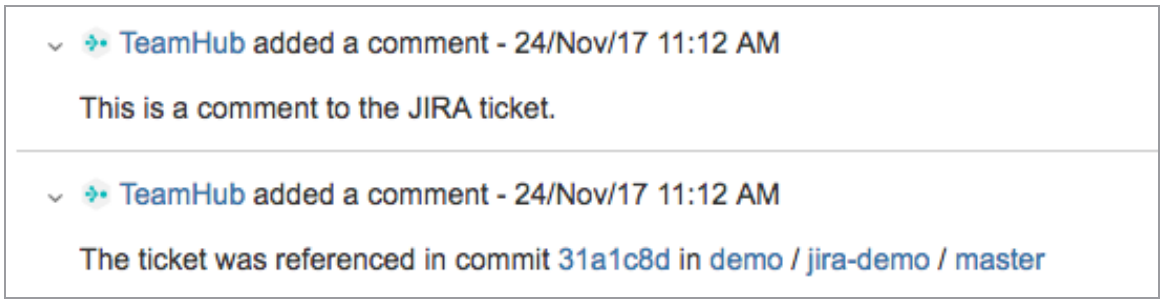
Smart commits are read from commit messages, which follow this basic syntax:

```
<any text> <ISSUE_KEY> <any text> #<COMMAND> <optional COMMAND_ARGUMENTS>
```

Following is a simple example for commenting on a JIRA issue with a commit:

```
JR-456 #comment this is a comment to the JIRA ticket.
```

The commit above references the JIRA issue and also leaves a comment inside the issue. The following figure illustrates the comment specified with the `#comment` verb as well as the separate reference that links the JIRA issue to the respective commit. You can click the link to open the code changes in TeamHub.

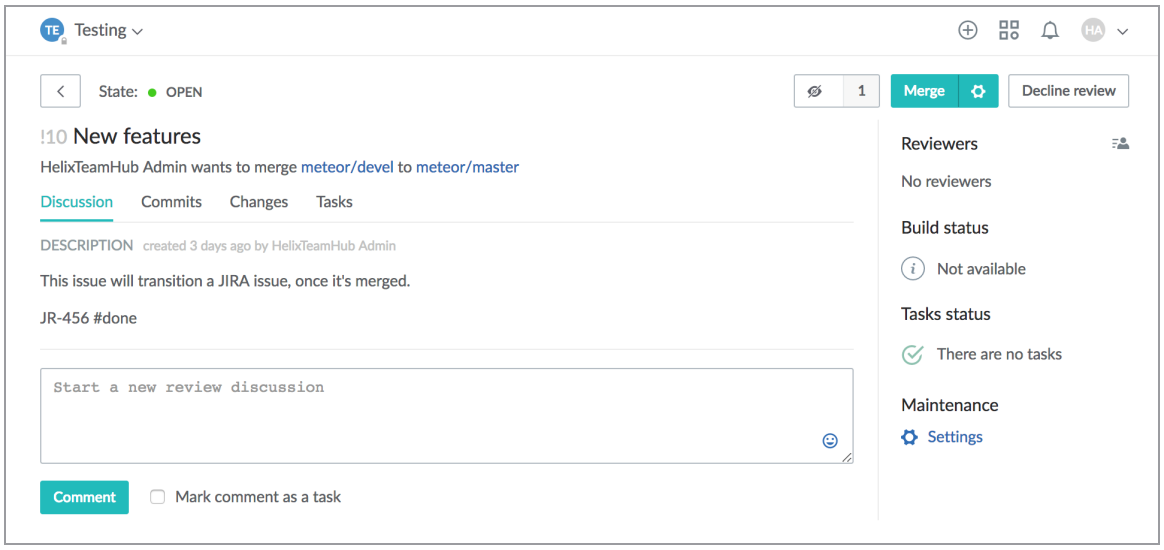


The following commit provides a more complex usage example: It transitions the issue, comments on it, and adds more time.

```
JR-456 #to-do #comment Still not working #time 1d 4.5h 10m It takes too much time!
```

Connecting JIRA smart commits to code reviews.

You can also leverage JIRA smart commits for the daily development workflow by having JIRA issues transition from one state to another when a code review in Helix TeamHub is merged. To accomplish this, you need to add a JIRA smart commit transition to the description of the code review. The following screenshot illustrates a `done` transition for JIRA issue #JR-456.



This works because in Helix TeamHub, the code review description is always included in the merge commit that is created when the code review is merged. Because the description contains the transition information, the JIRA hook works as it would work for any other commit, as does the transition.

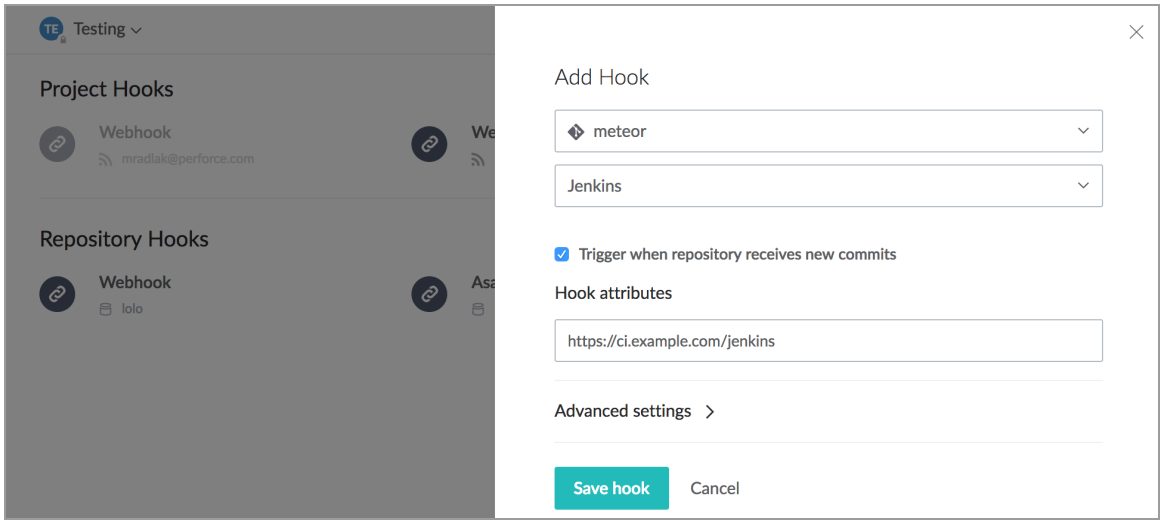
Jenkins

Jenkins is a self-contained open source automation server that you can use to automate tasks related to building, testing, and deploying software.

Jenkins webhook

When using the standard Jenkins Git Plugin, Jenkins Mercurial Plugin, or Jenkins Subversion Plugin to poll and check out your repository, you can quickly and easily switch to a push model using the Jenkins webhook.

To configure the Jenkins webhook in Helix TeamHub, simply provide a Jenkins instance URL in the **Add Hook** form for a repository.



Helix TeamHub Jenkins plugin

The Helix TeamHub Jenkins plugin integrates Jenkins build events to Helix TeamHub and lets you use them in code review workflows. For a step-by-step description of the setup process, see the Configuring builds with Jenkins instructions.

Wiki

> **Note**
> Helix TeamHub supports Mercurial and Git based Wikis.

A Wiki is a repository that you can modify locally. It lets you store project-specific documentation for your team or project, allowing you to keep everything in the same place as your source code and other assets. The Wiki enables you to:

- See changes in real time with the side-by-side editor.
- Look through versions to find the right data.
- Add attachments, include pictures and charts, and insert links.
- Manage large projects.

Wikis support markdown syntax.

Add a Wiki to an existing project

If a project is created without a Wiki, you can add one from the project. Only available if at least one of the supported Wiki repository types is enabled.

1. At the project scope, in the left pane, click **Wiki**.
2. Select the Wiki repository type from the **Select Wiki type:** dropdown.

   > **Tip**
   > Not displayed if only one Wiki repository type is available.

3. Click the **Create Wiki** button.

   Helix TeamHub creates the Wiki for the project.

Add or edit a page

1. At the project scope, in the left pane, click **Wiki**.
2. In the **Wiki** view, do one of the following:
   - To add a page, click the plus icon  .
   - To edit a page, click the **Edit page** button.

   The page form opens in **Side-by-side** view, showing the content editor pane on the left and the preview pane on the right. If you prefer to only see one of these panes at a time, click **Tabs**.
3. For a new page, enter a page title.

4. In the **Editor** pane, enter or edit content as needed.

   In **Side-by-side** view, as you type, the **Preview** pane shows how TeamHub will render the content.

   To get help on markdown syntax, click the **Syntax** link at the bottom of the editor. Syntax information displays on the right.

5. To enrich your content, click any of the links at the bottom of the editor:
   - **Attachments** to include images, attach files, or link to other Wiki pages
   - Smile icon  to insert emoticons
6. Click **Save**.

Delete a page

1. On the page you want to delete, click the trash can icon  .
2. When prompted for confirmation, click **OK**.

   TeamHub removes the Wiki page.

Helix TeamHub CLI tool - Technology preview feature

Important
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
Helix TeamHub supports Helix TeamHub CLI for Git and Helix Git repositories.

The Helix TeamHub CLI tool enables you to carry out common actions across multiple repositories without leaving the command line. Automation of the hth-cli commands is supported by the use of flags and piping.

For details on the licensing of hth-cli, see "Helix TeamHub CLI licensing" on page 169.

Supported operating systems:

- Linux
- Apple Mac
- Windows

This chapter contains:

Getting started with Helix TeamHub CLI

This section will help you to get started with the Helix TeamHub CLI tool. It covers installation, how to setup the hth-cli tool, and the first few steps required to start performing actions on repositories within your project.

Installation and setup

Prerequisites

- The Helix TeamHub CLI tool relies on Git for the repository operations. Git must be installed before you install the hth-cli tool. Download Git for your Operating System from https://git-scm.com/downloads.
- Helix TeamHub CLI uses the SSH protocol for repository operations. Please make sure that you have SSH keys configured for your account, see "Configuring SSH keys" on page 18.

> **Tip**
> To ensure hth-cli compatibility on Windows, please make sure you're using the Git Bash command prompt.

Installation

1. Download the hth-cli binary from the Helix TeamHub section of the Perforce download page.
2. Setup the hth-cli tool, see "Setup the hth-cli tool" below.

Setup the hth-cli tool

In this section we will create the `.hth/cli` directory and a global config file by running `hth setup` without flags. You will be prompted to enter your instance, company, and user details.

The location the `.hth/cli` directory is created in will depend on the operating system you are using:

- **Linux:** created in the home directory `~/`
- **Apple Mac:** created in the home directory `~/`
- **Windows:** created in `C:\Users\<username>`

> **Tip**
> Run `hth setup` with flags to specify your instance, company, and user details. For details of the `hth setup` flags, see "Setup hth-cli command" on page 158.

1. Setup the hth-cli tool to connect to your Helix TeamHub account:

```
  hth setup
  Setting up hth-cli.
  Input details of your Helix TeamHub installation.
? Instance URL https://my.helixteamhub.com
? Company ID helix20201
? User ID admin
✓ The hth-cli has been successfully set up under /Users/bob/.hth/cli
  Run hth config edit --global to edit your input
```

2. To create an active session for the subsequent commands, login to your account using the **`hth login`** command:

```
  hth login
Logging into https://my.helixteamhub/helix_20201...
? User ID admin
? Password
✓ Session created successfully
```

> **Tip**
> You can invalidate the session at any time by using **`hth logout`**.

3. Open your browser and navigate to the project you want to manage from the Helix TeamHub Web UI.
4. Create a new repository and select the **Manifest** and **Generate a default manifest file** checkboxes.

   This creates a manifest repository and populates your existing repositories with the required manifest configuration that ties the repositories together.
5. Setup your local working directory, see "Set up a local working directory" below.

Set up a local working directory

A local working directory contains a set of repositories within a Helix TeamHub project.

1. From the command line, navigate to the path you want to work in and setup your local working directory.
2. To clone the manifest repository to your local working directory, run **`hth init`** using the clone URL of your manifest repository:

```
  hth init ssh://git@gconn.my.helixteamhub.com:220/hth_cli/manifest
Cloning into '/Users/bob/Projects/hth/hth-cli-
testing/.hth/cli/manifest'...
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 4 (delta 0)
Receiving objects: 100% (4/4), done.
```

3. Run **`hth sync`** to clone all the repositories configured in the manifest file under the current path:

```
  hth sync
✓ Successfully cloned backend from
ssh://git@my.helixteamhub.com/hth_cli/backend
✓ Successfully cloned frontend from
ssh://git@my.helixteamhub.com/hth_cli/frontend
✓ Successfully cloned docs from ssh://git@my.helixteamhub.com/hth_
cli/docs
```

4. You are now all set to start using hth-cli, enjoy!
   - For information about how and where hth-cli stores its manifest and config files, see "Manifest and config files" below.
   - For examples of how to manage your repositories using hth-cli, see the "Helix TeamHub CLI examples" on the facing page section on what to do next.
   - For information about the hth-cli commands, see "Helix TeamHub CLI command reference" on page 154.

Manifest and config files

Helix TeamHub CLI stores its configuration on two levels:

- **Global config:** stored in your home directory (Linux and MacOS: **`~/`**, Windows: **`C:\Users\<username>`**), under **`.hth/cli`**. You can change the path to the global config file by setting the **`HTH_CLI_ROOT`** environment variable. This directory contains the following files:
  - **`api_keys.json`** - used to store active sessions (your login password is never stored)
  - **`config.json`** - used to store configuration regarding the Helix TeamHub instances, companies and users
- **Local config:** stored in your local working directory, under **`.hth/cli`**. This directory contains the following files:

- **`config.json`** - contains local overrides for the global config.json configuration
- **`manifest`** - the manifest repository for the local working directory

Helix TeamHub CLI examples

This section contains examples of how you can manage your repositories using hth-cli.

In this section:

Code Review Management

Code reviews are managed using the **`hth code-review`** command. The commands can be run inside your local working directory.

> **Tip**
> **`hth code-review`** can be shortened to **`hth cr`**.

Create a code review

To create a code review for the current branch, use the **`hth code-review create`** command:

```
  hth code-review create
? Destination branch master
? Title Add new feature
? Description (optional)
  Creating a code review 'Add new feature' out of the following repository
         backend (new-feature > master) for the hth_cli project located at
https://my.helixteamhub.com/acme/projects/hth_cli
? Do you want to continue? (y/n) y
✓ Code review created successfully. You can access it at
https://my.helixteamhub.com/acme/projects/hth_cli/reviews/1
```

> **Tip**
> You can use flags for further customization like selecting the source reference. For more information on the code review commands, see "Code Review and Multi-repo Code Review commands" on page 156.

List code reviews

To list the open code reviews for your current project, use the **`hth code-review list`** command:

```
  hth cr list
  Listing the code reviews for the hth_cli project located at
https://my.helixteamhub.com/acme/projects/hth_cli.
       !1 Add new feature
```

Merge a code review

To merge a code review:

1. Find the code review identifier (number) by, for example, using the **`hth code-review list`** command.
2. Run the **`hth code-review merge`** command:

   ```
     hth code-review merge 1
     Merging the code review !1 for the hth_cli project located at
   https://my.helixteamhub.com/acme/projects/hth_cli.
   ✓ The code review merged successfully.
   ```

3. That's it, you have now merged the code review!

Multi-Repo Code Review Management

> **Note**
> Helix TeamHub only supports Multi-repo code reviews for Helix Git repositories.

When developing software, it is common to have related changes that affect multiple repositories. Helix TeamHub's Multi-Repo Code Review feature allows you to track changes between multiple repositories, and merge the changes as an atomic transaction.

Multi-Repo Code Reviews are managed using the **`hth multi-repo-code-review`** command. The commands can be run inside your local working directory.

> **Tip**
> **`hth multi-repo-code-review`** can be shortened to **`hth mrcr`** or **`hth mcr`**.

Create a multi-repo code review

To create a multi-repo code review from the references you have currently checked out, run the **`hth mcr create`** command in your local working directory:

```
  hth mcr create
? Destination branch of 'backend' repository master
? Destination branch of 'frontend' repository master
? Destination branch of 'infra' repository master
? Title Feature XYZ
? Description (optional)
  Creating a multi-repo code review 'Feature XYZ' out of the following
repositories for the hth_cli project located at
https://my.helixteamhub.com/acme/projects/hth_cli
        backend (feature-xyz > master)
        frontend (feature-xyz > master)
        infra (feature-xyz > master)
? Do you want to continue? (y/n) y
✓ Multi-repo code review created successfully. You can access it at
https://my.helixteamhub.com/acme/projects/hth_cli/multi-reviews/1
```

Tip
You can use flags for further customization like selecting a custom set of repositories/sources references. For more information on the multi-repo code review commands, see "Code Review and Multi-repo Code Review commands" on page 156.

List multi-repo code reviews

To list open multi-repo code reviews for the project, use the **`hth mcr`** list command in your local working directory:

```
  hth mcr list
  Listing the code reviews for the hth_cli project located at
https://my.helixteamhub.com/acme/projects/hth_cli.
        %1 Feature XYZ
```

Merge a multi-repo code review

To merge a multi-repo code review:

1. Find the code review identifier (number) by, for example, using the **`hth mcr list`** command.
2. Run the **`hth mcr merge`** command:

```
  hth mcr merge 1
  Merging the multi-repo code review %1 for the hth_cli project
located at https://my.helixteamhub.com/acme/projects/hth_cli
✓ The multi-repo code review merged successfully.
```

3. That's it, you have now merged the multi-repo code review!

Run commands across repositories using the "hth each" command

hth-cli enables you to run commands across a set of repositories using the **`hth each`** command.

Checkout a new branch

To check out a new branch in Git, prefix the Git branch creation command with **`hth each`**:

```
  hth each git checkout -b new-feature
Running the command for backend
Switched to a new branch 'new-feature'
✓ Command succeeded.
Running the command for frontend
Switched to a new branch 'new-feature'
✓ Command succeeded.
Running the command for docs
Switched to a new branch 'new-feature'
✓ Command succeeded.
```

Add files to multiple repositories

To create a file under each repository, run the following command using **`hth each`**:

```
  hth each 'echo "2020.1" > VERSION'
```

Staging files:

```
  hth each "git add . && git status"
Running the command for backend
On branch new-feature

No commits yet
```

```
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   VERSION

✓ Command succeeded.
Running the command for frontend
On branch new-feature

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   VERSION

✓ Command succeeded.
Running the command for docs
On branch new-feature

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   VERSION

✓ Command succeeded.
```

Create commits

To create commits using **`hth each`**:

```
  hth each "git commit -m 'Update VERSION'"
Running the command for backend
[new-feature (root-commit) 5bc5281] Update VERSION
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
✓ Command succeeded.
Running the command for frontend
```

```
[new-feature (root-commit) 5bc5281] Update VERSION
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
✓ Command succeeded.
Running the command for docs
[new-feature (root-commit) 5bc5281] Update VERSION
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
✓ Command succeeded.
```

Push to remote

To push to remote using **`hth each`**:

```
  hth each git push -u origin new-feature
Running the command for backend
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 226 bytes | 226.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://my.helixteamhub.com/hth_cli/backend
 * [new branch]      new-feature -> new-feature
Branch 'new-feature' set up to track remote branch 'new-feature' from
'origin'.
✓ Command succeeded.
Running the command for frontend
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 226 bytes | 226.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://my.helixteamhub.com/hth_cli/frontend
 * [new branch]      new-feature -> new-feature
Branch 'new-feature' set up to track remote branch 'new-feature' from
'origin'.
✓ Command succeeded.
Running the command for docs
Enumerating objects: 3, done.
```

```
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 226 bytes | 226.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://my.helixteamhub.com/hth_cli/docs
 * [new branch]      new-feature -> new-feature
Branch 'new-feature' set up to track remote branch 'new-feature' from
'origin'.
✓ Command succeeded.
```

Helix TeamHub CLI command reference

This section details the commands available to you and how to use them.

Common flags available for all commands:

- To list all of the commands available to you from the command line, enter **`hth --help`** or **`hth -h`**.
- To get help on a specific command and its flags from the command line, enter **`hth command --help`** or **`hth command -h`** where ***`command`*** is the command you need help on.
- To run any command with verbose output (debug output), add the **`--verbose`** flag to the command.

Write operations request a confirmation

Any write operation such as; **`hth pull`**, **`hth commit`**, **`hth push`**, and so on will ask for a confirmation as shown below:

> **Note**
> This may be configurable at the project or global level.

```
Pushing/Pulling/Commiting changes for:
    backend#develop
    omnibus-acme#develop
    foo#topic2
to `myhth.com/acme/projects/platform`

Do you want to continue? [y/n]
```

Helix TeamHub CLI commands:

Login and Logout commands

This section describes the Helix TeamHub CLI commands used to login and logout of Helix TeamHub.

hth login or hth session create

Use one of the following commands to login to TeamHub. Both commands work in the same way and have the same flags:

- `hth login`
- `hth session create`

Flags

Flags	Description
`-i` or `--instance`	Specify the TeamHub instance URL to login to.
`-c` or `--company`	Specify the TeamHub company ID to login to.
`-u` or `--user`	Specify the TeamHub user ID to log in with.
`-p` or `--password`	Specify the password for the user
`--password-stdin`	Read the password from a pipe

When run inside a project directory

hth-cli uses instance and company names from the manifest file and looks for matching user short names from the global config file. Select a user from the list or input a user and enter the password.

When running outside of a project directory

1. If Helix TeamHub instances are configured in the global config, hth-cli asks you to select one or provide an URL to the Helix TeamHub instance.
2. If company names are configured in the global config, hth-cli asks you to select one or provide a company name.
3. If users are configured in the global config, hth-cli asks you to select one or provide a username. Input the password.

hth logout or hth session delete

Use one of the following commands to logout of TeamHub. Both commands work in the same way and have the same flags:

- **`hth logout`**
- **`hth session delete`**

Flags

Flags	Description
`-i` or **`--instance`**	Specify the TeamHub instance URL to logout from.
`-c` or **`--company`**	Specify the TeamHub company ID to logout from.
`-u` or **`--user`**	Specify the TeamHub user ID to logout.

Code Review and Multi-repo Code Review commands

This section describes the code review and multi-repo code review commands that let you manage code reviews.

hth code-review

`hth code-review` lets you manage code reviews for the current repository.

> **Tip**
> The **`hth code-review`** can be shortened to **`hth cr`**.

Commands

- **`create`** create a new code review
- **`close`** close a code review
- **`list`** list code reviews
- **`merge`** merge a code review
- **`reopen`** reopen a code review
- **`revert`** revert a code review
- **`view`** view a code review

Flags

There are no flags available for **`hth code-review`**.

hth multi-repo-code-review

> **Note**
> Helix TeamHub only supports Multi-repo code reviews for Helix Git repositories.

`hth multi-repo-code-review` lets you manage multi-repo code reviews for the repositories defined in the manifest file.

> **Tip**
> The **`hth multi-repo-code-review`** can be shortened to **`hth mrcr`** or **`hth mcr`**.

Commands

- **`create`** create a new multi-repo code review
- **`close`** close a multi-repo code review
- **`list`** list multi-repo code reviews
- **`merge`** merge a multi-repo code review
- **`revert`** revert a multi-repo code review
- **`view`** view a multi-repo code review

Flags

There are no flags available for **`hth multi-repo-code-review`**.

Scope commands

`hth scope` lets you manage the scope.

> **Tip**
> For more information about scopes, see "Navigation" on page 12.

Commands

- **`set`** set a scope
- **`show`** show a scope
- **`unset`** unset a scope

Flags

There are no flags available for **`hth scope`**.

Setup hth-cli command

`hth setup` configures hth-cli for the specified Helix TeamHub instance, company, and user.

Commands

The are no commands available for **`hth setup`**.

Flags

Flags	Description
`-i` or **`--instance`**	Set the TeamHub instance URL
`-c` or **`--company`**	Set the company ID
`-u` or **`--user`**	Set the user ID

Manifest commands

`hth manifest` lets you to manage the manifest file.

Commands

- **`download`** download a manifest
- **`edit`** edit a manifest file
- **`show`** show a manifest

Flags

There are no flags available for **`hth manifest`**.

Config commands

`hth config` lets you manage a configuration file.

Commands

- **`edit`** open a configuration file in a default editor
- **`show`** show a configuration

Flags

Flags	Description
`--local`	Uses local config
`--global`	Uses global config

Repository commands

This section details the Helix TeamHub CLI commands for repository sync, init, and the each command that runs a cross-repository command.

hth sync

`hth sync` synchronizes the repositories.

Commands

There are no commands available for **`hth sync`**.

Flags

Flags	Description
`-r` or `--ref`	Use rebase

hth init URL

`hth init URL` creates a local directory.

Commands

There are no commands available for **`hth init URL`**.

Flags

Flags	Description
`-r` or `--ref`	Sets the manifest repository reference
`-f` or `--filename`	Sets the manifest repository filename

Usage example

```
hth init URL
```

hth each

`hth each` runs a command against all of the repositories defined in the manifest file located in the current directory.

For more examples of **`hth each`**, see "Run commands across repositories using the "hth each" command" on page 151

Version command

`hth version` prints the hth-cli version.

Commands

There are no commands available for **`hth version`**.

Flags

There are no flags available for **`hth version`**.

Git commands supported by Helix TeamHub CLI

This section details the Git commands supported by hth-cli.

Supported Git commands

The Git commands in the table are directly supported by hth-cli. hth-cli passes all of the switchers and arguments to the native Git command.

Helix TeamHub command	Git command	Helix TeamHub command	Git command
`hth add`	`add`	`hth push`	`push`
`hth branch`	`branch`	`hth reset`	`reset`
`hth checkout`	`checkout`	`hth show`	`show`
`hth commit`	`commit`	`hth stash`	`stash`
`hth diff`	`diff`	`hth status`	`status`
`hth fetch`	`fetch`	`hth tag`	`tag`
`hth log`	`log`		

Note
The GIT commands in the table are not supported by the "hth each" above command.

Write operations request a confirmation

Any write operation such as; **`hth pull`**, **`hth commit`**, **`hth push`**, and so on will ask for a confirmation as shown below:

> **Note**
> This may be configurable at the project or global level.

```
Pushing/Pulling/Commiting changes for:
    backend#develop
    omnibus-acme#develop
    foo#topic2
to `myhth.com/acme/projects/platform`

Do you want to continue? [y/n]
```

Notifications

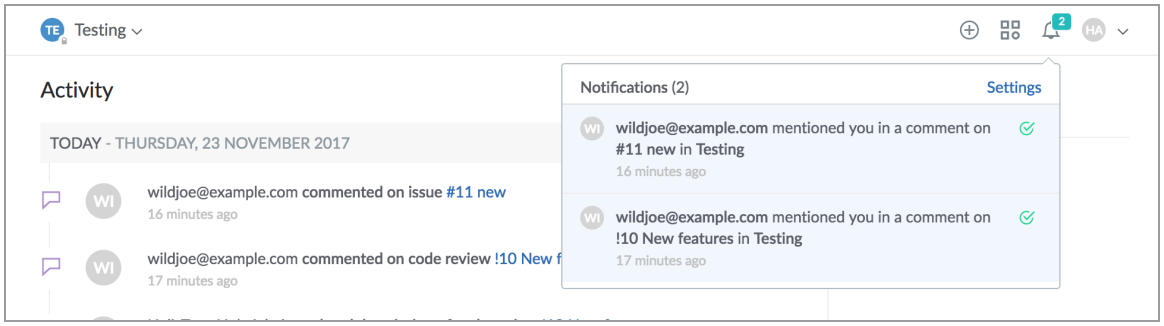
Helix TeamHub creates notifications for events. This section describes the type of notifications you can receive, how to subscribe to them, the events that trigger them, and the notification messages that are created.

Notification types

TeamHub offers the following notifications:

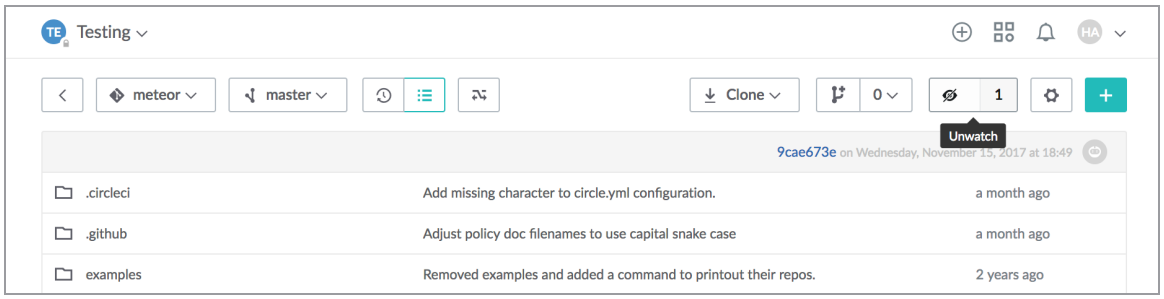
- In-app: Work inside the application
- Email: Are sent to your email address

You can access the latest notifications from the bell icon in the site header.



Clicking a notification opens the source, displaying the notification in context.

To receive notifications from projects, repositories, code reviews, and issues, you need to watch them. To watch an item in TeamHub, go to the respective tab and click the watch icon. To stop watching an item, click the crossed-out watch icon again.



By default, you watch each project, repository, code review, and issue you have created or been added to. In addition, TeamHub subscribes you to watching an item when you leave a comment or push changes to a repository.

Notification events and messages

This section contains a list of the events that trigger notifications and example notification messages:

Attachments

- **Attachment created for a code review:**

  `John` created an attachment `image1.png` in code review `!123`
- **Attachment created for a Multi-Repo Code Review:**

  `John` created an attachment `image1.png` in multi-repo code review `!123`
- **Attachment created for a Task:**

  `John` created an attachment `image1.png` in issue `#123`

Code Reviews

- **Review created:**

  `John` created a code review in `repo1` repository
- **Updated description, state, or title:**
  - `John` updated code review description: `John's updated description text`
  - `John` updated code review state to `approved`
  - `John` changed code review title to `MyReview`
- **Setting the 'require passing build' flag:**
  - `John` enabled build requirement
  - `John` disabled build requirement
- **Setting the 'all tasks to be resolved' flag:**
  - `John` enabled task comments requirement
  - `John` resolved task comments requirement
- **Updated the number of approvals required:**

  `John` changed code review threshold to `10`
- **New commits pushed to a head branch:**

  `John` pushed `5` new commits to `feature-01` branch
- **New reviewer:**
  - `John` joined reviewers
  - `John` added `Kate` to reviewers

- **Reviewer removed:**
  - **John** left reviewers
  - **John** removed **Kate** from reviewers
- **Reviewer voted:**

  John approved the review
- **Reviewer cleared vote:**

  John withdrew their approval

Comments

- **Code review comment:**
  - **John** created the task comment in the code review: **My task comment**
  - **John** commented on the code review: **My comment**
- **Code review file comment:**
  - **John** created the task comment in the code review, **repo1/foo**: **My comment**
  - **John** commented on the code review, **repo1/bar**: **My comment**
- **Code review line comment:**
  - **John** created the task comment in the code review, **repo1/bar:10**: **My comment**
  - **John** commented on the code review, **repo1/foo:55**: **My comment**
- **Repository commit comment:**

  John commented on the commit **0eb219d6a9193251f7ef7d921f5dc96cf15ec25e**: **My comment**
- **Repository commit file comment:**

  John commented on the commit **0eb219d6a9193251f7ef7d921f5dc96cf15ec25e**, **repo1/foo**: **My comment**
- **Repository commit line comment:**

  John commented on the commit **0eb219d6a9193251f7ef7d921f5dc96cf15ec25e**, **repo1/foo:10**: **My comment**
- **Task comment:**

  John commented on the issue: **My comment**

Multi-Repo Code Reviews

- **Created a Multi-Repo Code Review:**

  John created a multi-repo code review in **repo1** project
- **Updated description, state, or title:**
  - **John** updated multi-repo code review description: **My updated description**
  - **John** updated multi-repo code review state to **merged**
  - **John** changed multi-repo code review title to **My review title**

Tasks

- **Created a task:**

  John created issue in **Milestone 1** milestone
- **Updated description, state, or title:**
  - **John** updated issue description: **My updated description**
  - **John** updated issue state to **closed**
  - **John** changed issue title to **My task title**
- **Created/deleted a member:**
  - **John** joined the issue
  - **John** assigned **Kate** to the issue
  - **John** left the issue
  - **John** removed **Kate** from the issue
- **Milestone update:**

  John moved issue to **Milestone 4** milestone

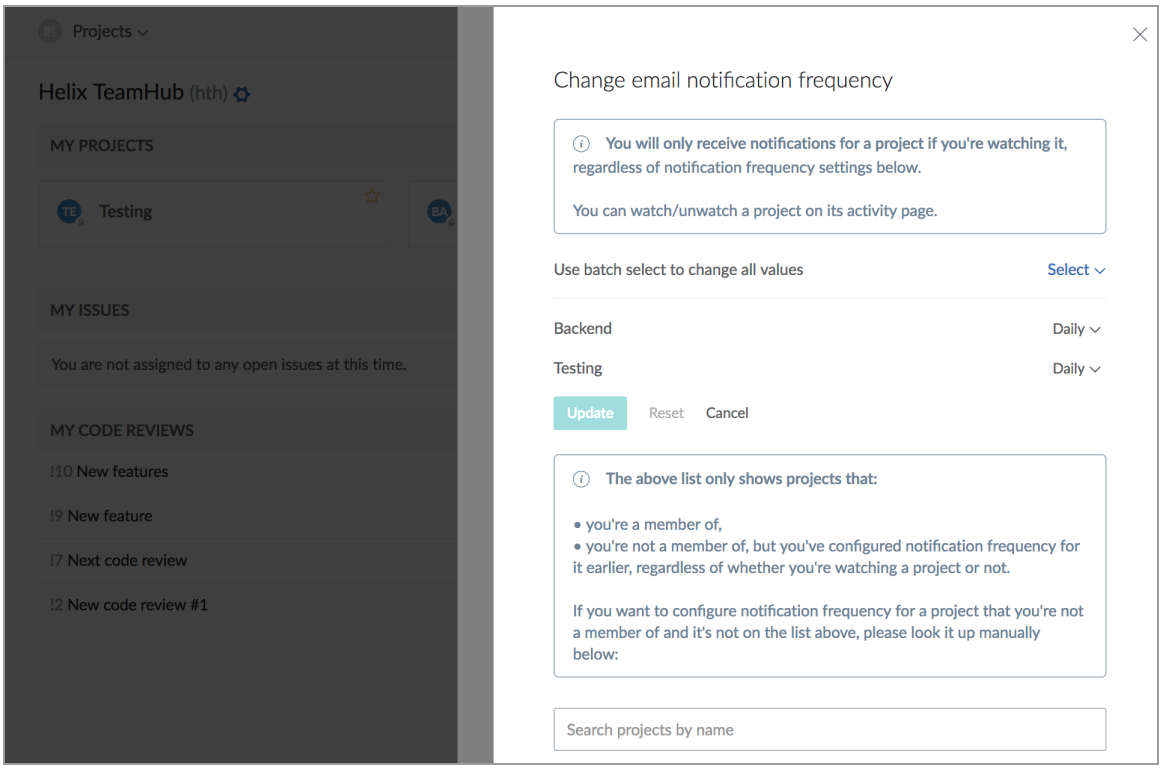
Configuring notification intervals

You can adjust the frequency of email notifications for your subscriptions.

To configure notification intervals:

1. Click the bell icon to display notifications, and then click **Settings**.

   The **Change email notification frequency** form displays a list of all projects you watch along with interval selectors on the right.



2. Change the frequency as needed, either individually or by selecting a value from the **Use batch select to change all values** list.

   The following options are available:

   - **Instant**: Recieve notifications instantly
   - **Daily**: Receive notifications once a day
   - **Weekly**: Receive notifications once a week
   - **Never**: Never receive notifications

3. Click **Update**.

Company Settings

You can maintain and configure your company from the **Company settings** form.

> **Note**
> Only the company administrator has access to this form.

To access the company settings, do one of the following:

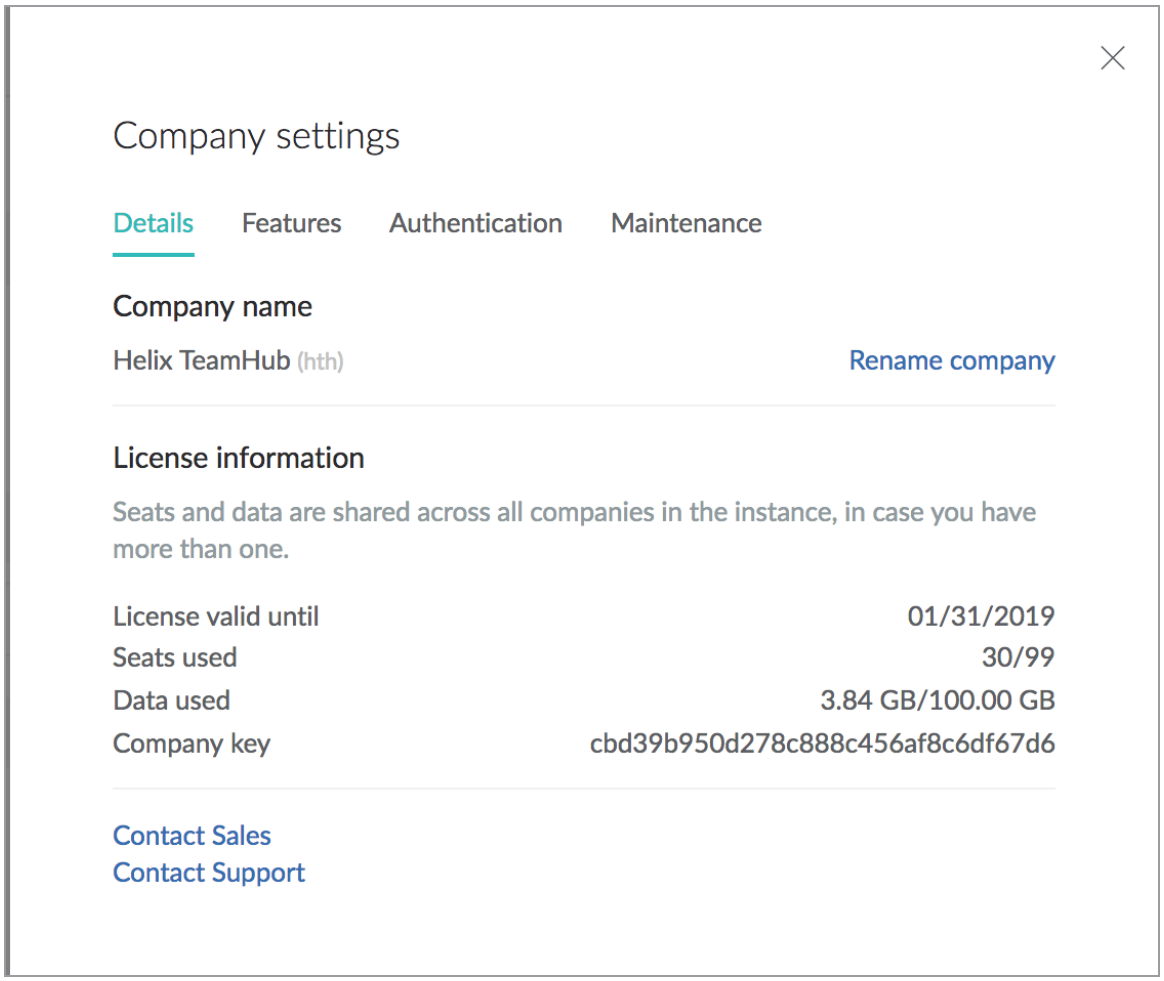
- In the **My Dashboard** view, click the gear icon next to the company name.
- In any view, click the user name in the site header and select **Company settings**.

License information

Helix TeamHub licensing

The **Company settings** form displays your company's license information on the **Details** tab in the **License information** area, including the following information:

- **License valid until**: The date on which your license is set to expire.
- **Seats used**: The total number of active accounts (users and collaborators) in your company/instance and the maximum number of active accounts allowed by the license. For details, see Note.
- **Data used**: The total amount of data in your company/instance and the maximum amount of data allowed by the license. For details, see Note.
- **Company key**: A unique identifier for your company.



To upgrade your license, click **Contact Sales**.

Note

- **Helix TeamHub cloud :** the seat and data usage are gathered from the current company you are logged into. Storage usage is calculated once a day from the active repositories and attachments. Once the storage limit is exceeded, the repositories become read-only. Data use can be decreased by deleting attachments and repositories, or by removing files from repositories. Depending on the repository type, the old files may still remain in the history after removal and consume storage.
- **Helix TeamHub on-premise:** the license covers the whole instance. The seat and data usage are gathered from all of the companies inside the instance. The license does not set restrictions on the maximum amount of data allowed.

Tip

- If required, you can reduce your active storage by deleting repositories, see "Maintenance settings" on page 71.
- When you delete content from your repositories:
  - **Helix Git** repositories are versioned and track change history, repository data is not deleted from the Helix server.
  - **Git, Mercurial, and Subversion** repositories are versioned and track change history. Files committed to the remote repository stay in the history even if you remove them. Those files continue to consume storage.
  - **Ivy, Maven, and WebDAV** repositories are unversioned and do not track history. Removing files frees up storage.
  - **Docker** repositories are unversioned and do not track history. Removing tags frees up storage.

Helix TeamHub CLI licensing

Maximum number of parallel Helix TeamHub CLI sessions allowed:

- **Helix TeamHub Cloud:** one hth-cli session per five seats.
- **Helix TeamHub on-premise:** no limit.

Feature settings

The **Company settings** form displays your company's feature settings on the **Features** tab. The following options are available:

- **Code Search:** Turn on to enable code searching. For details, see "Code search" on page 116.
- **File Sharing:** Turn on to enable file sharing, on by default. For details, see "Tabs" on page 75 in "Code browser views" on page 73.
- **Issues:** Turn on to enable issue tracking.
- **Wiki:** Turn on to enable Wiki support.
- **Version control systems:** Turn on the version control system or systems for which you want to enable repository creation. The settings have no effect on existing repositories.
- **Limit project creation to company admins:** Turn on to prevent users with non-admin roles from creating new projects.
- **Automatic depot creation:** Only available with Helix authentication. Turn on to include the **Automate depot creation and permission management** option in the **New project** form. This option enables TeamHub to automatically create a depot and appropriate groups for a new project

and grant permissions to the user **`gconn-user`**. See also "Projects" on page 26 and "Creating Git repositories stored in Helix server" on page 43.

- **Enable repository creation for project masters:** Turn on to allow project masters to create repositories. By default, only project admins can create repositories.
- **Allow collaborator creation to:** Select one of the following values:
  - **Everyone:** Any user in the company can invite collaborators.
  - **Company admins only:** Only users designated as company admins can invite collaborators. For more information, see "User Profiles & authentication" on page 17.
  - **No one (disabled):** Collaborator creation is unavailable.

Supported browsers

Helix TeamHub has full support for latest versions of Chrome, Firefox, Safari and Opera. In addition, Helix TeamHub supports Microsoft Internet Explorer from version 10 onwards and Microsoft Edge. Internet Explorer 10 support is partial, meaning that things will be mostly functional but might be missing some visual effects or "non-core" functionalities.