HelixCore

Helix Core P4 Command Reference

2020.1

April 2020

PERFORCE

www.perforce.com

Contents

How to use this Guide

This reference includes every Helix server command, environment variable, and configurable, and assumes knowledge of the concepts in *Solutions Overview: Helix Version Control System*.

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

Syntax

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

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

Feedback

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

Other documentation

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

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

Earlier versions of this guide

- 2019.2
- 2019.1
- 2018.2
- 2018.1

What’s new in the Helix Core P4 Command Reference

This section provides a summary with links to topics in this reference. For a complete list of what's new in this release, see the *Release Notes*.

2020.1 release

- The storage upgrade process:
  - is now visible through the "p4 monitor" on page 371 command.
  - can make use of a new configurable that can suppress the generation of digests during a storage upgrade from a version prior to 2019.1. See "lbr.storage.skipkeyed" on page 761.
- To monitor the responsiveness of a server, see:
  - the new "p4 heartbeat" on page 251 command
  - the new configurables "net.heartbeat.interval" on page 770, "net.heartbeat.wait" on page 771, "net.heartbeat.missing.interval" on page 771, "net.heartbeat.missing.wait" on page 772, and "net.heartbeat.missing.count" on page 773
- TLS 1.3 is now supported, but TLS 1.2 remains the default. See "ssl.tls.version.max" on page 815
- The "p4 protects" on page 409 command:
  - has new access levels for stream spec protections. See `readstreamspec`, `openstreamspec`, and `writestreamspec` in the "p4 protect" on page 401 topic.
  - has the new `-H` option, which displays the protections that apply to the current client's host (IP address)
  - supports using the `-M` option with the `-h` flag to display the protections that apply to the specified host (IP address)
- The host field in the protections table now allows multiple IP addresses or CIDR matchers to be specified on a single line with a comma-separated list.
- Global labels can now be updated from edge servers using either `p4 tag -g` or `p4 labelsync -g`. See "p4 tag" on page 584 and "p4 labelsync" on page 319.
- The new `--no-graph` option for "p4 have" on page 247 and "p4 have (graph)" on page 249 limits the output to non-graph files when working with a hybrid workspace.
- The `p4 graph log` command has added functionality to the `--oneline` option:
  - an optional `tree` value adds a column in the output for tree-SHA-1 values
  - an optional `--no-abbrev` value causes SHA-1 values in the output to appear in the original 40 characters, instead of the default 7 characters abbreviation

  See "p4 graph log (graph)" on page 233.

- `p4 extension --list --type=extensions` has two new fields to indicate if the extension has a global configuration and at least one instance configuration. See "p4 extension" on page 186.

2019.2 release

- "p4 configure" on page 122 `history` allows the super user to display information about changes to configurables on any 2019.2 server.
- The `-p` option of "p4 obliterate" on page 379 marks the revison as purged and leave the integration history intact rather than removing the records. This one-step command improves performance compared to first invoking "p4 archive" on page 67, and then invoking `p4 archive -p`
- "p4 stream" on page 539 and "p4 jobspec" on page 289 support the automatic assignment of values to identify custom fields.
- "p4 storage" on page 532 has new options, `-l` and `-d`, to locate and delete any existing "orphaned" files left over from a previous failed submit or shelve operation. Two new configurables associated with this feature are "lbr.storage.allowsymlink" on page 759 and "lbr.storage.delay" on page 760.
- "p4 verify" on page 627 has a new option, `-Z`, to boost performance.
- Added the ability to create custom fields in stream specs. See the new command, "p4 streamspec" on page 554.
- To avoid the risk of conflicting field codes, field codes for custom stream and job specs can now be generated automatically if the admin uses the optional `NNN` placeholder value. For details, see "p4 streamspec" on page 554 and "p4 jobspec" on page 289.
- A new configurable, "db.monitor.term.allow" on page 736, allows users to terminate their own processes.

2019.1 release

- "Private editing of streams" on page 540 in the "p4 stream" on page 539 topic
- Additional enhancements to streams:
  - "Option to make switching between streams faster " on page 570 with "p4 switch" on page 570
  - The following commands now handle open stream specs: "p4 revert" on page 481, "p4 resolve" on page 458, "p4 submit" on page 558
  - "p4 unshelve" on page 607 now defaults to unshelving both files and the stream spec. Previously, the default was to unshelve only files.
  - "p4 streamlog" on page 551displays the history of changes to the specified list of streams.

- "Background archive transfer for edge server submits" on page 562: the user submitting a change with `"p4 submit" on page 558` **`-b`** will see the submit complete as soon as the metadata commit is completed, and will not have to wait for the archives to transfer. To enable this feature, set "submit.allowbgtransfer" on page 818 and, optionally, "submit.autobgtransfer" on page 820.
- "p4 pull" on page 420 has the new option for `"-t target" on page 425` for recovery of failed archive transfers.
- "p4 server" on page 493 - For all server types, the `"DistributedConfig:" on page 502` field of that server spec shows a line for each configurable that is set to a non-default value. In this field, you can edit the value, add a new line to set a different configurable to a non-default value, or delete a line to reset that configurable to its default value.
- The Helix Core server extensions are a fully-supported alternative to triggers. See
  - "p4 extension" on page 186 command
  - Helix Core Extensions Developer Guide
- Support for locking Git LFS (Large File Storage) files in depots of type graph by using the new commands: "p4 graph lfs-lock (graph)" on page 231, "p4 graph lfs-locks (graph)" on page 232, and "p4 graph lfs-unlock (graph)" on page 232 such that the locks created in Helix Core server with **`p4 graph lfs-lock`** are visible to Git clients, and the locks created in Git with **`git lfs lock`** are visible to Helix Core server.
- By default, the "server.maxcommands.allow" on page 812 configurable enables the **`super`** and **`operator`** users to issue certain administrative commands even if the "server.maxcommands" on page 811 is blocking standard users.
- Failover: see the Description section on the "p4 failover" on page 190 topic about the High Availability standby server.
- "p4 archive" on page 67 has the new **`-z`** option, which can reduce disk space usage.
- The "net.autotune" on page 769 configurable is enabled (**`1`**) by default. This enables the TCP stack to manage the size of the network send and receive buffers, allowing more efficient use of the network, especially over slow, high-latency connections. This behavior can be disabled in clients, proxies, brokers and the server by setting the configurable to **`0`**. On Windows-based platforms, send buffer sizes are not autotuned but are manually configurable with "net.tcpsize" on page 791.
- "p4 reconcile" on page 431 has a new option, **`-t`**, to consider the file type
- "p4 integrated" on page 271 has two new options to make it easier to show where a change has been integrated to: **`-s`** and **`--into-only`**
- The definition of the "P4LANGUAGE" on page 664 variable was updated in 2019.1 and it should be set to the language tag and optional region for the user. For example, if **`$LANG`** is **`en-US.UTF-8`**, set **`P4LANGUAGE=en-US`**. Servers with existing message translations will need their message database to be re-seeded with an updated message file.

- Support for utilizing multiple processor groups on Windows depends on the new configurable, "sys.threading.groups" on page 829.
- Setting the new "push.unlocklocked" on page 793 configurable to `1` automatically unlocks files that were locked as part of a failed push.

2018.2 release

- Failover from the current master to a standby server has improved.
  - See "p4 failover" on page 190, "p4 journalcopy" on page 294, and "p4 server" on page 493
- Commands with additional functionality:
  - "p4 undo" on page 596 supports graph depot
  - "p4 user" on page 616 `-D` deletes the user's workspaces along with the user
- New configurables related to single sign-on:
  - "auth.sso.args" on page 730
  - "auth.sso.allow.passwd " on page 729
  - "auth.sso.nonldap" on page 731

2018.1 Patch release

Multi-factor authentication (MFA)	For help on multi-factor authentication: ■ For the administrator, at the P4 command line, type **`p4 help mfa`** • This information is also available in the Administrator's Guide chapter on Triggers under "Triggering for multi-factor authentication (MFA)" • "dm.user.allowselfupdate" on page 744 is a configurable related to MFA configuration ■ For the end-user: **`p4 help login2`** • also available at "p4 login2" on page 351 This feature is currently supported for most Helix Core server clients, including: ■ Swarm 2018.1 ■ The 2018.2 releases of P4V, P4Eclipse, P4VS, and P4EXP
Graph depot commands	Commands to read or write against Git repos stored in the Helix server within a Graph Depot are no longer in Tech Preview. See "Graph depot commands" on page 29.

Configurables	To see whether changing the value of a given configurable requires stopping the server, in that configurable's details look for "After you change the value of this configurable, you must explicitly "stop" the server." For an example, see "ssl.tls.version.min" on page 815.

2018.1 release

Area	Feature
graph depot	▪ For a graph depot, "p4 describe" on page 157 can use the "classic" syntax, or using the SHA1, can provide a commit description. See the `p4 description` "Examples for files" on page 159. ▪ "p4 filelog" on page 199 ▪ "p4 fstat" on page 215 ▪ "p4 print" on page 395 ▪ "p4 revert" on page 481 ▪ "p4 show-permissions (graph)" on page 523 ▪ "p4 graph show-ref (graph)" on page 235 ▪ "p4 graph tags (graph)" on page 237 ▪ `p4 log` - at the command-line, type `p4 help-graph log` ▪ `p4 rebase` - at the command-line, type `p4 help-graph rebase` ▪ triggers related to graph depots: see the What's New in the *Helix Core Server Administrator Guide*. ▪ Support for Git Large File Storage (LFS). A replica can sync LFS files from graph depots. See the File Types chapter on "File type modifiers" on page 709, specifically the `+F` modifier.

"Configurables - alphabetical list" on page 717	■ "rpl.journalcopy.location" on page 799 be useful in a distributed environment. For details, see *Helix Core Server Administrator Guide*. ■ To prevent "p4 info" on page 261 from exposing sensitive fields, consider using "dm.info.hide" on page 740. ■ "p4 configure" on page 122 **`set`** now notifies you when you attempt to set a numeric configurable to a value outside of the acceptable range. Previously, a value below the minimum was silently altered to the minimum, and a value above the maximum was silently altered to the maximum. (Configurables already set outside the acceptable ranges will continue to be silently altered.) ■ To prevent users from being created if they would have no permissions on the server, you can configure "auth.ldap.userautocreate" on page 732 ■ Additional security is available by using "auth.tickets.nounlocked" on page 732 ■ "filesys.checklinks" on page 748 supports an additional value, `3`
"p4 admin" on page 58	For fail-over scenarios, the **`p4 admin end-journal`** command can be useful. See that command's "Examples" on page 61
"p4 keys" on page 309	**`p4 keys -e`** ***`nameFilter`*** supports a period in key names
"p4 stream" on page 539	Stream path definitions can include a wildcard in the final expression of a path, following the last slash. You can use this feature to refer to a collection of files, such as **`path_type pattern/to/....exe`** or **`path_type pattern/to/*.txt`** For examples, see the"To manage files of similar file-type in your stream specs, consider using wildcards (... and *) explicitly following the final slash in the path definition:" on page 548
Second factor authentication	For details and examples, see the **`p4 help 2fa`** documentation and the Support Knowledgebase article, "Second Factor Authentication Support". Please note that not all client applications have added support for second factor authentication yet.
LDAP	You can track the activity of "p4 ldapsync" on page 332. See **`ldapsync.csv`** at "p4 logparse" on page 356.

Getting started with commands

This book contains reference material for users and administrators of the Helix Core server, also referred to as Helix server.

Commands by functional area

Functional areas

Administration	Branching and Merging
Changelists	Client Workspace
Distributed Version Control	Environment
Files	Help
Jobs	Security

See also "Graph depot commands" on page 29.

Functional Area	Link to topic
Administration	"p4 admin" on page 58, "p4 archive" on page 67, "p4 bgtask" on page 72, "p4 cachepurge" on page 81, "p4 configure" on page 122, "p4 counter" on page 132, "p4 counters" on page 135, "p4 dbschema" on page 138, "p4 dbstat" on page 139, "p4 depot" on page 146, "p4 depots" on page 155, "p4 diskspace" on page 178, "p4 extension" on page 186, "p4 failover" on page 190, "p4 heartbeat" on page 251, "p4 journals" on page 302, "p4 key" on page 306, "p4 keys" on page 309, "p4 license" on page 336, "p4 lockstat" on page 343, "p4 logappend" on page 345, "p4 logger" on page 347, "p4 logparse" on page 356, "p4 logrotate" on page 359, "p4 logschema" on page 360, "p4 logstat" on page 362, "p4 logtail" on page 364, "p4 monitor" on page 371, "p4 obliterate" on page 379, "p4 ping" on page 391, "p4 property" on page 399, "p4 proxy" on page 412, "p4 pull" on page 420, "p4 reload" on page 436, "p4 renameuser" on page 444, "p4 replicate" on page 449, "p4 restore" on page 474, "p4 reviews" on page 488, "p4 server" on page 493, "p4 serverid" on page 505, "p4 servers" on page 507, "p4 storage" on page 532, "p4 triggers" on page 588, "p4 typemap" on page 592, "p4 unload" on page 599, "p4 verify" on page 627
Branching and Merging	"p4 branch" on page 75, "p4 branches" on page 79, "p4 copy" on page 129, "p4 cstat" on page 137, "p4 integrate" on page 265, "p4 integrated" on page 271, "p4 interchanges" on page 276, "p4 istat" on page 278, "p4 label" on page 310, "p4 labels" on page 316, "p4 labelsync" on page 319, "p4 list" on page 339, "p4 merge" on page 366, "p4 populate" on page 393, "p4 resolve" on page 458, "p4 resolved" on page 472, "p4 stream" on page 539, "p4 streamlog" on page 551, "p4 streams" on page 553, "p4 streamspec" on page 554, "p4 tag" on page 584
Changelists	"p4 change" on page 83, "p4 changes" on page 92, "p4 changelist" on page 90, "p4 changelists" on page 91, "p4 describe" on page 157, "p4 filelog" on page 199, "p4 opened" on page 383, "p4 reopen" on page 447, "p4 review" on page 486, "p4 shelve" on page 517, "p4 submit" on page 558, "p4 undo" on page 596, "p4 unshelve" on page 607
Client workspace	"p4 clean" on page 97, "p4 client" on page 100, "p4 clients" on page 118, "p4 flush" on page 212, "p4 have" on page 247, "p4 ignores" on page 258, " p4 sync" on page 574, "p4 update" on page 615, "p4 where" on page 631, "p4 workspace" on page 633, "p4 workspaces" on page 634
Distributed version control (DVCS)	"p4 init" on page 263, "p4 fetch" on page 194, "p4 push" on page 427, "p4 remote" on page 438, "p4 remotes" on page 441, "p4 unsubmit" on page 610

Functional Area	Link to topic
Environment	"p4 set" on page 513, "Environment and registry variables" on page 637, "P4AUDIT" on page 640, "P4AUTH" on page 641, "P4BROKEROPTIONS" on page 642, "P4CHANGE" on page 643, "P4CHARSET" on page 644, "P4COMMANDCHARSET" on page 649, "P4CLIENT" on page 647, "P4CONFIG" on page 650, "P4DEBUG" on page 652, "P4DIFF" on page 654, "P4DIFFUNICODE" on page 655, "P4EDITOR" on page 656, "P4HOST" on page 659, "P4IGNORE" on page 660, "P4JOURNAL" on page 663, "P4LANGUAGE" on page 664, "P4LOG" on page 665, "P4MERGE" on page 667, "P4MERGEUNICODE" on page 669, "P4NAME" on page 670, "P4PAGER" on page 672, "P4PASSWD" on page 673, "P4PCACHE" on page 674, "P4PFSIZE" on page 675, "P4POPTIONS" on page 676, "P4PORT" on page 677, "P4ROOT" on page 680, "P4TARGET" on page 683, "P4TICKETS" on page 685, "P4USER" on page 687, "PWD" on page 688, "TMP, TEMP" on page 689
Files	"p4 add" on page 53, "p4 attribute" on page 71, "p4 copy" on page 129, "p4 delete" on page 143, "p4 diff" on page 162, "p4 diff2" on page 168, "p4 dirs" on page 175, "p4 edit" on page 180, "p4 files" on page 203, "p4 fstat" on page 215, "p4 grep" on page 238, "p4 move" on page 377, "p4 lock" on page 341, "p4 print" on page 395, "p4 reconcile" on page 431, "p4 rename" on page 443, "p4 revert" on page 481, "p4 status" on page 530, "p4 sizes" on page 527, "p4 unlock" on page 602
Help	"p4 help" on page 254, "p4 help-graph (graph)" on page 256, "File specifications" on page 695, "Global options" on page 690, "File types" on page 707
Jobs	"p4 fix" on page 207, "p4 fixes" on page 210, "p4 job" on page 280, "p4 jobs" on page 283, "p4 jobspec" on page 289
Security	"p4 group" on page 240, "p4 groups" on page 245, "p4 login" on page 349, "p4 login2" on page 351, "p4 logout" on page 354, "p4 passwd" on page 387, "p4 protect" on page 401, "p4 protects" on page 409, "p4 tickets" on page 587, "p4 trust" on page 590, "p4 user" on page 616, "p4 users" on page 625, "P4CLIENTPATH" on page 648, "P4SSLDIR" on page 681, "P4TRUST" on page 686
Streams	"p4 stream" on page 539, "p4 streamlog" on page 551, "p4 streams" on page 553, "p4 streamspec" on page 554

Graph depot commands

In addition to command-line help, you can use this book's topics on the:

- graph depot version of classic commands - "Commands that differ for graph depots" on the next page
- commands that are for graph depots only - "Graph depot commands" on page 32

p4 help-graph

On the command-line, to display help for the graph data model that supports git, type `p4 help-graph` or see "p4 help-graph (graph)" on page 256.

Commands that differ for graph depots

Some of the existing commands behave differently for graph depots.

The Helix Core server natively supports two data models for read and write operations:

- **Classic data model** within Helix Core
- **Graph data model** for Git repos stored in Helix server

The content of the online Help and command-line Help reflects the differences of these two data models:

Classic	Graph	Command-line Help for Graph
"p4 add" on page 53	"p4 add (graph)" on page 56	`p4 help-graph add`
"p4 client" on page 100	"p4 client (graph)" on page 114	`p4 help-graph client`
"p4 delete" on page 143	"p4 delete (graph)" on page 145	`p4 help-graph delete`
"p4 describe" on page 157	"p4 describe (graph)" on page 160	`p4 help-graph describe`
"p4 diff" on page 162	"p4 diff (graph)" on page 166	`p4 help-graph diff`
"p4 diff2" on page 168	"p4 diff2 (graph)" on page 172	`p4 help-graph diff2`
"p4 dirs" on page 175	"p4 dirs (graph)" on page 176	`p4 help-graph dirs`

Classic	Graph	Command-line Help for Graph
"p4 edit" on page 180	"p4 edit (graph)" on page 183	`p4 help-graph edit`
"p4 filelog" on page 199	"p4 filelog (graph)" on page 201	`p4 help-graph filelog`
"p4 files" on page 203	"p4 files (graph)" on page 205	`p4 help-graph files`
"p4 fstat" on page 215	"p4 fstat (graph)" on page 224	`p4 help-graph fstat`
"p4 have" on page 247	"p4 have (graph)" on page 249	`p4 help-graph have`
"p4 lock" on page 341	"p4 lock (graph)" on page 342	`p4 help-graph lock`
"p4 merge" on page 366	"p4 merge (graph)" on page 368	`p4 help-graph merge`
"p4 opened" on page 383	"p4 opened (graph)" on page 385	`p4 help-graph opened`
"p4 print" on page 395	"p4 print (graph)" on page 397	`p4 help-graph print`
"p4 reconcile" on page 431	"p4 reconcile (graph)" on page 434	`p4 help-graph reconcile`
"p4 resolve" on page 458	"p4 resolve (graph)" on page 468	`p4 help-graph resolve`

Classic	Graph	Command-line Help for Graph
"p4 revert" on page 481	"p4 revert (graph)" on page 484	`p4 help-graph revert`
"p4 submit" on page 558	"p4 submit (graph)" on page 569	`p4 help-graph submit`
"p4 switch" on page 570	"p4 switch (graph)" on page 572	`p4 help-graph switch`
" p4 sync" on page 574	" p4 sync (graph)" on page 581	`p4 help-graph sync`
"p4 unlock" on page 602	"p4 unlock (graph)" on page 604	`p4 help-graph unlock`

Tip
To learn how to add the content of a graph depot repo to a Helix Core "stream", see "p4 stream" on page 539 > "Form Fields" on page 542 > Paths, where a Note gives examples.

Graph depot commands

The following graph depots commands do not apply to "classic" Helix Core server depots:

Command	Command-line Help
"p4 help-graph (graph)" on page 256	`p4 help-graph`
"p4 graph lfs-lock (graph)" on page 231	`p4 help-graph lfs-lock`
"p4 graph lfs-locks (graph)" on page 232	`p4 help-graph lfs-locks`
"p4 graph lfs-unlock (graph)" on page 232	`p4 help-graph lfs-unlock`
"p4 graph log (graph)" on page 233	`p4 help-graph log`
"p4 graph rebase (graph)" on page 234	`p4 help-graph rebase`
"p4 graph tag (graph)" on page 236	`p4 help-graph tag`
"p4 graph tags (graph)" on page 237	`p4 help-graph tags`
"p4 pubkey (graph)" on page 415	`p4 help-graph pubkey`

Command	Command-line Help
"p4 pubkeys (graph)" on page 418	`p4 help-graph pubkeys`
"p4 repo (graph)" on page 451	`p4 help-graph repo`
"p4 repos (graph)" on page 454	`p4 help-graph repos`
"p4 grant-permission (graph)" on page 227	`p4 help-graph grant-permission`
"p4 revoke-permission (graph)" on page 490	`p4 help-graph revoke-permission`
"p4 check-permission (graph)" on page 95	`p4 help-graph check-permission`
"p4 show-permission (graph)" on page 522	`p4 help-graph show-permission`
"p4 show-permissions (graph)" on page 523	`p4 help-graph show-permissions`
"p4 show-ref (graph)" on page 525	`p4 help-graph show-ref`

Getting help with p4 help

In addition to the material in this manual, you can get help for Helix Core server commands by using the **p4 help** command, which provides information about individual commands or for areas like jobs, revisions, or file types.

The output to the **p4 help** command as well as the syntax diagrams included in this manual show the short form of the command options. You can also specify command options using long-form syntax. For example, instead of the following command format:

```
$ p4 reopen -c 1602 -t text+F //depot/my/file
```

You can now use this format:

```
$ p4 reopen --change 1602 --filetype text+F //depot/my/file
```

Note that long-form option names are preceded by two hyphens rather than the usual single hyphen.

Options that are rarely used have only a short form.

To display long-form option syntax for a particular command, use the `--explain` option. For example:

```
$ p4 reopen --explain
```

This will generate output like the following:

```
--omit-moved (-1): disables following renames resulting from 'p4 move'
--filetype (-t): specifies the filetype to be used.
--change (-c): specifies the changelist to use for the command.
Usage: reopen [-c changelist#] [-t type] files...
```

To display information about a single option for a command, specify the option name with `--explain`. For example:

```
$ p4 revert --explain -k
```

Command aliases

A small set of commands have predefined aliases. For example, you can use **`p4 integ`** for **`p4 integrate`**, or you can use **`p4 changes`** for **`p4 changelists`**. You can also define your own aliases for commands, and these can range from simple word substitutions to what might be called light scripting.

There are many reasons for creating command aliases: you want to use commands in a language other than English, you want to use commands that are familiar to you from other version control systems, you want to use different defaults, you want to streamline system administration, or you want to use different output formats.

This section covers the following topics:

- The process of creating an alias
- Basic syntax of alias definitions
- Simple and complex alias definitions
- How you put it all together
- Advanced topics
- Limitations

Command aliases can only be used by command line clients. Aliases do not work with the derived clients, APIs, or GUIs. Because aliasing is a client-side feature, you can use a command alias with any server, proxy, broker, or replica configuration. However, the particular commands you can run still depend on the server to which you are issuing the commands.

Also in this section:

Defining aliases

Command aliases are defined in an *alias file*. To define one or more aliases, you do the following:

1. Create a file named `.p4aliases` in your home directory (`p4aliases.txt` in Windows).

   (The file is stored in your `$HOME` directory on Unix and Mac systems, and in your `$USERPROFILE` directory on Windows.)

   If you do not put the alias file in the home directory, you must define the `P4ALIASES` environment variable to specify the location of the alias file.

2. Add one or more alias definitions to the alias file.

   The following topics in this section explain the syntax of alias definitions and provides examples of alias definitions.

   Tip
   The alias file can:

   - contain blank lines, but they will be ignored
   - contain comments lines. A comment line begins with `#` as the first non-blank character.
   - be edited as often as you like to add, modify, or delete definitions

3. Preview the effect of the aliases you have defined by running a command like the following for a given command alias:

   ```
   $ p4 --aliases=dry-run myalias
   ```

   The output to the alias command will show you the command or commands that would be run without actually running the command. For more information, see "Previewing alias substitutions" on page 41.

4. Run the command alias to execute the command or commands associated with the alias.

   The server processes command aliases in the order they have been defined, going through each one until it finds one that modifies the current command. It then restarts from the beginning, rechecking each alias. This means that a given command might be transformed more than once before it is run, depending on the aliases that use the command.

Use the `p4 aliases` command to get a listing of all currently defined aliases.

Command alias syntax

The definition of a command alias can be complex. This section describes the basic syntax for defining a command alias and introduces the elements that you can use in a definition. The sections that follow provide examples for defining complex aliases.

In its simplest form, the syntax for a command alias definition looks like this:

```
alias = transformation
```

For example, you want to use French for a command name:

```
fiches = files
```

Having included this definition in your alias file, you can now execute a command like **`p4 fiches @2015/3/15`**, and have the server list information about all file revisions in the depot as of March 15, 2015.

The alias can use arguments. In this case, the alias arguments are matched against the values the user provides in the transformation. The arguments in the transformation do not have to occur in the same order as they are shown in the alias. They are matched by name. Syntax for this definition looks like this:

```
alias-name [[$(arg1)...[$(argn)]]= transformation
```

Syntax for the transformation can vary widely. Here is one possibility:

```
command $(arg1) $(arg2) $(arg3)
```

For example:

```
recent-changes $(max) = changes -m $(max)
```

The recent-changes alias might then be called as follows:

```
$ p4 recent-changes 5
```

And the command would show the last five submitted changelists.

Alias definitions can contain the following elements:

- command arguments
- environment variables

  These include all Helix server environment variables (for example, **`P4USER`**, **`P4CLIENT`**, **`P4PORT`**) as well as OS variables.
- input/output redirection
- special operators

The following table describes the special operators for use in command alias definitions:

Operator	Meaning
`$(arg)`	Specifies an alias argument in the *alias* and is matched with arguments in the *transformation*.
`#`	As the first non-blank character of a line, indicates a comment.
`&&`	Chain commands. See "Limitations" on page 42 for information about chaining commands.
`\`	Continue line (use to break up long lines when there are no new commands).
`<`	Take input from.
`>`	Send output to.

Operator	Meaning
`$(EQ)`	Equal to.
`$(LT)`	Less than.
`$(GT)`	Greater than.
`p4subst`	String substitution, for example in editing specs. See "Editing specifications" on page 41.

Basic examples

This section provides examples of simple command alias definitions and illustrates the many uses for even the simplest definitions.

- **Help me remember who I am.**

  ```
  me = set P4USER
  ```

- **Help me feel more comfortable as a user of another source control system.**

  ```
  checkout = sync
  commit = submit
  purge = clean
  stash = shelve
  stash-list = changes -s shelved
  ```

- **Create a personalized status command that also shows files that need syncing.**

  ```
  my-status = status && sync -n
  ```

- **Set different defaults.**

  ```
  annotate = annotate -u
  grep = grep -i
  changes = changes -u $(P4USER)
  ```

- **Simplify system administration.**

  Shutting down the server, displaying active users:

  ```
  halt = admin shutdown
  active-users = changes -m 3 &&
                 monitor show &&
                 lockstat
  ```

- **Remembering to clean up empty changelists**

  ```
  kill-shelf $(cl) = shelve -d -c $(cl) &&
                     change -d $(cl)
  ```

- **Change the order of arguments.**

  ```
  clone $(p4port) $(path) $(dir) = -d $(dir) -u bruges clone -p
  $(p4port) -f $(path)
  ```

  Now the following command does what you want:

  ```
  clone perforce:1666 //depot/main/p4...  ~/local-repos/main
  ```

Complex examples

More complex alias commands can be formed using redirection and special operators.

- **Cherry picking.**

  This alias definition:

  ```
  cherry-pick-change $(cl) $(s) $(t) = integrate
  //depot/$(s)/...@$(cl),$(cl) //depot/$(t)/...
  ```

  Turns this command:

  ```
  $ p4 cherry-pick-change 1015978 p15.2 main
  ```

  Into this:

  ```
  $ p4 integrate //depot/p15.2/...@1015978,1015978
  //depot/main/...
  ```

  Here is another cherry picking example that creates a little merge script:

  ```
  cherry-pick $(cl) $(s) $(t) $(msg) = \
                 integrate //depot/$(s)/...@$(cl),$(cl)
  //depot/$(t)/... &&
                 resolve -am -Ac //depot/$(t)/... &&
                 submit -d $(msg) &&
                 sync
  ```

  You could then execute a command like the following:

  ```
  $ p4 cherry-pick 1015978 two one "line a merged into one"
  ```

  This would run the following commands:

```
p4 integrate //depot/two/...@1015978,1015978 //depot/one/...
p4 resolve -am -Ac //depot/one/...
p4 submit -d "Cherry-pick change https://swarm.perforce.com/@1015978
[1015978]
              from //depot/two/... to //depot/one/..."
p4 sync
```

- **Simple pipelining.**

  Starting with a simple example:

  ```
  newStreamsDepot $(dpt) = depot -o -t stream $(dpt) > $(depotSpec) &&
                           depot -i < $(depotSpec)
  ```

  Note that when using redirection, the **$** variables used in the transformation side of the definition, do not need to correspond to the arguments specified one the left side of the equation. In the example above, *depotSpec* is a variable created during the execution of the **`newStreamDepot`** alias.

  Here are a couple of aliases for merge down copy up:

  ```
  mergedown $(b) = fetch &&
                   switch $(b) &&
                   merge &&
                   resolve -am &&
                   submit -d "Merged down from main"

  copyup $(b)    = switch dev &&
                   merge --from $(b) &&
                   resolve -as &&
                   submit -d "Copied up from $(b)" &&
                   push

  # Note the use of the branch name in the submit message of the copyup
  alias.
  ```

- **DVCS: Aliases to communicate with multiple servers.**

  Use aliases like the following when copying spec objects from the shared server to your personal server.

  ```
  copy-user $(p4port) = -p $(p4port) user -o $(u) > $(spec) &&
                      user -i < $(spec)
  ```

```
copy-job $(p4port) $(j) = -p $(p4port) job -o $(j) > $(spec)  &&
                    job -i < $(spec)

copy-stream $(p4port) $(s)  = -p $(p4port) stream -o $(s) > $(spec)
&&
                    stream -i < $(spec)
```

Putting it all together

Combining the various elements allows you to build aliases.

- **Make a new task stream**

  If your streams are stored in the depot named **`//stream`**, here's how you would make a new task stream:

  ```
  newTaskStream $(task) $(parent) = stream -o -t task -P
  //stream/$(parent) \
                              //stream/$task) > $(streamSpec) &&
                              stream -i < $(streamSpec) &&
                              populate -r -S //stream/$(task) &&
                              client -s -S //stream/$(task) &&
                              sync
  ```

  Then switching to a new stream becomes simple:

  ```
  $ p4 newTaskStream job084103 bp-dev
  ```

- **Delete a stream**

  ```
  nuke-stream $(branch) = stream -d //stream/$(branch) &&
                          obliterate -y //stream/$(branch)...
  ```

- **Make an alias of an alias**

  ```
  checkout = sync
  commit = submit

  co  = checkout
  cmt = commit
  ```

Previewing alias substitutions

Use the client-side command option **`--aliases=dry-run`** to display the command or commands that would have run without actually running them.

For example, if your alias file contains the following:

```
nuke-stream $(branch) = stream -d //stream/$(branch) &&
                        obliterate -y //stream/$(branch)...
```

And you execute the following command:

```
$ p4 --aliases=dry-run nuke-stream test1
```

The command would return:

```
p4 stream -d //stream/test1
p4 obliterate -y //stream/test1
```

Advanced topics

This section describes more advanced uses of command aliases.

Editing specifications

The **`p4subst`** special operator allows you to edit specs. It is roughly analogous to doing the following in a shell pipeline:

```
| sed 's/regular_expression/literal/g'
```

The **`p4subst`** special operator should normally be used in an alias as fellows:

```
something > $(output) &&
p4subst "regular expression" "literal replacement" < $(output) > $(result)
&&
something else < $(result)
```

As an example, consider the string substitution in the following alias:

```
newChange $(desc) = change -o > $(chg) &&
                   p4subst "$(LT)enter.*$(GT)" $(desc) < $(chg) > $(chg2)
&&
                   change -i < $(chg2)
```

The alias replaces the default change description with the argument provided to **`p4 newChange`** ***`description`***.

Creating alias files for each workspace

To create an alias file for each workspace, add the following line to your **`P4CONFIG`** file:

```
P4ALIASES=$configdir/p4aliases.txt
```

Since your **`P4CONFIG`** file is found wherever you might be working and that location is known by the special **`$configdir`** value, you can have a **`P4ALIASES`** file that is specific to this workspace and which is conveniently found no matter where you are in that workspace.

Limitations

Aliases can be very powerful. Be mindful of consequences:

- Multi-command chains in an alias are different than shell pipelines. Each sub-command in a shell pipeline is started by the shell as a separate child process, and their input and output is connected using operating system constructs. A multi-command alias, on the other hand, executes in the context of a top-level **`p4`** process, and it executes each sub-command serially, inside the parent **`p4`** process, storing the output in memory. This limits the amount of data that can be piped from one command to the next.
- If one chained command fails, no subsequent commands are executed.

Naming conventions

- Clients, depots, labels, and branches may not have the same name.
- The following names are reserved and cannot be used to name anything: **`head`**, **`have`**, **`none`**.

The following table provides some suggestions:

Object	Naming convention
branches	Best to name them.
clients	The following scheme is commonly used, but not enforced in any way. Use it if it suits your purpose. *user.machineTag.product* *user.machineTag.product.branch* *user* is the OS user. *machineTag* is the host name or something that describes the host, for example **`Win7VM`** or **`P4MBPro`** (for MacBook Pro). Whether you use *product* or *product.branch* depends on whether your workspace gets re-purposed from stream to stream, in which case use *product*), or you have multiple workspaces, one for each branch, in which case use *product.branch* to tie the workspace name to the branch.

Object	Naming convention
depots	Best to keep the names short. Depot names are part of an organization hierarchy for all your digital assets, so naming them and planning directory structure is especially important.
jobs	Name jobs to match your external defect tracker issues: for example, `PRJ-1234` for JIRA issues.
labels	Label names are site-dependent and might vary with code management schemes and versioning needs. For example `R-3.2.0` might refer to release 3.2.0.

Creating scripts

In addition to chaining commands in an alias to create a light script, you can combine the commands described in this manual in scripts. The Helix Core server supports *Triggers*, which are user-written scripts called by a Helix server whenever certain operations occur. Examples of such operations are changelist submissions, changes to forms, and login attempts.

See Triggers and Extensions in the *Helix Core Server Administrator Guide*.

Commands and metadata

Some commands write metadata to the database. Other commands read metadata from the database.

- commands that write metadata can create things
- commands that read metadata can report about what has been created

Examples:

Write Metadata	Read Metadata
p4 branch	p4 branches
p4 changelist	p4 changelists
p4 client	p4 clients
p4 depot	p4 depots
p4 repo	p4 repos
p4 resolve	p4 resolved
p4 user	p4 users

Write Metadata	Read Metadata
p4 sync p4 merge p4 revert	
	p4 aliases p4 help

Commands

Click a letter

p4 add

Open files in a client workspace for addition to the depot.

"Syntax" on page 19

```
p4 [g-opts] add [-c changelist] [-d -f -I -n] [-t filetype] file
...
```

Description

`p4 add` opens files within the client workspace for addition to the depot. The specified files are linked to a changelist. The files are not actually added to the depot until the changelist is committed with `p4 submit`. The added files must either not already exist in the depot, or exist in the depot but be marked as deleted at the head revision.

The commands `p4 add *` or `p4 add ...` are synonymous to `p4 reconcile -a *`.

This means that any files in the workspace that do not exist in the depot are opened for add. Using the `-a` option does not affect the behavior of `p4 add -d`.

To open a file with `p4 add`, the file must exist in your client view, but does not need to exist in your workspace at the time of `p4 add`. The file must exist in your workspace when you run `p4 submit`. Otherwise the submission fails. `p4 add` does not create or overwrite files in your workspace. If a file does not exist, you must create it yourself.

By default, the specified files are opened in the default changelist. To open the files in a specified changelist, use the `-c` option.

To move files from the default changelist to a numbered changelist, use the `p4 change` command.

By default, `p4 add` skips over files mentioned in any applicable `P4IGNORE` files. To override this behavior, use the `-I` option to ignore the contents of any `P4IGNORE` files.

When adding files, the command first examines the typemap table (`p4 typemap`) to see if the system administrator has defined a file type for the files being added. If a match is found, the file's type is set as defined in the typemap table. If a match is *not* found, the command examines the first bytes of the file based on the "filesys.binaryscan" on page 747 configurable (by default, 65536 bytes) to determine whether it is `text` or `binary`, and the files are stored in the depot accordingly. By default, text file revisions are stored in reverse delta format. Newly-added text files larger than the limit imposed by the "filetype.maxtextsize" on page 755 configurable (by default, 10 MB) are assigned filetype `text+C` and stored in full. Files compressed in the `.zip` format (including `.jar` files) are also automatically detected and assigned the type `ubinary`. Other binary revisions are stored in full, with compression.

The `-t` *`filetype`* option explicitly specifies a file type, overriding both the typemap table and the default file type detection mechanism.

"Syntax" on page 19

To add files containing the characters `@`, `#`, `*`, and `%`, use the `-f` option. This option forces literal interpretation of characters otherwise used by Helix server as wildcards.

If you open a file for edit or move/add, and another user subsequently deletes the file you opened, the operation will fail with an error when you submit the changelist. To ensure that you create the desired target file, specify the `-d` option ("downgrade"). More specifically:

- You open a file for edit, then another user submits a changelist that deletes or moves the file. When you submit your edits, Helix server returns an error and the file remains open for edit. To restore the file (including any changes you have made) to the depot location from which you checked it out, open the file for add and specify the `-d` option, then submit the file.
- You open a file for move/add and another user submits a changelist that deletes the source file. When you submit the move, Helix server returns an error and the file remains open for add/move. To create the desired target file, issue the **p4 add -d** command, specifying the target file, and submit the file.

Options

`-c` ***changelist***	Opens the files for `add` within the specified *changelist*. If this option is not used, the files are linked to the default changelist.
`-d`	Downgrade file open status to simple add.
`-f`	Use the `-f` option to force inclusion of wildcards in filenames. See "File specifications" on page 695 for details.
`-I`	Do not perform any ignore checking; ignore any settings specified by `P4IGNORE`.
`-n`	Preview which files would be opened for add, without actually changing any files or metadata.
`-t` ***filetype***	Adds the file as the specified *filetype*, overriding any settings in the typemap table. See "File types" on page 707 for a list of Helix server file types.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

- "Wildcards" on page 695 in file specifications provided to `p4 add` are expanded by the local operating system. For instance, the `...` wildcard cannot be used with `p4 add`.
- There is no difference between adding files to an empty depot and adding files to a depot that already contains other files. You can populate new, empty depots by adding files from a client workspace with `p4 add`.
- Do not use ASCII expansions of special characters with `p4 add -f`. To add the file `status@june.txt`, use:

  ```
  p4 add -f status@june.txt
  ```

  If you manually expand the `@` sign and attempt to add the file `status%40june.txt`, Helix server interprets the `%` sign literally, expands it to the hex code `%25`, resulting in the filename `status%2540june.txt`.

Examples

`p4 add -t binary file.pdf`	Assigns a specific file type to a new file, overriding any settings in the typemap table.
`p4 add -c 13 *`	Opens all the files within the user's current directory for `add`, and links these files to changelist `13`.
`p4 add README ~/src/*.c`	■ Opens the `README` file in the user's current working directory for `add`. ■ Opens all `*.c` files in the user's `~/src` directory for `add`. ■ These files are linked to the default changelist
`p4 add -f *.c`	Opens a file named `*.c` for `add`. To refer to this file in views, or with other commands, you must subsequently use the hex expansion `%2A` in place of the asterisk. For more information, see "Limitations on characters in filenames and entities" on page 699.

Related Commands

To open a file for edit	`p4 edit`
To open a file for deletion	`p4 delete`
To move (rename) a file	`p4 move`
To copy all open files to the depot	`p4 submit`

To read files from the depot into the client workspace	`p4 sync`
To create or edit a new changelist	`p4 change`
To change default behavior of text and binary file detection	`p4 configure`
To list all opened files	`p4 opened`
To revert a file to its unopened state	`p4 revert`
To move an open file to a different pending changelist	`p4 reopen`
To change an open file's file type	`p4 reopen -t` ***filetype***

p4 add (graph)

Open a new file to add it to the repo.

"Syntax" on page 19

`p4 [`*`g-opts`*`] add [-c `*`changelist`*` -n -t `*`filetype`*`] `*`file`*` ...`

Description

Open a file for adding to the depot.

- To associate the open files with a specific pending changelist, use the **`-c`** flag. If you omit the **`-c`** flag, the open files are associated with the default changelist
- To specify file type, use the **`-t`** flag
- To display a preview of the specified add operation without changing any files or metadata, use the **`-n`** flag

Options

`-c` ***`changelist`***	Opens the files for **`add`** within the specified *changelist*. If this option is not used, the files are linked to the default changelist.
`-n`	Preview which files would be opened for add, without actually changing any files or metadata.

`-t` ***`filetype`***	By default, **p4 add** assumes the file is a simple text file. The following alternate filetypes are available: ▪ **`text+x`** - executable file, such as a shell script ▪ **`symlink`** - symbolic link ▪ **`binary`** - binary file ▪ **`binary+F`** - large binary file, to be stored using git-lfs
`g-opts`	See "Global options" on page 690.

p4 admin

Perform administrative operations on the server.

"Syntax" on page 19

```
p4 [g-opts] admin checkpoint [-z | -Z] [prefix]
p4 [g-opts] admin journal [-z] [prefix]
p4 [g-opts] admin stop
p4 [g-opts] admin restart
p4 [g-opts] admin updatespecdepot [-a | -s type]
p4 [g-opts] admin resetpassword -a | -u user
p4 [g-opts] admin setldapusers
p4 [g-opts] admin end-journal
```

Description

The `p4 admin` command allows Helix server superusers to perform administrative tasks, even when working from a different machine than the one running the shared Perforce service.

To stop the service, use `p4 admin stop`. This locks the database to ensure that it is in a consistent state upon restart, and then shuts down the background process.

To restart the service, use `p4 admin restart`. The database is locked, the service restarts, and some of the `p4 configure` settings that require a restart are applied.

Important
When you look up the details of certain configurables under "Configurables - alphabetical list" on page 717, it might say:

After you change the value of this configurable, you must explicitly "stop" the server.

Note
`p4 admin restart` is not sufficient.

For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

For Windows, see Starting and stopping the Helix server.

"Syntax" on page 19

To take a checkpoint, use **`p4 admin checkpoint [`*`prefix`*`]`**. This is equivalent to logging in to the server machine and taking a checkpoint with **`p4d -jc [`*`prefix`*`]`**. A checkpoint is taken and the journal is copied to a numbered file. If a *prefix* is specified, the files are named ***`prefix`*`.ckp.`*`n`*** or ***`prefix`*`.jnl.`*`n-1`*** respectively, where ***`n`*** is a sequence number. The MD5 checksum of the checkpoint is written to a separate file, **`checkpoint.`*`n`*`.md5`**, and the **`lastCheckpointAction`** counter is updated to reflect successful completion.

Note
You must be connected to the server to issue the **`p4 admin checkpoint`** command.

You can store checkpoints and journals in the directory of your choice by specifying the directory as part of the prefix. (Rotated journals are stored in the **`P4ROOT`** directory, regardless of the directory in which the current journal is stored.) If no *prefix* is specified, the default filenames **`checkpoint.`*`n`*** and **`journal.`*`n-1`*** are used.

The **`p4 admin journal`** command is equivalent to **`p4d -jj`**. For details, see Triggering on journal rotation in the *Helix Core Server Administrator Guide*. The files are created in the server root specified when the Perforce service was started.

The **`p4 admin updatespecdepot`** command causes the service to archive stored forms into the spec depot.

Note

- If the **`-a`** option is used, all of the form specification types are archived.
- If the **`-s`** option option is used
  - only those of the specified *type* are archived
  - the other types are created in the spec depot

The **`p4 admin resetpassword`** command forces specified users with existing passwords to change their passwords before they can run another command. This command works only for users whose **`authMethod`** is set to **`perforce`**. However, you can use it in a mixed environment, that is an environment in which authentication is based both on Helix server and LDAP.

- To force password reset of all users with passwords (including the superuser who issued the command), use **`p4 admin resetpassword -a`**.
- To force a single users to reset their password, use **`p4 admin resetpassword -u`** ***`user`***.

The **`p4 admin setldapusers`** command allows you to convert all existing non-super users to use LDAP authentication. The command changes the **`AuthMethod`** field in the user specification for each user from **`perforce`** to **`ldap`**. If **`super`** users want to use LDAP authentication, they must set their **`AuthMethod`** manually.

Options

`-a`	For `p4 admin updatespecdepot`, update the spec depot with all current forms.
`-s` *type*	For `p4 admin updatespecdepot`, update the spec depot with forms of the specified type, where type is one of `client`, `depot`, `repo`, `branch`, `label`, `typemap`, `group`, `user`, `job`, `stream`, `triggers`, `protect`, `server`, `license`, `jobspec`.
`-z`	For `p4 admin checkpoint` and `p4 admin journal`, save the checkpoint and saved journal file in compressed (gzip) format, appending the `.gz` suffix to the files.
`-Z`	For `p4 admin checkpoint`, save the checkpoint in compressed (gzip) format, appending the `.gz` suffix to the file, but leave the journal uncompressed for use by replica servers.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

- The `p4 admin updatespecdepot` and `p4 admin resetpassword` commands require `super` access. The others require that the user be an operator (see `p4 user`) or have `super` access.
- To require all newly-created users with passwords to reset their passwords before invoking their first command, set the `dm.user.resetpassword` configurable:

  ```
  p4 configure set dm.user.resetpassword=1
  ```

  Running `p4 admin resetpassword -a` resets only the passwords of users who presently exist (and who have passwords).
- Because `p4 admin stop` shuts down the Perforce service, you might see an error message indicating that the connection was closed unexpectedly. You can ignore this message.
- The spec depot must exist before running `p4 admin updatespecdepot`.
- `p4 dbstat`, `p4 lockstat`, and `p4 logstat` are standalone commands; the old `p4 admin` syntax remains as an alias for backward compatibility.
- See the *Helix Core Server Administrator Guide* and *Helix Core Server Administrator Guide*.

"Syntax" on page 19

Examples

`p4 admin stop`	Stop the shared service
`p4 admin checkpoint`	Create a checkpoint named **`checkpoint.`*`n`***, and start a new journal named **`journal`**, copying the old journal file to **`journal.`*`n-1`***, where ***`n`*** is a sequence number.
`p4 admin checkpoint name`	Create a checkpoint named **`name.ckp.`*`n`***, and start a new journal named **`journal`**, copying the old journal file to **`name.jnl.`*`n-1`***, where ***`n`*** is a sequence number.
`p4 admin end-journal`	In a failover scenario, this command: ■ ends journal replication at the most recent successfully replicated consistency point ■ returns the journal number and the offset of that consistency point ■ stops to the standby server's **`journalcopy`** thread

Related Commands

To see the status of the last checkpoint	`p4 counter` **lastCheckpointAction**

p4 aliases

Display command aliases that are currently defined in a `.p4aliases` file.

For complete information, see "Command aliases" on page 34.

"Syntax" on page 19

```
p4 [g-opts] aliases
```

Description

The `.p4alias` file contains the definitions of the command aliases you have created.

The command output for **`p4 aliases`** does not include pre-defined aliases, for example **`p4 changes`** for **`p4 changelists`**. It only displays the contents of your `.p4aliases` file, and it does not include comments. For example:

```
$ p4 aliases
co => edit
ci => submit
st => status
shelved => changes -s shelved -u $(P4USER) -c $(P4CLIENT)
pending =>  changes -s pending -u $(P4USER) -c $(P4CLIENT)
desc => describe -s
purge => clean -I
blame => annotate -u
nuke-shelf $(change) => shelve -dc $(change) &&
                        revert -c $(change) //... &&
                        change -d $(change)
newChange $(desc) => change -o > $(chg) &&
                     p4subst  "$(LT)enter.*$(GT)" $(desc) < $(chg) >
$(chg2) &&
                     change  -i < $(chg2)
```

Options

`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`none`

p4 annotate

Print file lines along with their revisions.

By default, ignores changes to text files over 10 MB in length.

Superusers of Helix server can override this limit by setting the `"dm.annotate.maxsize" on page 739` configurable.

"Syntax" on page 19

```
p4 [g-opts] annotate [-a -c -i -I -q -t -T -u] [-doptions]
FileSpec[revSpec]
```

Description

The **`p4 annotate`** command displays the revision number for each line of a revision (or range of revisions) of a file (or files). Using the **`-u`** option displays the name of the user who modified the change and the date when the modification occurred. To know why the modification happened, use the **`p4 filelog`** command on the indicated revision(s).

To display the changelist number associated with each line of the file, use the **`-c`** option.

If you specify a revision number, only revisions up to that revision number are displayed. If you specify a revision range, only revisions within that range are displayed.

By default, the first line of output for each file is a header line of the form:

```
filename#rev - action change num (type)
```

where:

- ***filename*#*rev*** is the file's name and revision specifier
- *action* is the operation the file was open for: **`add`**, **`edit`**, **`delete`**, **`branch`**, or **`integrate`**
- *num* is the number of the submitting changelist
- type of the file at the given revision

To suppress the header line, use the **`-q`** (quiet) option.

To print all lines (including lines from deleted files and/or lines no longer present at the head revision), use the **`-a`** (all) option.

The output of **`p4 annotate`** is highly amenable to scripting or other forms of automated processing.

Here is a sample of the **`p4 annotate`** output with the **`-u`** option. The first column specifies the revision number. The second column, the name of the user. The third column, the modify date. The fourth column, the revised line.

```
320: mjones 2017/05/06          sr->w.digest.Clear();
172: qsmith 2016/10/27          sr->w.size.Unknown();
169: odavis 2018/04/21          sr->w.traitLot.Clear();
196: ywillson 2017/06/12        sr->w.tampered.Clear();
```

Using tagged output with the `-u` option adds three lines: one for the user, one for the time, and one for the client workspace.

```
... upper 962279
... lower 961206
... user jbond
... time 2011/03/18 11:57:14
... client bond-james
... data   else
```

Note the upper and lower entries in the tagged output. For `-a` output, these indicate the revision range where the given line appears. For `-c` output, these indicate the changelist range where the given line appears.

Options

`-a`	All lines, including deleted lines and lines no longer present at the head revision, are included. Each line includes a starting and ending revision.
`-c`	Display the changelist number, rather than the revision number, associated with each line. If you use the `-a` option and the `-c` option together, each line includes a starting and ending changelist number.
`-d` *`options`*	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" on the facing page for a listing of these options.
`-i`	Follow file history across branches. If a file was created by branching, the output includes revisions up to the branch point. The use of the `-i` option implies the `-c` option. The `-i` option cannot be combined with `-I`.
`-I`	Follow integrations into the file. If a line was introduced into the file by a merge, the source of the merge is indicated as the changelist that introduced the line. If that source was itself the result of an integration, that source will be used instead . The use of the `-I` option implies the `-c` option. The `-I` option cannot be combined with `-i`.

`-q`	Quiet mode, which suppresses the one-line header for each file.
`-t`	Force `p4 annotate` to display non-text (binary) files.
`-u`	Display the name of the user who modified the change and the date when the modification occurred.
`-T|--tab=N`	Align output to a tab stop of 8. You can specify a different tab value using the `--tab` option and specifying the desired value for *N*.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`read`

The diff options supported by `p4 annotate` are described in the following table:

Option	Name
`-db`	ignore changes made within whitespace
`-dl`	ignore line endings
`-dw`	ignore whitespace altogether

Examples

`p4 annotate file.c`	Print all lines of `file.c`, each line preceded by the revision that introduced that line into the file.
`p4 annotate -c file.c`	Print all lines of `file.c`, each line preceded by the changelist number that introduced that line into the file.
`p4 annotate -a file.c`	Print all lines of `file.c`, including deleted lines, each line preceded by a revision range. The starting and ending revision for each line are included.
`p4 annotate -a -c file.c`	Print all lines of `file.c`, including deleted lines, each line preceded by a range of changelists. The starting and ending changelists for which each line exists in the file are included.

p4 archive

Archive obsolete revisions to an archive depot.

"Syntax" on page 19

```
p4 [g-opts] archive [-n -h -p -q -t -z] -D depot FileSpec
[revSpec]
```

Description

This commands enables a Helix Core server user with **`admin`** access to move the specified revisions into a *depot* of type **`archive`**.

When files are moved into an archive depot, their last action is changed to **`archive`**.

Commands that access file content, such as **`p4 sync`** and **`p4 diff`**, skip **`archive`** revisions, but commands that do not require access to file content, such as **`p4 filelog`**, continue to report metadata concerning the archived revisions.

Criteria

Without the **`-z`** option, the command archives only revisions that meet all of the following criteria:

1. Stored in full (**`+F`**) or compressed (**`+C`**) format, rather than RCS format
2. Located in a local depot (not a **`remote`** or another **`archive`** depot)
3. Not copied or branched from another revision
4. Not copied or branched to another revision

You can use **`p4 archive -n`** for testing purposes before mounting the file system associated with the archive depot. Storage for the archive depot must be mounted before running this command without the **`-n`** option.

Tip
If you want to disable server locks when running the **`p4 archive`** command, set the value of the the `"server.locks.archive" on page 809` configurable to **`0`**.

Warning
Use with caution. The following commands permanently remove file data:

"Syntax" on page 19

- `p4 archive -p`
- `p4 obliterate -y`

Options

`-D` *depot*	Specify an archive depot to which files are to be archived.
`-h`	Do not archive head revisions.
`-n`	Do not archive revisions. Instead, report on which revisions would have been archived.
`-p`	Purge any archives of the specified files named in the archive depot. The action for affected revisions is set to `purge` on completion. **Warning** File contents are no longer accessible from `p4 restore`. **Tip** If you want to retain the metadata of purged files, do one of the following: ▪ use the `-p` option of "p4 obliterate" on page 379 (recommended) ▪ perform the two-step sequence of `p4 archive` followed by `p4 archive -p` (time-consuming)
`-q`	Quiet mode, which suppresses messages about skipped revisions.
`-t`	Archive text files (or other revisions stored in delta format, such as files of type `binary+D`)
`-z`	Can reduce the use of disk space because it includes in the archive any files that have lazy copies or are lazy copies. (A lazy copy is a reference to the location of the full file.) With this flag, only "Criteria" on the previous page 1 and 2 must be met. Unless all copies are archived, the original file remains in the depot.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`admin`

- To archive files stored in delta format, use the `-t` option.

  > **Tip**
  > There might be a computational cost associated with the manipulation of large numbers of RCS deltas.

- If a revision is stored in an archive depot, and the stored revision is accessible to the versioning service, a user can determine which archived revision to restore by using "p4 print" on page 395:

  ```
  p4 print -A -o myOutputFile //archive/depot/myFile
  ```

  This command redirects all the versions of the archived file to *myOutputFile*. The user can then request that a Helix server Administrator use `p4 restore` to restore one or more versions of the file.

  > **Tip**
  > A user with `admin` access can run "p4 fstat" on page 215 `-Ob` to see the path, revision, type, full and relative local paths of the server archive file.

Examples

`p4 archive file#3`	- Archive revisions `1` through `3` of ***`file`*** - If a single revision is specified as a file argument, **`p4 archive`** implicitly targets revisions `#1` through the specified revision for archiving
`p4 archive file#3,3`	- Archive revision `3` of ***`file`*** - To archive only a single revision ***`rev`***, use the form **`p4 archive file#`*`rev`*`,`*`rev`***
To archive files stored in delta format, use the `-t` option:	
`p4 archive -D archives -t //depot/....txt`	Use the `...` wildcard to archive the files with the `.txt` extension.

`p4 archive -D archives -t //depot/....txt@3,3`	Use the `...` wildcard to archive the files with the `.txt` extension that belong to changelist `3`.
`p4 archive -D archives -t //depot/....txt@3`	Use the `...` wildcard to archive the files with the `.txt` extension that belong to changelists `1` through `3`.

If Helix server must manipulate a large numbers of RCS deltas, the computational cost might be noticable.

Related Commands

To create a depot	`p4 depot`
To restore files from an archive depot	`p4 restore`
To obliterate files without archiving them	`p4 obliterate`

p4 attribute

Set per-revision attributes on revisions

"Syntax" on page 19

```
p4 [g-opts] attribute [-e -f -p] -n name [-v value] files ...
p4 [g-opts] attribute [-e -f -p] -i -n name file
```

Description

The `p4 attribute` command sets per-revision attributes on file revisions.

To display attributes, use `p4 fstat -Oa`.

Options

`-e`	Indicates that the value is specified in hex.
`-f`	Set the attribute on submitted files. If a propagating trait is set on a submitted file, a revision specifier cannot be used, and the file must not be currently open in any workspace.
`-i`	Read an attribute value from the standard input. Only one file argument is allowed when using this option.
`-n` *`name`*	The name of the attribute to set.
`-p`	Create a propagating attribute: an attribute whose value is propagated to subsequent revisions whenever the file is opened with `p4 add`, `p4 edit`, or `p4 delete`.
`-v` *`value`*	The value of the attribute to set. To clear an attribute, omit the `-v` option.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`write`, or `admin` to use the `-f` option

- Multiple attributes can be set or cleared by specifying multiple `-n` ***`name`*** options and an equal number of corresponding `-v` ***`value`*** options (to set) or no `-v` options (to clear).
- In distributed environments, the following commands are not supported for files with propagating attributes: `p4 copy`, `p4 delete`, `p4 edit`, `p4 integrate`, `p4 reconcile`, `p4 resolve`, `p4 shelve`, `p4 submit`, and `p4 unshelve`. Integration of files with propagating attributes from an edge server is not supported; depending on the integration action, target, and source, either the `p4 integrate` or the `p4 resolve` command will fail.

  If you use propagating attributes with files, direct these commands to the commit server, not the edge server.

p4 bgtask

Run background commands or triggers on the server.

"Syntax" on page 19

```
p4 bgtask [-b -d -i -m -w] [-e -t ]
```

Description

Enables a superuser on the p4 command-line client to run commands or programs remotely on the server in the background.
The server saves output to the server log file.

> **Tip**
> To minimize memory consumption on the server, a long-running task should minimize its output to standard output and standard error.

The superuser defines the commands in the triggers table (with `-t`) or as string arguments on the command line (with `-e`). Supports setting startup commands, replication pull commands, for a specific serverId. The superuser can specify an interval for re-running the command.

See also Defining background tasks in the triggers table in the *Helix Core Server Administrator Guide*.

Options

`-b`	Maximum number of execution errors before ceasing to attempt execution. The default is 1.
`-d`	Detach the client, which means the client does not see the output of the server-side task execution. This is particularly useful for a task that runs for a long period of time. See "About detached mode" below.
`-e`	Command string to execute.
`-i`	Seconds between command invocations. The default is 1 second. The maximum is 31 days.
`-m`	Maximum number of times the command is run. The default is 1.
`-t`	Name of the background trigger in the triggers table to execute.
`-w`	Seconds to pause after execution error before attempting the next execution. The default is 5.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

About detached mode

In detached mode, bgtask creates two separate pairs of log entries:

```
2017/10/04 14:20:35 pid 22735 user@ws0 127.0.0.1
[p4/2014.1.main/LINUX26X86_64/576838] 'user-cron -m 5 -d -i 1 -c perl
foreground_detached.pl'
```

```
2017/10/04 14:21:35 pid 22735 completed .001s 0+0us 0+0io 0+0net 2140k 0pf
```

```
2017/10/04 14:20:35 pid 22735 user@ws0 background
[p4/2014.1.main/LINUX26X86_64/576838] 'user-cron -m 5 -d -i 1 -c perl
foreground_detached.pl'
```

```
2017/10/04 14:20:46 pid 22735 completed 10.1s 0+4us 0+8io 0+0net 2164k 0pf
```

The first two lines record the start and end for the user portion of the command.

The final two lines record the start and end for the background portion.

"Syntax" on page 19

Examples

`startup.1=bgtask -t log_checker`	Upon server startup, run the `log_checker` task in the background. (See the "startup.N" on page 816 configurable)
`p4 bgtask -t verify`	Immediately runs a background task that executes the `verify` trigger that the superuser has already defined in the triggers table.
`p4 bgtask -d -i 86400 -m 5 -t p4dstate`	Run the specified trigger every day. (A day has 86400 seconds.)
`p4 bgtask -e "top -b -n 1"`	Sample the server load.
`p4 bgtask -e "powershell -Command Import-Csv C:\p4root\errors.csv"`	See the errors that have been logged.

p4 branch

Create or edit a branch mapping and its view.

"Syntax" on page 19

```
p4 [g-opts] branch [-f] branchspec
p4 [g-opts] branch -d [-f] branchspec
p4 [g-opts] branch -i [-f]
p4 [g-opts] branch [-S stream] -o
placeholderForStreamBranchSpecName
p4 [g-opts] branch [-S stream -P parent] -o
placeholderForStreamBranchSpecName
```

Description

`p4 branch` enables you to create a mapping between two sets of files for use with `p4 integrate`. A *branch view* defines the relationship between the files you're integrating from (the ***`fromFiles`***) and the files you're integrating to (the ***`toFiles`***). Both sides of the view are specified in depot syntax.

After you have created a branch mapping, you can integrate files:

```
p4 integrate -b branchspec
```

Important

- Saving a **`p4 branch`** form has no immediate effect on any files in the depot or your client workspace.
- A branch, depot, label, and workspace may not share the same name.

Streams require a placeholder value for branchspec

`placeholderForStreamBranchSpecName` indicates that you can use any value you want, because the stream will generate its own name for the branch spec.

The first syntax variant for streams:

```
p4 [g-opts] branch [-S stream] -o
placeholderForStreamBranchSpecName
```

displays the branch spec of the specified stream and its actual parent.

The second syntax variant for streams:

`p4 [`*`g-opts`*`] branch [-S `*`stream`*` -P `*`parent`*`] -o `*`placeholderForStreamBranchSpecName`*

acts as if `stream` were the child of `parent`, which currently might or might not be true.

See "Examples for stream" on page 78.

Form Fields

Field Name	Type	Description
`Branch:`	read-only	The branch name, as provided on the command line.
`Owner:`	mandatory	The owner of the branch mapping. By default, this will be set to the user who created the branch. This field is unimportant unless the `Option:` field value is `locked`. The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Access:`	read-only	The date the branch mapping was last accessed.
`Update:`	read-only	The date the branch mapping was last changed.
`Options:`	mandatory	Either `unlocked` (the default) or `locked`. If `locked`, only the `Owner:` can modify the branch mapping, and the mapping can't be deleted until it is `unlocked`.
`Description:`	optional	A short description of the branch's purpose.
`View:`	mandatory	A set of mappings from one set of files in the depot (the ***`source files`***) to another set of files in the depot (the ***`target files`***). The view maps from one location in the depot to another; it can't refer to a client workspace. For example, the branch view `//depot/main/... //depot/r2.1/...` maps all the files under `//depot/main` to `//depot/r2.1`.

Options

`-d`	Delete the named branch mapping. Files are not affected by this operation; only the stored mapping from one codeline to another is deleted. Normally, only the user who created the branch can use this option.

`-f`	Force option. Combined with `-d`, permits Helix server administrators to delete branches they don't own. Also permits administrators to change the modification date of the branch mapping (the `Update:` field is writable when using the `-f` option).
`-i`	Read the branch mapping from standard input without invoking an editor.
`-o`	Write the branch mapping to standard output without invoking an editor.
`-P` *`parent`*	For a specified stream, display the mapping that is generated by hypothetically treating the stream as if it were a child of the specified parent. Requires `-S`. One use case is to evaluate whether you want to establish a new parent for this stream.
`-S` *`stream`*	Display the mapping generated for the specified stream. This option enables you to see how change is propagated between the stream and its parent.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`

- A branch view defines the relationship between two related codelines. For example, if the development files for a project are stored under `//depot/project/dev/...`, and you want to create a related codeline for the 2.0 release of the project under `//depot/project/r2.0/...`, specify the branch view as:

  ```
  //depot/project/dev/... //depot/project/r2.0/...
  ```

  Branch views can contain multiple mappings. To specify a view, see "Views" on page 702 .

- If a path or file name contains spaces, use quotes around the path. For instance:

  ```
  //depot/project/dev/... "//depot/project/release 2.0/..."
  ```

- Paths can be excluded from a branch view to prevent a subset of files from being merged. For example, the following view entry prevents any files named `AssemblyInfo.cs` from being merged between `MAIN` and `REL`:

  ```
  -//depot/MAIN/.../AssemblyInfo.cs
  //depot/REL/.../AssemblyInfo.cs
  ```

  Similarly, entire directories can be excluded from a branch view:

  ```
  -//depot/MAIN/bin/... //depot/REL/bin/...
  ```

  To specify a view, see "Views" on page 702 .

- Branch views can also be used with `p4 diff2` with the syntax `p4 diff2 -b` ***`branchnamefromFiles`***. This will diff the files that match the pattern ***`fromFiles`*** against their corresponding ***`toFiles`*** as defined in the branch view.

Examples for stream

Display the mapping generated for the `5dev1` stream and its actual parent: `p4 branch -S //depotstream1/5dev1 -o placeholder1`
Display the mapping generated for the `5dev1` stream and the `main5` stream, treating `5dev1` as the child of `main5` (which might or might be the case): `p4 branch -S //stream1/5dev1 -P //stream1/main5 -o placeholder2`

Related Commands

To view a list of existing branch mappings	`p4 branches`
To copy changes from one set of files to another	`p4 integrate`
To view differences between two codelines	`p4 diff2`

p4 branches

List existing branch mappings.

"Syntax" on page 19

```
p4 [g-opts] branches [[-e | -E] filter] [-m max][-t] [-u user | -
-me]
```

Description

Print the list of all branch mappings currently known to the system.

Use the `-m` ***max*** option to limit the output to the first ***max*** branch mappings.

Use the `-e` or `-E` ***filter*** options to limit the output to branches whose name matches the ***filter*** pattern. The `-e` option is case-sensitive, and `-E` is case-insensitive.

Use the `-u` ***user*** option to limit the output to branches owned by the named user.

Options

`-e` ***filter***	List only branches matching ***filter*** (case-sensitive).
`-E` ***filter***	List only branches matching ***filter*** (case-insensitive).
`-m` ***max***	List only the first ***max*** branch mappings.
`-t`	Display the time as well as the date of the last update to the branch.
`-u` ***user***	List only branches owned by ***user***.
`--me`	Equivalent to `-u $P4USER`.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

"Syntax" on page 19

Related Commands

To create or edit a branch mapping	`p4 branch`

p4 cachepurge

Reclaim disk space on a replicated server.

"Syntax" on page 19

```
p4 [g-opts] cachepurge -a [-n -R -O] [-i n] [-S n] [-D file ...]
p4 [g-opts] cachepurge -f n [-n -R -O] [-i n] [-S n] [-D file
...]
p4 [g-opts] cachepurge -m n [-n -R -O] [-i n] [-S n] [-D file
...]
p4 [g-opts] cachepurge -s n [-n -R -O] [-i n] [-S n] [-D file
...]
```

Description

A replica used as a standby spare or for disaster recovery maintains a complete copy of the master server's versioned file archives. Replicas that are used for other purposes might not need to hold a copy of the content of every version of every file. If a replica is not needed for disaster recovery, you can reclaim disk space on it by periodically deleting versioned files. This is only safe to do if you have a backup of these files.

The **`p4 cachepurge`** command allows an administrator to reclaim disk space for those replicated servers that are not used for disaster recovery. File content is deleted only from the replica, not from the master server nor from any other replica. If a command that accesses purged file content is issued to this replica, the file is retrieved from the master server.

Each time the **`p4 cachepurge`** command runs, it attempts to delete enough file content from the replica to achieve the goal set by the values specified for the command parameters.

Options

`-a`	Delete all file content. This option reclaims the maximum amount of disk space, but any file content must be retrieved from the master. If the `-O` option is not specified, file names are used to determine order of deletion.
`-D file ...`	Limit action of command to the specified set of files.

`-f` *n*	Delete sufficient file content to leave *n* number of bytes of free space for the file system.
`-i` *n*	Repeat the command every *n* seconds. If you omit this option, the command runs only once.
`-m` *n*	Delete *n* file revisions. The amount of space this frees up depends on the size of the files.
`-n`	Display a preview of the **`cachepurge`** operation without deleting any files.
`-O`	Delete the files from the oldest to the newest; that is, delete older files before deleting newer files.
`-R`	Delete files in the order specified by the `-O` option. If the `-O` option is not specified, file names are used to determine order of deletion.
`-s` *n*	Delete *n* bytes of file data. This can be helpful in those cases when you can predict the growth rate of file system resources.
`-S` *n*	Do not delete the *n* most recent revisions of each file. For example, specifying `-S 1`, means that the head revision of each file is retained in the replica’s cache if it is already there.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`superuser`

p4 change

Create or edit a changelist specification.

The command `p4 changelist` is an alias for `p4 change`.

"Syntax" on page 19

```
p4 [g-opts] change [-s] [-f | -u] [[-O] changelist]
p4 [g-opts] change -d [-fsO] changelist
p4 [g-opts] change -o [-sf] [[-O] changelist ]
p4 [g-opts] change -i [-s] [-f | -u]
p4 [g-opts] change -t restricted | public [-U user] [-fuOI]
changelist
p4 [g-opts] change -U user [-t restricted | public] [-f]
changelist
p4 [g-opts] change -d -f --server=serveridchangelist
```

Description

When files are opened with `p4 add`, `p4 delete`, `p4 edit`, or `p4 integrate`, the files are listed in a *changelist*. Edits to the files are kept in the local client workspace until the changelist is sent to the depot with `p4 submit`. By default, files are opened within the default changelist, but multiple changelists can be created and edited with the `p4 change` command.

The command `p4 -Ztag change -o` displays, in addition to other information, the access time for shelved files. You can use this information to determine if a shelved file has been abandoned and needs to be removed.

To edit the description of a pending changelist, or to view the fields of a submitted changelist, use `p4 change` *`changelist`*.

Changelist number - pending versus submitted

A submitted changelist number is an integer that uniquely identifies a changelist. Helix server assigns **pending** changelist numbers in sequence. However, Helix server might renumber a changelist upon submission to ensure that the set of **submitted** changelist numbers increases with time. Submitted changelist numbers are ordinal (increasing), but not necessarily consecutive. For example, 103, 105, 108, 109.

`p4 change` brings up a form for editing or viewing in the editor defined by the environment variable `P4EDITOR`. When no arguments are provided, this command creates a numbered changelist. All files open in the default changelist are moved to the numbered changelist.

If `p4 submit` of the default changelist fails, a numbered changelist is created in its place. The changelist must be referred to by its changelist number from that point forward.

Form Fields

Field Name	Type	Description
`Change:`	Read-only	Contains the changelist number if editing an existing changelist, or `new` if creating a new changelist.
`Date:`	Read-only	Date the changelist was last modified.
`Client:`	Read-only	Name of current client workspace.
`User:`	Read-only	Name of the change owner. The owner of an empty pending changelist (that is, a pending changelist without any files in it) can transfer ownership of the changelist to another existing user either by editing this field, or by using the `-U` *`user`* option. The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Status:`	Read-only	`pending`, `shelved`, `submitted`, or `new`. Not editable by the user. The status is `new` when the changelist is created, `pending` when it has been created but has not yet been submitted to the depot with `p4 submit`, `shelved` when its contents are shelved with `p4 shelve`, and `submitted` when its contents have been stored in the depot with `p4 submit`.

"Syntax" on page 19

Field Name	Type	Description
`Type:`	Writable, optional	Type of change: `restricted` or `public`. The `Type:` field can be used to hide the change or its description from users. A `shelved` or `committed` change (as denoted in the `Status:` field) that is `restricted` is accessible only to users who own the change or have `list` permission to at least one file in the change. Public changes are displayed without restrictions. By default, changelists are `public`. A Helix server superuser can set the default changelist type (for changelists created after the configurable is set) by setting the `defaultChangeType` configurable.
`Description:`	Writable, mandatory	Textual description of changelist. This value *must* be changed before submission. If you do not have access to a restricted changelist, the description is replaced with a "no permission" message.
`ImportedBy:`	Read-only	Displays the name of the user who ran the `p4 fetch`, `p4 push`, or `p4 unzip` command that imported this change into the server. This field is primarily useful for distributed versioning (DVCS) scenarios, in which changelists are copied from one server to another, and help you correlate the changelist's basic identity as it is copied. In such configurations, Perforce recommends using the `submit.identity` configurable to enable automatic generation of changelist identities by the `p4 submit`.
`Identity:`	Writable, mandatory	Contains a label which uniquely identifies this changelist across all servers where it has been fetched, pushed, or unzipped. This field is primarily useful for distributed versioning (DVCS) scenarios, in which changelists are copied from one server to another, and help you correlate the changelist's basic identity as it is copied. In such configurations, Perforce recommends using the `submit.identity` configurable to enable automatic generation of changelist identities by the `p4 submit`.

"Syntax" on page 19

Field Name	Type	Description
`Jobs:`	List	A list of jobs that are fixed by this changelist. The list of jobs that appears when the form is first displayed is controlled by the `p4 user` form's `JobView:` setting. Jobs can be deleted from or added to this list.
`Stream:`	Writable, optional	What opened stream is to be added to this changelist. If you want to add a stream spec to the changelist, the stream spec must already be opened to the current workspace. You can remove an opened stream from this list.
`Files:`	List	The list of files being submitted in this changelist. Files can be deleted from this list, and files that are found in the default changelist can be added.

Tip
If there is a line under a field, indent that line. For example,

```
Description:
    Created by maria
```

Options

`-d`	Delete the changelist. This is usually allowed only with pending changelists that contain no files or pending fixes, but the superuser can delete changelists under other circumstances with the addition of the `-f` option. If you try to forcibly delete a changelist whose client is bound to another server, you need to specify the `--serverid` option and specify the server id of the other server. This ensures that you do not accidentally delete the changelist believing it to be on your own server.
`-f`	Force option. Allows the description, modification date, or user of a submitted changelist to be edited. Editing a submitted changelist requires `admin` or `super` access. Superusers and administrators can also overwrite read-only fields when using the `-f` option. The `-u` and the `-f` options are mutually exclusive.
`-f -d`	Forcibly delete a previously submitted changelist. Only a Helix server administrator or superuser can use this command, and the changelist must have had all of its files removed from the system with `p4 obliterate`.

`-i`	Read a changelist specification from standard input. Input must be in the same format used by the `p4 change` form.
`-o`	Write a changelist specification to standard output.
`-O`	If a changelist was renumbered on submit, and you know only the original changelist number, use `-O` and the original changelist number to view or edit the changelist.
`-I`	Specifies that the changelist number is the Identity field of a changelist.
`-s`	Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of `closed`. To leave a job unchanged, use the special status of `same`. On new changelists, the fix status is displayed as the special status `ignore`. (If the status is left unchanged, the job is not fixed by the submission of the changelist.) This option works in conjunction with the `-s` option to `p4 fix`, and is intended for use in conjunction with defect tracking systems.
`--serverid=`*`serverid`*	If you try to forcibly delete a changelist whose client is bound to another server, you need to specify the `--serverid` option and to specify the server id of the other server. This ensures that you do not accidentally delete the changelist, believing it to be on your own server. This variant of the `p4 change` command must be issued directly to the commit server.
`-t` *`type`*	Change a submitted changelist's *type* to either `restricted` or `public`.
`-u`	Update a submitted changelist. Only the `Jobs:`, `Description:`, or `Type:` fields can be updated, and only the submitter of the changelist can update the changelist. The `-u` and the `-f` options are mutually exclusive.
`-U user`	The `-U`*user* option changes the owner of an empty pending changelist. To reassign a changelist, you must either already be the changelist's owner, or a user with `admin` permissions and use the `-f` option. (Unlike manually editing the `User:` field in the `p4 change` form, this option is much more convenient for use within a trigger or script.)
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`, or `list` to use the `-o` option, or `admin` to use the `-f` option

- You should create multiple changelists when editing files corresponding to different logical tasks. For example, if edits to files `file1.c` and `file2.c` fix a particular bug, and edits to file `other.c` add a new feature, `file1.c` and `file2.c` should be opened in one changelist, and `other.c` should be opened in a different changelist.
- `p4 change`*changelist* edits the specification of an existing changelist, but does not display the files or jobs that are linked to the changelist. Use `p4 opened-c` ***`changelist`*** to see a list of files linked to a particular changelist and `p4 fixes-c` ***`changelist`*** to see a list of jobs linked to a particular changelist.
- To move a file from one changelist to another, use `p4 reopen`, or use `p4 revert` to remove a file from all pending changelists.

Examples

`p4 change`	Create a new changelist.
`p4 change -f 25`	Edit previously submitted changelist 25. Administrator or superuser access is required.
`p4 change -d 29`	Delete changelist 29. This succeeds only if changelist 29 is `pending` and contains no files.
`p4 change -f -d 121`	Force the deletion of the specified changelist

Related Commands

To submit a changelist to the depot	`p4 submit`
To move a file from one changelist to another	`p4 reopen`
To remove a file from all pending changelists	`p4 revert`

"Syntax" on page 19

To list changelists meeting particular criteria	`p4 changes`
To list opened files	`p4 opened`
To list fixes linked to particular changelists	`p4 fixes`
To link a job to a particular changelist	`p4 fix`
To remove a job from a particular changelist	`p4 fix` **-d**
To list all the files listed in a changelist	`p4 opened-c` ***changelist***
To obtain a description of files changed in a changelist	`p4 describe`***changelist***

p4 changelist

Create or edit a changelist specification.

"Syntax" on page 19

```
p4 [g-opts] change [-s] [-f | -u] [[-O] changelist]
p4 [g-opts] change -d [-fsO] changelist
p4 [g-opts] change -o [-sf] [[-O] changelist]
p4 [g-opts] change -i [-s] [-f | -u]
p4 [g-opts] change -t restricted | public [-U user] [-fuO]
changelist
p4 [g-opts] change -U user [-t restricted | public] [-f]
changelist
```

Description

The command `p4 changelist` is an alias for `p4 change`.

p4 changelists

List submitted and pending changelists.

"Syntax" on page 19

```
p4 [g-opts] changelists [-i -t -l -L -f] [-c client] [-m max] [-s status]
                              [-u user] [[FileSpec][revSpec]]
```

Description

The command **`p4 changelists`** is an alias for **`p4 changes`**.

p4 changes

List submitted and pending changelists.

The command `p4 changelists` is an alias for `p4 changes`.

"Syntax" on page 19

```
p4 [g-opts] changes [-i -t -l -L -f] [-c client] [ -e
changelist#][-m max]
                         [-r] [-s status] [-u user | --me]
[[FileSpec][revSpec]]
```

Description

Use `p4 changes` to view a list of submitted and pending changelists. When you use `p4 changes` without any arguments, all numbered changelists are listed. (The default changelist is never listed.)

By default, the format of each line is:

```
Change num on date by user@client [status] description
```

If you use the `-t` option to display the time of each changelist, the format is:

```
Change num on datehh:mm:ss by user@client [status] description
```

The ***`status`*** value appears only if the changelist is `pending` or `shelved`. The description is limited to the first 31 characters unless you provide the `-L` option for the first 250 characters, or the `-l` option for the full description.

If you provide file patterns as arguments, the changelists listed are those that affect files matching the patterns, whether `submitted` or `pending`.

Revision specifications and revision ranges can be included in the file patterns:

- To limit output to only those changelists made from the named client workspace or the named user, use the `-c client` or the `-u user` option
- To limit output to only those changelists with the provided ***`status`*** (`pending`, `shelved`, or `submitted`) value, use the `-s status` option
- To limit output to only changes that are greater or equal to the specified changelist number, use the `-e changelist#` option

In a distributed configuration, changes that are pending or shelved on an edge server are visible via the `p4 changes` command on other servers in the installation.

Administrators can use the `-f` option to view restricted changelists.

You can combine options and file patterns to limit the changelists that are displayed.

Tip
You can also use the `-m` ***max*** option to limit output to ***max*** changes:

`p4 changes -m 5` shows the five most recent changes.

To reverse the order of the list so that the earliest changes appear before the most recent changes, use the `-r` option:

`p4 changes -r -m 5` shows the five oldest changes.

Note
The global `-u` in "Global options" on page 690 has a different meaning than the `p4 changes -u` option:

$ **p4 -u bruno changes -u gale**

where `p4 -u bruno` uses the global option to change the current user to `bruno`, and `-u gale` asks for a list of the changes that user `gale` made.

Options

`-c` ***client***	List only changes made from the named client workspace
`-f`	View restricted changes (requires `admin` permission)
`-i`	Include changelists that affected files that were integrated with the specified files
`-l`	List long output, with the full text of each changelist description
`-L`	List long output, with the full text of each changelist description truncated at 250 characters
`-m` ***max***	List only the highest numbered ***max*** changes.
`-r`	Reverse the order of the list, earliest first instead of most recent first
`-s` ***status***	Limit the list to the changelists with the given status (`pending`, `submitted`, or `shelved`)
`-t`	Display the time as well as the date of each change
`-u` ***user***	List only changes made from the named user
`--me`	Equivalent to `-u $P4USER`
g-opts	See "Global options" on page 690

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

- If `p4 changes` is called with multiple file arguments, the sets of changelists that affect each argument are evaluated individually. The final output is neither combined nor sorted. The effect is the same as calling `p4 changes` multiple times, once for each file argument.
- If files are not specified, `p4 changes` limits its report according to whether or not changes are public or restricted. Restricted `submitted` or `shelved` changes are not reported unless you either own the change or have `list` permission for at least one file in the change. Restricted `pending` (but unshelved) changes are visible only to the change owner.
- `p4 changes` *`myfile`*`#have` accesses the db.have table lockless for the duration of the compute phase.

Examples

`p4 changes file.c#3`	Show the changelists associated with file versions #1, #2, and #3.
`p4 changes file.c#3,6`	Show the changelists associated with file versions #3, #4, #5, and #6.
`p4 changes -e 800 file.c`	Show the changelists that are greater or equal to changelist number `800`
`p4 changes -m 5 //depot/project/...`	Show the last five submitted, pending, or shelved changelists that include any file under the `project` directory.
`p4 changes -m 5 -c eds_elm`	Show the last five submitted, pending, or shelved changelists from client workspace `eds_elm`.
`p4 changes -m 5 -s submitted -u edk`	Show the last five submitted changelists from user `edk`.
`p4 changes file.c@2010/05/01,2010/06/01`	Show any changelists that include file `file.c`, as mapped to the depot through the client view, during the month of May 2010.
`p4 changes -m 1 -s submitted`	Output a single line showing the changelist number of the last submitted changelist.

`p4 changes @2011/04/01,@now`	Show all changelists submitted from April 1, 2011 to the present.
`p4 changes @2011/04/01`	Show all changelists submitted *before* April 1, 2011.
`p4 changes -r -m 1 //depot/project/branch/...`	Show the first change on the specified branch.

Related Commands

To submit a pending changelist	`p4 submit`
To create a new pending changelist	`p4 change`
To read a detailed report on a single changelist	`p4 describe`

p4 check-permission (graph)

Check access permission granted to a user of a repo.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 check-permission -n //repo/name -u user [-r ref] -p permission
p4 check-permission -n //repo/name -u user [-r ref] -p all
```

Description

- Administrator or super user can use this command to check the permissions of any user of a specified repo.

  Note
  An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

- Any user can check the permission of another user that is at the same, or a lower, level. For example, user `bruno`, can check whether `marie` has the `create-repo` permission to the `//gd1/name2` repo provided that `bruno` has this same permission or higher for this same repo. For details about types of permissions, see "p4 grant-permission (graph)" on page 227.

Options

`-n`	Applies to the repo with the specified name.
`-u`	Applies to the specified user.
`-p`	Applies to the specified permission.
`all`	Used to list all the permissions explicitly
`-r`	(Optional) Applies to the specified branch or tag.

Examples

`p4 check-permission -n //repo/qa/main -u bruno -p write-ref`	Does user bruno have the `write-ref` permission to that repo? The output might be this confirmation: `write-ref`
`p4 check-permission -n //repo/qa/main -u bruno -p all`	List each and every permission that user bruno has to that repo. The output might be: `read, write-ref, write-all, create-repo`

Related commands

Display the permissions for the specified depot of type graph or a repo	"p4 show-permission (graph)" on page 522
Display a user-centric view of the permissions for repos across multiple repos or depots of type graph	"p4 show-permissions (graph)" on page 523
Assign a graph permission to a user or group	"p4 grant-permission (graph)" on page 227

p4 clean

Restore workspace files to match the state of corresponding depot files.

The `p4 clean` command is equivalent to the `p4 reconcile -w` command.

"Syntax" on page 19

```
p4 [g-opts] clean [-e -a -d -I -m -l -n] [file ...]
```

Description

The `p4 clean` command takes the following actions when finding inconsistencies between files in a user's workspace and corresponding depot files:

1. Files present in the workspace, but missing from the depot are **deleted** from the workspace.

   > **Warning**
   > Before you issue the command to have `p4 clean` delete files, make sure that you have successfully navigated to the correct directory. Otherwise, you might unintentionally delete local files that you want to keep.

2. Files present in the depot, but missing from your workspace. The version of the files that have been synced from the depot are added to your workspace.
3. Files modified in your workspace that have not been checked in are restored to the last version synced from the depot.

To limit the scope of `p4 clean` to add, edit, or delete, use the `-a`, `-e`, or `-d` options. For example, using the `-a` option deletes any new files in your workspace.

By default, `p4 clean` does not check files and/or paths mentioned in the `P4IGNORE` file if they have been added (rather than edited). Use the `-I` option to override this behavior and ignore the `P4IGNORE` file.

To preview the set of proposed workspace reconciliation actions, use the `-n` option.

Options

`-a`	Added files: Find files in the workspace that have no corresponding files in the depot and delete them.

`-d`	Deleted files: Find those files in the depot that do not exist in your workspace and add them to the workspace.
`-e`	Edited files: Find files in the workspace that have been modified and restore them to the last file version that has synced from the depot.
`-I`	Do not perform any ignore checking, which means to ignore any settings specified by `P4IGNORE` for added files.
`-m`	Compare the file sync or submit time (in the depot) with the file **modification time** (in the workspace) to help determine whether the file has changed. Normally Helix server uses file digests to determine whether files in the workspace differ from the head revisions of these files in the depot. This can be time consuming for large files. But when the timestamps are the same, the costly digest comparisons can be skipped. This option is only relevant if you are using **clean** to find changed files rather than files that were deleted or added.
`-l`	Display output in local file syntax with relative paths, similar to the workspace-centric view of `p4 status`.
`-n`	Preview the results of the operation without performing any action.
file	The files whose versions you want reconciled with their latest depot versions. If you omit this parameter, the files in your local working directory are used.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`read`

- The `p4 clean` command produces output in depot syntax. To see file names and paths in local syntax, you must use the `-l` option, or use `p4 status`. Compare the output of the following commands, one without the `-l` option, and the other one with the option.

```
C:\test\local\client\copy\l>p4 clean -n bar
//depot/copy/l/bar#none - deleted as c:\test\local\client\copy\l\bar
C:\test\local\client\copy\l>p4 clean -n -l bar
//depot/copy/l/bar#none - deleted as bar
```

- When called without arguments, `p4 clean` adjusts the specified files in your workspace to reflect their latest state in the depot.

"Syntax" on page 19

Related Commands

An equivalent for **`p4 reconcile -w`**	`p4 reconcile`

p4 client

Create or edit a client workspace specification and its view.

The command `p4 workspace` is an alias for `p4 client`.

"Syntax" on page 19

```
p4 [g-opts] client [-f] [-t template] [-T type] [clientname]
p4 [g-opts] client -o [-t template] [-T type] [clientname]
p4 [g-opts] client -d [-f [-Fs]]clientname
p4 [g-opts] client -s [-S stream | -t clientname] clientname
p4 [g-opts] client -S stream [[-c change] -o] [clientname]
p4 [g-opts] client -i [-f]
p4 [g-opts] client -d -f --serverid=serverid [-Fs]
```

Description

A Helix server *client workspace* is a set of files on a user's machine that mirror a subset of the files in the depot. More precisely, it is a named *mapping* of depot files to workspace files. Use the `p4 client` command to create or edit a client workspace specification. Invoking this command displays a form in which the user enters information so that the Helix server can maintain the workspace.

The `p4 client` command puts the client spec into a temporary file and invokes the editor configured by the environment variable "P4EDITOR" on page 656. For new workspaces, the client name defaults to the "P4CLIENT" on page 647 environment variable, if set, or to the current host name. Saving the file creates or modifies the client spec.

The client view, which is specified in the `p4 client` form's `View:` field, specifies the mapping between files in the workspace and depot.

The mapping between a client workspace file and a depot file:

- can specify the same or different relative locations
- can specify the same or different names
- is typically a many-to-many mapping, such as `path/to/....html path/from/....htm`, where `...` is a wildcard and the fourth "." is the literal `.` before the file extension. See the Wildcards in "File specifications" on page 695.

When `p4 client` completes, the new or altered workspace specification is stored in the Helix server database. The files in the workspace are not touched. The new view does not take effect until the next `p4 sync`.

To submit changes to a stream, you must associate the stream with a workspace by using the command **`p4 client -S`** ***`stream clientname`***. To change the stream associated with a workspace, use the command **`p4 client -s -S`** ***`stream clientname`***.

Tip

- The client storage type cannot be changed after client is created. For example, a `readonly` client cannot be changed into a `writeable` client.
- The terms "client, "workspace", and "workspace client" mean the same thing.

About mapping in client workspace views

To exclude matching files, precede the mapping with the `-` minus sign.

If more than one mapping line refers to the same files, the later mapping line overrides the earlier one.

map multiple server directories to the same client workspace directory

To map multiple server directories to the same client workspace directory, use the `+` sign to overlay the later mapping on an earlier one.

```
//depot/project1/... //bruno-client/project
+//depot/project2/... //bruno-client/project
```

If files match both the earlier and later mappings, the file matching the later mapping is used. For more details, see Map different depot locations to the same workspace location in *Helix Core Server User Guide*.

map a server directory to multiple client workspace directories

To map the same server directory to more than one client workspace directory, use the `&` sign.

```
//depot/... //bruno-client/...
&//depot/and/tools/... //bruno-client/and/utility1/...
&//depot/and/tools/... //bruno-client/and/utility2/...
```

Files mapped in this way are read-only. For more details, see Map a single depot path to multiple locations in a workspace in *Helix Core Server User Guide*.

Form Fields

Field Name	Type	Description
`Client:`	Read-only	The client workspace name, as specified in the `P4CLIENT` environment variable or its equivalents. When called without a *clientname* argument, `p4 client` operates on the workspace specified by the `P4CLIENT` environment variable or one of its equivalents. If called with a *clientname* argument on a `locked` workspace, the workspace specification is read-only. Be aware of the "Limitations on characters in filenames and entities" on page 699.
`Owner:`	Writable, optional	The name of the user who owns the workspace. The default is the user who created the workspace. The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Update:`	Read-only	The date the workspace specification was last modified.
`Access:`	Read-only	The date and time that the workspace was last used in any way. The access time is only valid for the server where the client resides, which is where the client was created or where the client was moved to. This is an issue only in a commit-edge architecture. Reloading a workspace with `p4 reload` does not affect the access time.

Field Name	Type	Description
Note Add your content		
		from any host. The hostname must be provided exactly as it appears in the output of `p4 info` when run from that host. **Note** This field is meant to prevent accidental misuse of client workspaces on the wrong machine. Providing a host name does not guarantee security, because the actual value of the host name can be overridden with the `-H` option to any `p4` command, or with the `P4HOST` environment variable. For a similar mechanism that does provide security, use the IP address restriction feature of `p4 protect`.
`Description:`	Writable, optional	A textual description of the workspace. The default text is `Created by owner`.
`Root:`	Writable, mandatory	The directory (on the local host) relative to which all the files in the `View:` are specified. The default is the current working directory. The path must be specified in local file system syntax. (See "Syntax forms" under "File specifications" on page 695) If you change this setting, you must physically relocate any files that currently reside there. On Windows client machines, you can specify the root as `null` to enable you to map files to multiple drives.

Field Name	Type	Description
`AltRoots:`	Writable, optional	Up to two optional alternate client workspace roots. Helix server applications use the first of the main and alternate roots that match the application's current working directory. Use the `p4 info` command to display the root being used. This enables users to use the same Helix server client workspace specification on multiple platforms, even those with different directory naming conventions. If you are using multiple or alternate workspace roots (the `AltRoots:` field), you can always tell which root is in effect by looking at the `Client root:` reported by `p4 info`. If you are using a Windows directory in any of your workspace roots, you must specify the Windows directory as your main workspace root and specify your other workspace roots in the `AltRoots:` field. For example, an engineer building products on multiple platforms might specify a main client root of `C:\Projects\Build` for Windows builds, and an alternate root of `/staff/userid/projects/build` for any work on UNIX builds.
`Options:`	Writable, mandatory	A set of switches that control particular workspace options. See "Options field" on page 110.

Field Name	Type	Description
`SubmitOptions:`	Writable, mandatory	Options to govern the default behavior of `p4 submit`. ▪ `submitunchanged` All open files (with or without changes) are submitted to the depot. This is the default behavior of Helix server. ▪ `submitunchanged+reopen` All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist. ▪ `revertunchanged` Only those files with content, type, or resolved changes are submitted to the depot. Unchanged files are reverted. ▪ `revertunchanged+reopen` Only those files with content, type, or resolved changes are submitted to the depot and reopened in the default changelist. Unchanged files are reverted and *not* reopened in the default changelist. ▪ `leaveunchanged` Only those files with content, type, or resolved changes are submitted to the depot. Any unchanged files are moved to the default changelist. ▪ `leaveunchanged+reopen` Only those files with content, type, or resolved changes are submitted to the depot. Unchanged files are moved to the default changelist, and changed files are reopened in the default changelist. This option is similar to `submitunchanged+reopen`, except that no unchanged files are submitted to the depot.
`LineEnd:`	Writable, mandatory	Configure carriage-return/linefeed (CR/LF) conversion. See "Processing line endings" on page 111.
`Stream:`	Writable, optional	Associates the workspace with the specified stream. Helix server generates the view for stream-associated workspaces. You cannot modify that view manually.

"Syntax" on page 19

Field Name	Type	Description
`StreamAtChange:`	Writable, optional	A changelist number that sets a back-in-time view of a stream. When `StreamAtChange` is set, running `p4 sync` (when called with no arguments) updates the workspace to files at this changelist revision, instead of the head revision. You cannot submit changes (`p4 submit` returns an error) when `StreamAtChange` is set, because the workspace view no longer reflects the current stream inheritance. This field is ignored unless the `Stream` field is also set to a valid stream.
`ServerID:`	Writable, optional	If set, restricts usage of the workspace to the named server. If unset, use is allowed on master server and on any replicas of the master other than Edge servers. **Important** To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting `serverID`, but support `P4NAME` for backward compatibility.
`View:`	Writable, multi-line	Specifies the mappings between files in the depot and files in the workspace. See `p4 help views` for more information. A new view takes effect on the next `p4 sync` operation.

Field Name	Type	Description
`ChangeView:`	Writable, optional, multi-line	Restricts access to depot paths to a particular point in time. Files specified for the ChangeView field are read-only: they may be opened but not submitted. For example: `//depot/path/...@1000` Revisions of the files in the specified path will not be visible if they were submitted after the specified changelist number. Files matching a ChangeView path may not be submitted. **Note** The names of automatic labels can be used as specifiers in import mappings on stream specs and in the **ChangeView** on client specs.
`Type:`	Writable, optional	Specifies the type of client: `writeable` — The default. `readonly` — for short lived clients used in build automation scripts. Such clients cannot edit or submit files. `partitioned` — similar to `readonly` but with the additional ability to edit and submit files using that client. **Note** Using writeable clients in build automation scripts can fragment the `db.have` table, which records the files that a client has synced. If you are experiencing performance issues when syncing, consider using use a read-only or partitioned client. A client of type `readonly` or `partitioned` is assigned its own `db.have` table. The location of this table must first be specified with the `client.readonly.dir` configurable by an administrator. See also "Using read-only and partitioned clients in automated builds" in *Helix Core Server Administrator Guide*

Tip
If there is a line under a field, indent that line. For example,

```
Description:
    Created by maria
```

Options

`-d` ***`clientname`***	Delete the specified client workspace whether or not the workspace is owned by the user. The workspace must be unlocked and must have no opened files or pending changes. (The `-f` option permits Helix server administrators to delete locked workspaces owned by other users.) Clients can be deleted even if they have shelved files (see `-Fs` option). If you try to forcibly delete a client bound to another server, you need to specify the `--serverid` option and specify the server id of the other server. This ensures that you do not accidentally delete the client believing it to be connected to your own server.
`-f`	Allows the last modification date, which is normally read-only, to be set. Administrators can use the `-f` option to delete or modify locked workspaces owned by other users. Use of this option requires `admin` access granted by `p4 protect`.
`-Fs`	Allows the deletion of a client even when that client contains shelved changes. The client is deleted and the shelved changes are left intact. (You must use the `-f` option with the `-Fs` option.)
`-i`	Read the client workspace specification from standard input.
`-o`	Write the client workspace specification to standard output.
`-o -c` ***`change`***	When used with `-S` ***`stream`***, displays the workspace specification that would have been created for a *stream* at the moment the *change* was submitted.
`-s`	Switch workspace view. To switch the workspace view to a stream, specify `-S` ***`stream`***. To switch the view defined for another workspace, specify `-t`*clientname*. Switching views is not allowed in a client that has opened files. The `-f` option can be used with `-s` to force switching with opened files. View switching has no effect on files in a client workspace until `p4 sync` is run.
`--serverid=`***`serverid`***	If you try to forcibly delete a client bound to another server, you need to specify the `--serverid` option and specify the server id of the other server. This ensures that you do not accidentally delete the client believing it to be connected to your own server. This variant of the `p4 client` command must be issued directly to the commit server.

`-S stream`	Associates the workspace with the specified stream, which is used to generate its workspace view.
`-t` *`clientname`*	Copy client workspace *clientname*'s view and options into the `View:` and `Options:` field of this workspace. If you specify a default client template using the `template.client` configurable, you do not have to specify this option.
`-T type`	By default, clients are `writeable`. You can also set `type` to: ■ `readonly`, which prevents files from being opened or submitted ■ `partitioned`, which allows files to be opened and submitted
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- Use quotation marks to enclose depot-side or client side mappings of file or directory names that contain spaces.
- Spaces in workspace names are translated to underscores. For example, typing the command `p4 client "my workspace"` creates a workspace called `my_workspace`.
- By default, any user can edit any workspace specification with `p4 client`*`clientname`*. To prevent this from happening, set the `locked` option and use `p4 passwd` to create a password for the workspace owner.
- To specify a workspace on Windows that spans multiple drives, use a `Root:` of `null`, and specify the drive letters in the workspace view. For instance, the following workspace spec with a `null` root maps `//depot/main/...` to an area of the `C:` drive, and other releases to the `D:` drive:

```
Client: eds_win
Owner:  edk
Description:
        Ed's Windows Workspace
Root:   null
Options:        nomodtime noclobber
SubmitOptions:  submitunchanged
View:
        //depot/main/...     "//eds_win/c:/Current Release/..."
        //depot/rel1.0/...    //eds_win/d:/old/rel1.0/...
        //depot/rel2.0/...    //eds_win/d:/old/rel2.0/...
```

Tip
Indent each line under a field, as shown above for `Description:` and `View:`

Use lowercase drive letters when specifying workspaces across multiple drives.

Options field

The `Options:` field contains values separated by spaces. Each of the options has two possible settings:

Option	Choice	Default
`[no]allwrite`	If `allwrite` is set, unopened files in the workspace are left writable. If `allwrite` is set and `noclobber` is NOT specified, a safe synchronization is performed. A setting of `allwrite` leaves unopened files writable by the current user. It does not set filesystem permissions to ensure that files are writable by any user of a multi-user system.	`noallwrite`
`[no]clobber`	If `clobber` is set, a `p4 sync` overwrites ("clobbers") files in the workspace that have the same name as the newly-synced files if those files are writable but unopened. If `allwrite` is set and `noclobber` is NOT specified, a safe synchronization is performed.	`noclobber`
`[no]compress`	If `compress` is set, the data stream between the user’s workstation and the Perforce service is compressed. The compress option speeds up communications over slow links by reducing the amount of data that has to be transmitted. Over fast links, the compression process itself may consume more time than is saved in transmission. In general, compress should be set only for line speeds under T1.	`nocompress`

Option	Choice	Default
`[un]locked`	Grant or deny other users permission to edit or delete the workspace specification. (To make a `locked` workspace effective, the workspace's owner's password must be set with `p4 passwd`.) If `locked`, only the owner is able to use or edit the workspace specification. Helix server administrators can override the lock by using the `-f` (force) option with `p4 client`.	`unlocked`
`[no]modtime`	For files *without* the `+m` (modtime) file type modifier: ■ If `modtime` is set, the modification date (on the local filesystem) of a newly synced file is the datestamp *on the file* when the file was last modified. ■ If `nomodtime` is set, the modification date is the date and time *of sync*, regardless of version. For files *with* the `+m` (modtime) file type modifier: the modification date (on the local filesystem) of a newly synced file is the datestamp on the file when the file was submitted to the depot, regardless of the setting of `modtime` or `nomodtime` on the client. Files with the `modtime` (`+m`) type are intended for udevelopers who need to preserve original timestamps on files. The use of `+m` in a file type overrides the workspace's `modtime` or `nomodtime` setting. For a more complete discussion of the `+m` modifier, see "File types" on page 707.	`nomodtime` (date and time of sync) for most files. Ignored for files with the `+m` file type modifier.
`[no]rmdir`	If **rmdir** is set, when a command such as `p4 sync` causes a *non-empty workspace directory to become empty*, that workspace directory is deleted. **Tip** If **rmdir** is set, `p4 sync` might remove your current working directory. If so, change to an existing directory before continuing with your work.	`normdir`

Processing line endings

The `LineEnd:` field controls the line-ending character(s) used for text files in the client workspace. Changing the line end option does not actually update the client files; you can refresh them with `p4 sync -f`.

The `LineEnd:` field accepts one of five values:

Option	Meaning
`local`	Use mode native to the client (default)
`unix`	UNIX-style (and Mac OS X) line endings: `LF`
`mac`	Mac pre-OS X: `CR` only
`win`	Windows- style: `CR` + `LF`.
`share`	The `share` option normalizes mixed line-endings into UNIX line-end format. The `share` option does not affect files already synced into a workspace; however, when files are submitted to the depot, the share option converts all Windows-style `CR/LF` line-endings and all Mac-style `CR` line-endings to the UNIX-style `LF`, leaving lone `LF` line-endings untouched. When you sync your workspace, line endings are set to `LF`. If you edit the file on a Windows machine, and your editor inserts `CR` characters before each `LF`, the extra `CR` characters do not appear in the archive file. The most common use of the `share` option is for users of Windows workstations who mount their UNIX home directories as network drives; if you sync files from UNIX, but edit the files on a Windows machine. The `share` option implicitly edits the file(s) during a submit. As a consequence, if you have set the `LineEnd` field to `share` in your client spec, the `p4 resolve` command may prompt you to edit the file before resolving.

For more information, see the Support Knowledgebase article, "CR/LF Issues and Text Line-endings".

Working with streams

Without `-s`, the `-S` ***`stream`*** option can be used to create a new client spec dedicated to a stream. If the client spec already exists, and `-S` is used without `-s`, it is ignored. Using `-S` sets the client's `Stream` field. The special syntax `-S` ***`//a/stream`***`@`***`changelist`*** can be used to set both `Stream` and `StreamAtChange` at the same time.

The `-S` ***`stream`*** option can be used with `-o -c` ***`change`*** to inspect an old stream client view. It yields the client spec that would have been created for the stream at the moment the change was recorded.

Working with build servers

A server of type build-server (see `p4 help server`) is a replica that supports build farm integration, and the `p4 client` command may be used to create or edit client workspaces on a build-server. Such workspaces may issue the `p4 sync` command in addition to any read-only command supported by the replica. For more information, run `p4 help buildserver`.

When creating or editing a client workspace for a build-server, the client specified by the optional `name` argument, as well as the client specified by the `P4CLIENT` environment variable or via the global `-c client` argument must not exist, or must be restricted to this server; this command may not be used to create or edit a workspace that is not restricted to this build-server.

Working with read-only clients

Build automation scripts, which routinely create, sync, and tear down clients, may fragment the `db.have` table over time. To avoid this, you can specify the type `readonly` for these clients. Such clients cannot add, delete, edit, integrate, or submit files, but this should not be an issue in build scripts.

A readonly client is assigned its own personal `db.have` database table, and the location of this table is specified using the `client.readonly.dir` configurable.

To set up a read-only client:

1. Set the `client.readonly.dir` configurable to the directory where the db.* tables for the client should be stored.

   For example, if you create a read-only client whose name is `myroc` and you set `client.readonly.dir` to `/perforce/1`, then syncing files using this client will write to the following database

   `/perforce/1/server.dbs/client/`*`hashdir`*`/db.myroc`
2. Set the `Type` field of the client spec to `readonly`.

Including Graph Depot repos in your client

See "p4 client (graph)" on the facing page.

Examples

`p4 client`	Edit or create the workspace specification named by the value of `P4CLIENT` or its equivalents.
`p4 client -t gale bruno`	Create or edit a workspace named `bruno`, opening the form with the field values and workspace options in the workspace named `gale` as defaults.
`p4 client -d release1`	Delete the workspace named `release1`.
`p4 client -o` *`build-client`* `| sed "s/Created by/Created by automated build/" | p4 client -i`	Automate the modification of the `Description:` field in a client specification. This example uses `-o` and `-i` to redirect from standard out to standard in.

Related Commands

To list all workspaces known to the system	`p4 clients`
To read files from the depot into the workspace	`p4 sync`
To open new files in the workspace for addition to the depot	`p4 add`
To open files in the workspace for edit	`p4 edit`
To open files in the workspace for deletion	`p4 delete`
To write changes in workspace files to the depot	`p4 submit`
Graph depot version	"p4 client (graph)" below

p4 client (graph)

Create or edit a client workspace specification

The command `p4 workspace` is an alias for `p4 client`.

"Syntax" on page 19

```
p4 [g-opts] client  [-f] [-t template] -T graph [graphClientName]
p4 [g-opts] client -o [-f] [-t template] -T graph
[graphClientName]
```

Description

To modify Graph Depot files using p4 commands, your workspace must obey several additional rules, beyond those described in "p4 client" on page 100:

- Specify `Type: graph`
- Specify `View:` ***`map`***, where *map* describes the files in the repos that are to be used by this client

Hybrid client that maps to both classic and graph depots

You can create a client spec that maps solely to a classic depot, solely to a graph depot, or to a hybrid client that combines both.

If you create a `hybrid client`, the options are:

- a client of a classic depot in which graph depot files are read-only and classic files are editable - see "classic client spec that includes a mapping to a graph directory " on the next page

- a client of a graph depot in which classic files are read-only and graph depot files are editable - see "graph depot client spec that includes a mapping to a classic directory " on the facing page
- a read-only client in which both classic and graph depot files are read-only

For more information about depots of type `graph`, see:

- "Including Graph Depot repos in your client" on page 113 in `p4 client`
- "Working with depots of type graph" on page 153 in `p4 depot`
- "Stream and graph depot - .git suffix and repo path" on page 541 in "p4 stream" on page 539

Options

`-f`	Allows the last modification date, which is normally read-only, to be set. Administrators can use the `-f` option to delete or modify locked workspaces owned by other users. Use of this option requires `admin` access granted by `p4 protect`.
`-o`	Write the client workspace specification to standard output.
`-T type`	By default, clients are `writeable`. You can also set `type` to: ■ `readonly`, which prevents files from being opened or submitted ■ `partitioned`, which allows files to be opened and submitted
g-opts	See "Global options" on page 690.

Examples

classic client spec that includes a mapping to a graph directory

This client spec associated with a graph depot spec does not include a line that specifies "`Type: graph`", so updates to Graph Depot paths are prohibited.

```
Client:   mixed_client1
Update:   2017/04/04 09:51:30
Access:   2017/04/04 09:51:48
Owner:    bruno
Host:     laptop153
Description:
    Created by bruno for a writable "classic" depot with read-only access
to a graph depot.
Root:     /home/user/mixed_client1
```

```
Options:  noallwrite noclobber nocompress unlocked nomodtime normdir
SubmitOptions:    submitunchanged
LineEnd:  local
View:
    //depot/main/projectA/... //mixed_client1/depot/main/projectA/...
    //repo/projectB/... //mixed_client1/repo/projectB/...
```

Note
`depot` is the default name for a writable "classic" depot.

`repo` is the default name for a depot of type `graph`.

This client spec provides write access to `projectA`, which belongs to a writable "classic" depot, and read-only access to `projectB`, which belongs to a depot of type `graph`. One use case for such a client is to support including files from both kinds of projects into a single software build.

With this client spec, `p4 sync` results in:

```
//depot/main/projectA/projA.txt#1 - added as /home/user/mixed_
client1/depot/main/projectA/projA.txt
//depot/main/projectA/readme.txt#1 - added as /home/user/mixed_
client1/depot/main/projectA/readme.txt
//repo/projectB/projB.txt - added as /home/user/mixed_
client1/repo/projectB/projB.txt
//repo/projectB/readme.txt - added as /home/user/mixed_
client1/repo/projectB/readme.txt
```

Tip
If this client attempts to edit a file in a graph depot, an error message appears. For example,

```
$ p4 edit aRepoFile.c
//repo/projectB/aRepoFile.c - can only edit file in a local depot
```

graph depot client spec that includes a mapping to a classic directory

The client spec associated with a graph depot spec MUST include a line that specifies "`Type: graph`".

```
Client:   mixed_client2
Update:   2018/05/16 19:01:30
Access:   2018/05/16 19:01:59
```

```
Owner:    bruno
Host:     laptop153
Description:
Created by bruno for a writable graph depot, with read-only access to a
classic depot.
Root:     /home/user/mixed_client2
Options:  noallwrite noclobber nocompress unlocked nomodtime normdir
SubmitOptions:    submitunchanged
LineEnd:  local
Type:     graph
View:
   //repo/projectB/... //mixed_client2/repo/projectB/...
   //depot/main/projectA/... //mixed_
client2/depot/main/classicProjectA/...
```

Tip
If this client attempts to edit a file in a classic depot, an error message appears. For example,

```
$ p4 edit aClassicFile.c
aClassicFile.c - no such file(s).
```

p4 clients

List all client workspaces currently known to the system.

"Syntax" on page 19

```
p4 [g-opts] clients [-t] [-u user | --me] [[-e|-E] filter] [-m
max]
              [-S stream]  [-a | -s serverID]
```

```
p4 [g-opts] clients -U
```

Description

`p4 clients` lists all the client workspaces known to the Helix Core server. Each workspace is reported on a single line of the report. The format of each line is:

`Client` ***`clientname moddate root clientrootdescription`***

For example:

```
Client paris 2009/02/19 root /usr/src 'Joe's client'
```

describes a client workspace named `paris`, last modified on February 19, 2009 with a root of `/usr/src`. The description of the workspace entered in the `p4 client` form is `Joe’s client`.

Use the `-m` ***`max`*** option to limit the output to the first ***`max`*** client workspaces.

Use the `-e` or `-E` ***`filter`*** options to limit the output to clients whose name matches the ***`filter`*** pattern. The `-e` option is case-sensitive, and `-E` is case-insensitive.

Use the `-u` ***`user`*** option to limit the output to workspaces owned by the named user.

The command `p4 workspaces` is an alias for `p4 clients`.

Options

`-a`	List all client workspaces, not just workspaces bound to this server.
`-e` ***`filter`***	List only client workspaces matching ***`filter`*** (case-sensitive).
`-E` ***`filter`***	List only client workspaces matching ***`filter`*** (case-insensitive).

`-m` ***max***	List only the first ***max*** client workspaces.
`-s` ***serverID***	List only client workspaces bound to the specified ***serverID***. On an edge server, the `-s` option defaults to the edge server's serverID.
`-S` ***stream***	List client workspaces associated with the specified stream.
`-t`	Display the time as well as the date of the last update to the workspace.
`-u` ***user***	List only client workspaces owned by ***user***.
`--me`	Equivalent to `-u $P4USER`.
`-U`	List only client workspaces unloaded with `p4 unload`.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Examples

`p4 clients -m 5`	List a maximum of five workspaces for any type of depot.

Related Commands

To edit or view a client workspace specification	`p4 client`
To see the name of the current client workspace and other useful data	`p4 info`
To view a list of Helix server users	`p4 users`

p4 clone

Clone a new local Helix server from a remote server.

"Syntax" on page 19

```
p4 [-u user] [-d dir] [-c client] clone [-m depth] [-v] -p port -
r remote
p4 [-u user] [-d dir] [-c client] clone [-m depth] [-v] -p port -
f filespec
```

Description

When you clone from a remote server, you copy the portion of its contents that you want to work with into your local server.

Options

`-c` ***`client`***	Specifies the client name. If not specified, defaults to the name established with the **`p4 init`** command.
`-d` ***`directory`***	Specifies the directory in which the new Helix server is initialized. If not specified, defaults to the current directory.
`-f` ***`filespec`***	Specifies a *filespec* in the remote server to use as the path to clone. The Helix Core server uses this path to determine the stream setup in the local server. Helix server also uses this file specification to determine the stream setup in the personal server. Specifying the filespec also creates a default remote spec called `origin`. The `-f` preceding the filespec is optional. You can simply issue the command `p4 clone //`***`file/path`***.
`-m` ***`depth`***	Specifies the maximum number of revisions of each file to clone; a *shallow* clone.
`-p` ***`port`***	Specifies the address (`P4PORT`) of the remote server you wish to clone from. This flag is optional. If not specified,`p4 clone` uses the remote server specified by the `P4PORT` environment variable.
`-r` ***`remotespec`***	Specifies the remote spec call *remotespec* installed on the remote server to use as a template for the clone and stream setup.

"Syntax" on page 19

`-u` *`username`*	Specifies the Helix server user.
`-v`	Enables verbose mode.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`read` on the remote server.

Examples

```
p4 -u bruno -d Ace clone -p perforce:1666 -f //depot/main/...
```

As user `bruno`, clone the server `perforce:1666`, retrieving only the files and history from the remote server path `//depot/main/...`

Related Commands

To initialize a Helix server	`p4 init`

p4 configure

Set, view, and manage server configuration variables.

"Syntax" on page 19

```
p4 [g-opts] configure set [server_id#]variable=value
p4 [g-opts] configure unset [server_id#]variable
p4 [g-opts] configure show [allservers | variable]
p4 [g-opts] configure history [allservers | variable]
```

Description

Note
Although we recommend the syntax above, using "p4 serverid" on page 505 instead of "P4NAME" on page 670, the following syntax is still supported.

```
p4 [g-opts] configure set [P4NAME#]variable=value
p4 [g-opts] configure unset [P4NAME#]variable
p4 [g-opts] configure show [allservers | P4NAME | variable]
p4 [g-opts] configure history [allservers | P4NAME | variable]
```

Configuration variables are used to control and customize the behavior of the Helix Core service. A configurable setting might affect the client, the server, or a proxy.

Tip
An alternative is using "p4 server" on page 493, which conveniently allows some configuration in the server spec. See the "p4 server" on page 493 topic on the `DistributedConfig:` field.

Important
The configuration variables are described both in this Reference and at the command line:

This Reference	Command-line
See "Configurables - alphabetical list" on page 717 (indicates which ones require stopping the server.)	`p4 help configurables`
See "Environment and registry variables" on page 637	`p4 help environment`

Tip
The super user can use `p4 [g-opts] configure history [allservers | P4NAME | variable]` to display the history of configurables, which are recorded by the 2019.2 server onwards. See the "Options" on page 126.

Precedence

The following table shows how the value of a configurable variable is set, where `1` overrides `2`, `2` overrides `3`, and `3` overrides `4`:

Precedence	How the value is set
1	Command line "`-v`" options that are passed at server startup. For example: `$ p4d -v "net.keepalive.idle" on page 774=2700`
2	Persistently, using the `p4 configure set` command. This method allows you to set the specified configurable for a named server or for any server.
3	Using environment variables. ▪ For Windows, use "p4 set" on page 513 `-S Perforce`, such as `p4 set -S Perforce P4DEBUG=""net.keepalive.idle" on page 774=2700"` and the value will persist ▪ On Unix, use the `export` command, which does not persist **Note** Certain server-related configurables are read-only. For example, the `p4 configure` or `p4 configure set` commands cannot change the value of the `P4ROOT` or `P4JOURNAL` environment variables.
4	Using default values (no action required).

Tip
You can use `K` and `M` to represent large numbers. For example, `10M` is the default value for the "dm.shelve.maxfiles" on page 743 configurable.

Viewing the values of configuration variables on all servers

To display the configuration across all servers, use

```
p4 configure show allservers
```

The output might be similar to:

```
any: lbr.autocompress = 1
any: submit.allowbgtransfer = 1
paris-edge: P4LOG = /home/perforce/servers/edge1/log
headquarters-commit: P4LOG = /home/perforce/servers/commit-hq/log
```

where `any` means a configurable defined on the commit server that is used by **all** connected servers unless specifically overridden in that particular server's configuration.

Viewing the values of configuration variables on one server

To display the configuration state of the current server, a named server, or any configurable, including a Helix Core environment variable, use

```
p4 configure show
```

Each configurable is displayed along with its value, where the entry in the parentheses `( )` indicates how the value was set:

If the output line is ...	the value was set by ...
`P4PORT=20192 (-p)`	`p4d -p`
`monitor=2 (-v)`	`p4d -v`
`net.parallel.max: 10 (configure)`	`p4 configure`
`serverid=commit (serverid)`	"p4 serverid" on page 505

Note that `monitor` is set to `2`. To find out whether a specific configurable variable has been set in more than one way, specify that configurable variable:

```
p4 configure show monitor
```

which might output:

```
monitor=2 (-v)
monitor=10 (configure)
```

to indicate that `p4d -v` set "monitor" on page 766 to `2`, and `p4 configure` set `monitor` to `10`. Because `p4d` has precedence over `p4 configure`, the output of `p4 configure show` indicates that `monitor` is set to `2`.

Unsetting a value

To remove a custom setting of a configurable, use the `p4 configure unset` command.

After installing Helix server, it is good practice to:

- enable process monitoring by setting "monitor" on page 766 to `1` or `2`
- require ticket-based authentication by setting "security" on page 804 to `3` or `4`

- prevent the automatic creation of new users by setting `"dm.user.noautocreate" on page 745` to **1** or **2**
- force new users that you create to reset their passwords by setting `"dm.user.resetpassword" on page 747` to **1**

Stopping the server for some configurables

Changes to most configurables take effect immediately. For example, ■ `"monitor" on page 766` (enable/disable the **`p4 monitor`** command) ■ `"security" on page 804` (set the security level).
Changes to **`P4AUTH`**, **`P4PORT`**, the `"startup.N" on page 816` configurables used in replicated environments, `"net.tcpsize" on page 791`, and `"net.backlog" on page 770` require a restart. To restart the server, use `"p4 admin" on page 58` **`restart`**.
For certain configurables, such as "ssl.tls.version.min" on page 815: After you change the value of this configurable, you must explicitly "stop" the server. **Note** **`p4 admin restart`** is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service. For Windows, see Starting and stopping the Helix server.

Setting configurables in multi-server environments

Servers can be identified by name. In replicated and multi-server environments, a master can control the settings of multiple replicas by specifying the server name as part of the configurable. For example, the following command sets the value of the `"serviceUser" on page 814` configurable for an edge server (**`tokyo_edge`**). The command is executed on the commit server.

```
$ p4 configure set tokyo_edge#serviceUser=svc_tokyo_edge
```

See Deployment architecture in the *Helix Core Server Administrator Guide*.

Accessing configurables when the server is down

If the Helix server is not running or you cannot access the server, you can use the **`p4d`** command to list, set, and unset server configurables:

- To list all server configuration variables, use the `-cshow` option. For example:

```
$ p4d -r $P4ROOT -cshow
```

- To set or unset values, use `-cset` or `-cunset`. For example:

```
$ p4d -r $P4ROOT "-cset myServer#auth.ldap.timeout=30"
$ p4d -r $P4ROOT "-cunset myServer#db.replication"
```

For more information, see the Support Knowledgebase article, "Accessing Server Configuration Variables".

Options

`set` ***`variable`*** `=`***`value`***	Sets the named variable to the provided value.
`unset` ***`variable`***	Unsets the named variable.
`show`	Shows the current configuration of the server currently specified by `P4PORT`. **Note** Regarding the maximum size of the `db.monitor` table, see "db.monitor.shared" on page 735 configurable, and note that `p4 configure show` indicates the actual maximum, but `p4 configure show allservers` indicates a manual preference that is only enforced if sufficient memory is available.
`show allservers`	Shows the configuration variables for all servers known to the system.
`show` ***`variable`***	Shows the setting of the specified configuration variable.
`show` ***`P4NAME`***	If a Helix server was invoked with `-In` *P4NAME* or with the `P4NAME` environment variable set to a server name, shows the settings of the named server. **Important** To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting `serverID` instead `P4NAME`.

`history`	Records any changes to the value of each configurable in the `db.configh` table, which contains the name of the configurable, the targeted `serverId`, the old value, the new value, the `user` who made the change, the `datetime` of the change, the `configureVersion` number, and the `serverId` on which the change occurred.
`history allservers`	Shows the history of configuration variables for all servers known to the system.
`history` *`variable`*	Shows the history of the specified configuration variable.
`history` *`P4NAME`*	If a Helix server was invoked with `-In` *P4NAME* or with the `P4NAME` environment variable set to a server name, shows the history of the settings of the named server. **Important** To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting `serverID` instead of `P4NAME`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Examples

`p4 configure set "Replica1#startup.1=pull -i 1"`	On the server named Replica1, set the startup.n configurable to poll `1` every second
`p4 configure set "rpl.labels.global" on page 800=1`	The server returns: `For server 'any', configuration variable 'rpl.labels.global' set to '1'` where `'any'` means this setting applies to ALL servers, unless there is a local override for the same setting.

Related Commands

The **`p4 configure`** command replaces many of the settings formerly set by **`p4 counter`**.

To list all counters and their values	**`p4 counters`**
To set Helix Core server system variables	"p4 set" on page 513

p4 copy

Copy files from one location in the depot to another.

"Syntax" on page 19

```
p4 [g-opts] copy [-c change] [-n -f -v -q] [-m max] fromFile[rev]
toFile
p4 [g-opts] copy [-c change] [-n -f -v -q] [-m max] -b branch [-
r]
                  [toFile[rev] ...]
p4 [g-opts] copy [-c change] [-n -f -v -q] [-m max] -b branch -s
                 fromFile[rev] [toFile ...]
p4 [g-opts] copy [-c change] [-n -f -v -q] [-m max] -S stream [-P
parent]
                 [-Fr] [toFile[rev] ...]
```

Description

Using the client workspace as a staging area, the **`p4 copy`** command propagates an exact copy of the source files to the specified target by branching, replacing, or deleting files. No manual resolve is required. Changes in the target that were not previously merged into the source are overwritten. To update the target, submit the files. To revert copied files, use the **`p4 revert`** command.

Target files that are identical to the source are not affected by the **`p4 copy`** command unless you use the **`-f`** option. When **`p4 copy`** creates or modifies files in the workspace, it leaves them read-only. You can use **`p4 edit`** to make them writable.

Options

`-b` ***`branch`***	Specify a branch view to be used to determine source and target files.
`-c` ***`change`***	Open the files in the specified pending changelist rather than in the default changelist.

`-f`	Force the creation of extra revisions in order to explicitly record that files have been copied. Deleted source files are copied if they do not exist in the target, and files that are already identical are copied if they are not connected by existing integration records.
`-F`	Force copy operation; perform the operation when the target stream is not configured to accept a copy of the source. To determine a stream's expected flow of change, use `p4 istat`.
`-m` *`max`*	Specify the maximum number of files to copy, to limit the size of the operation.
`-n`	Preview the copy.
`-P` *`parent`*	Specify a target stream other than the parent of the source stream. Requires `-S`.
`-q`	Quiet mode, which suppresses normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.
`-r [`*`toFile`*` [`*`rev`*`] ...]`	Reverse the mappings in the branch view, integrating from the target files to the source files. Requires the `-b` option.
`-s` *`fromFile`* `[`*`rev`*`] [`*`toFile`*` ...]`	Treat *`fromFile`* as the source and both sides of the branch view as the target. To restrict the scope of the target further, specify the optional *`toFile`* parameter. Overrides the `-r` option, if specified. Requires `-b`.
`-S` *`stream`*	Specify the source stream. Changes are copied to its parent. You can use the `-r` option to reverse direction. To submit copied stream files, the current client must be switched to the target stream or to a virtual child stream of the target stream.
`-v`	Do not sync the target files. By default, `p4 copy` syncs the target files. If a large number of files is involved and you do not require the files to be present in your workspace, you can minimize overhead and network traffic by specifying `-v`.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required

fromFile: Yes
toFile: No
	No	`read` access for `fromFile`
		`open` access for `toFile`

You can use a revision specifier to select the revision to copy; by default, the head revision is copied. The revision specifier can be used on ***fromFile*** or ***toFile***, but not on both. When used on ***toFile***, it refers to source revisions, not to target revisions. You may not use a range as a revision specifier.

Examples

`p4 copy -S //projectX/dev`	Create a stream quickly (without checking integration history)
`p4 copy //projectX/dev/... //projectX/main/...`	Promote work from a development stream to the mainline

Related Commands

Update a child stream with a more stable parent stream	"p4 merge" on page 366
Propagate changes after considering all integration history and scheduling resolves, if necessary	"p4 integrate" on page 265

p4 counter

Access, set, increment, or delete a persistent variable.

"Syntax" on page 19

```
p4 [g-opts] counter countername
p4 [g-opts] counter [-f -v] counternamevalue
p4 [g-opts] counter [-f] -d countername
p4 [g-opts] counter [-f -v] -i countername
p4 [g-opts] counter [-f] -m [pair list]
p4 [g-opts] counter --from oldvalue --to newvaluecountername
```

Description

Counters provide long-term variable storage for scripts that access Helix server. Counters can be assigned textual values as well as numeric ones.

The command includes the following variants:

- The variant `p4 counter` ***`countername`*** returns the value of variable *countername*.

  If a counter does not exist, its value is returned as zero; counter names are not stored in the database until set to a nonzero value.

- The variant `p4 counter` ***`countername value`*** sets the value of variable *countername* to *value*. If *countername* does not already exist, it is created.

- The variant `p4 counter -d` ***`countername`*** deletes the counter *countername*. This has the same effect as setting the counter to zero.

- The variant `p4 counter -i` ***`countername`*** increments the counter by one and returns the new value. Use this option instead of a value argument.

- The variant `p4 counter -m` ***`pair list`*** defines multiple operations to be performed. Each operation is defined by a value pair in the pair list. To set a counter use a name and value; to delete a counter use a `-` (hyphen) followed by the name. See "Examples" on page 134.

  This variant is useful in multi-server environments where running individual commands is likely to introduce unwanted latency.

- The final variant (`--from` … `--to`) sets the specified counter to the new value only if the current value of the counter is ***`oldvalue`***. A counter that has never been set or that has been deleted cannot be set using this syntax variant.

This variant effectively provides a compare-and-set function that can be used as a building block for higher-level tools and process that use counters.

Helix server uses a number of counters in the course of its regular operations. These might be useful to various tools. For example, review tools can keep track of which changes have been reviewed and which changes are still under review by writing such status information into counters.

For the list of Helix server counters, see "p4 counters" on page 135. Superusers can use the `-f` option to force changes to counters. However, changes to counters do incur risk.

The last changelist number known to the Perforce service (the output of `p4 counter change`) includes pending changelists created by users, but not yet submitted to the depot. It can be useful to know the changelist number of the last *submitted* changelist, which is the second field of the output of the command:

```
$ p4 changes -m 1 -s submitted
```

The last changelist number *successfully* submitted (that is, no longer pending) to the Perforce service is held in the `maxCommitChange` counter.

Options

`-d` *`countername`*	Delete variable *countername*.
`-i` *`countername`*	Increment variable *countername* by 1 and return the new value. This option can only be used with numeric counters.
`-f`	Set or delete counters that are reserved for use by Helix server (listed in `p4 help counters`). **Never** set the `change` counter to a value that is lower than its current value. Only operators or `super` users can use this flag.
`-m` *`pair list`*	Specify a list of operations to be performed. Each operation is defined by a value pair in the pair list. To set a counter, use a name and value; to delete a counter use a `-` (hyphen) followed by the name. See "Examples" on the facing page.
`-v`	Display the previous value of the specified counter after the counter has been set or incremented.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list` to display a counter's value `review` to set a new value `super` to use the `-f` option

Examples

`p4 counter mycounter 123`	Set the value of a counter `mycounter` to `123`. If `mycounter` does not exist, it is created. Requires `review` access.
`p4 counter mycounter`	Display the value of `mycounter`. If `mycounter` does not exist, its value is displayed as `0`. Requires `list` access.
`p4 counter -m firstcounter 5 secondcounter 4`	Set two counters.
`p4 counter -m - xset -yset`	Delete two counters.
`p4 counter -m firstcounter 6 -secondcounter`	Set one counter; delete one counter.

Related Commands

To configure the versioning service	`p4 configure`
To list all configurables and their values	`p4 configure` **show**
To list all counters and their values	`p4 counters`
List and track changelists	`p4 review`
List users who have subscribed to particular files	`p4 reviews`

p4 counters

Display list of long-term variables used by Helix server and associated scripts.

"Syntax" on page 19

```
p4 [g-opts] counters [-e nameFilter] [-m max]
```

Description

Helix server uses counters as variables to store the number of the last submitted changelist and the number of the next job. **`p4 counters`** provides the current list of counters, along with their values:

`change`	Current change number.
`job`	Current job number.
`journal`	Current journal number
`lastCheckpointAction`	Data about the last complete checkpoint
`logger`	Event log index used by **`p4 logger`**.
`maxCommitChange`	The last changelist number successfully submitted to the Helix server.
`traits`	Internal trait lot number used by **`p4 attribute`**.
`upgrade`	Server database upgrade level.

Options

`-e` ***`nameFilter`***	List counters with a name that matches the ***`nameFilter`*** pattern, for example: `p4 counters -e 'mycounter-*'`
`-m` ***`max`***	List only the first ***`max`*** counters.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Related Commands

To view or change the value of a counter	`p4 counter`

p4 cstat

Dump change/sync status for current client workspace.

"Syntax" on page 19

```
p4 [g-opts] cstat [files ...]
```

Description

The `p4 cstat` command lists changes that are required, already synced, or partially synced to the current client workspace.

The output is returned in the tagged format used by the `p4 fstat` command:

```
... change changenum
... status have|need|partial
```

A client workspace might **have** change 222 (that is, be synced to changelist 222), but depending on what others have done after the sync, could either:

- **need** change 223 (if no files in changelist 223 have yet been synced),
- or have a **partial** sync of changelist 223 (if some, but not all, of the revisions in changelist 223 have been synced).

Options

g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Related Commands

To check for integrations needed for a stream	`p4 istat`

p4 dbschema

Report information about metadata in the database on the Helix server.

"Syntax" on page 19

```
p4 [g-opts] dbschema [tablename[:tableversion]]
```

Description

The **`p4 dbschema`** command reports information about the database structure in which the Helix Core server stores metadata.

By default, all current tables are reported. To restrict output to a specified table, use the name of the corresponding **`db.`*`tablename`*** file in the Helix server root.

The results are returned as tagged output.

This command is intended for systems integrators.

Options

`tablename`	Restrict output to the specified table name.
`tableversion`	Restrict output to the specified table version.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	**`super`**

Examples

`p4 dbschema db.protect`	Display information about the **`db.protect`** database table.

p4 dbstat

Display size or simple statistics for one or more database tables.

"Syntax" on page 19

```
p4 [g-opts] dbstat [-h][-f] {-a | dbtable ...}
p4 [g-opts] dbstat -s
```

Description

The **p4 dbstat** command displays statistics on the internal state of the database on the Helix server. The ***dbtable*** corresponds to the **db.*** files in your server’s root directory. This command is typically used in conjunction with Perforce technical support to estimate disk seeks due to sequential database scans.

Using the **-f** with the **-h** options, a histogram of free page distribution is shown, but the distance report is omitted.

To obtain size information, use **p4 dbstat -s**.

Warning
Because **p4 dbstat** blocks write access to the database while it scans the tables, use this command with care. You will most often use this command when working with Perforce technical support.

Options

-a	Display statistics for all tables.
-f	Displays a page count, free pages, and percent free data for the specified table(s).
-h	Display a histogram showing distances between leaf pages.
-s	Report file sizes of database tables.
dbtable	Display statistics for the specified table (for instance, **db.have**, **db.user**, and so on.) As you are most likely to need data for a particular table when working with support, they will let you know the name of the table for which you need information.
g-opts	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

p4 dbverify

Perform low-level verification of the database tables.

"Syntax" on page 19

```
p4 [g-opts] dbverify [-t db.tablename] [-U][-v]
```

Description

The **`p4 dbverify`** command performs a series of low-level structural integrity checks on the database tables. Run this command periodically to determine if tables have become damaged.

By default, all current tables are verified. This can be computationally expensive and might require scheduled user downtime on large systems. To restrict verification to a specified table, use the name of the corresponding **`db.`*`tablename`*** file in the Helix server root.

For a faster integrity check, use the **`-U`** option, which looks for tables with non-zero unlock counts. Each database table has an accompanying unlock count. When data are ready to be written to a table, the table's unlock count is incremented and the table is locked. When the write is complete, the table is unlocked and its unlock count is decremented. If the process that writes the data does not unlock the table, the unlock count remains incremented. This might happen if the system goes down before the write is complete.

`p4 dbverify -U` has minimal performance impact. However:

- the presence of a non-zero unlock count does not necessarily indicate corruption
- the presence of a zero unlock count does not guarantee data integrity

Options

`-t db.`*`tablename`*	Restrict verification to the specified table name.
`-U`	Perform a less-detailed validation
`-v`	Provide verbose information on the verification.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`operator` or `super`

Tip
`p4 dbverify` is mostly equivalent to **`p4d -xv`**, but some differences exist. See the Managing the database tables topic in *Helix Core Server Administrator Guide*.

p4 delete

Open file(s) in a client workspace for deletion from the depot.

"Syntax" on page 19

```
p4 [g-opts] delete [-c changelist] [-n -k -v] [--remote=remote]
file ...
```

Description

The **`p4 delete`** command opens file(s) in a client workspace for deletion from the depot. The files are immediately removed from the client workspace, but are not deleted from the depot until the corresponding changelist is committed with **`p4 submit`**.

Although it will *appear* that a deleted file has been deleted from the depot, the file is never truly deleted, as older revisions of the same file are always accessible. Instead, a new head revision of the file is created which marks the file as being deleted. If **`p4 sync`** is used to bring the head revision of this file into another workspace, the file is deleted from that workspace.

A file that is open for deletion does not appear on the workspace's *have list*.

Options

`-c` ***`changelist`***	Opens the files for **`delete`** within the specified changelist. If this option is not provided, the files are linked to the default changelist.
`-k`	Delete the file on the shared versioning service, but keep a copy of the deleted file in your workspace.
`-n`	Preview which files would be opened for delete, without actually changing any files or metadata.
`--remote=`*`remote`*	Opens the file for delete in your personal server, and additionally — if the file is of type **`+l`** — takes a global exclusive lock on the file in the shared server from which which you cloned the file. For more information, see the section Support for exclusive locking in the Fetching and Pushing chapter of *Using Helix Core Server for Distributed Versioning*.

`-v`	Delete a file that is not synced into the client workspace. To use this option, specify these files in depot syntax; because such files are not synced, client syntax or local syntax can introduce ambiguities in the list of files to delete. (If the files are synced, **`p4 delete -v`** ***`file`*** removes the files from your workspace in addition to opening them for deletion.) To delete a set of files without transferring them to your workstation when another version of these files already exists in your workspace, use `p4 sync-k` ***`file`***, followed by **`p4 delete -k`** ***`file`***. This allows you to delete a file that is in the depot without affecting the file (by the same name) in your workspace. For example, if you have `myfile` version 5 in the depot and `myfile` version 6 in your workspace, this sequence of commands, allows you to keep version 6 but delete version 5. If you don't do this and just do **`p4 delete -v`**, it will delete version 5 in the depot and version 6 in the workspace.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

- A file that has been deleted from the client workspace with **`p4 delete`** can be reinstated in the client workspace and removed from the pending changelist with `p4 revert`. To do this, you must revert the deletion before submitting the changelist.
- Helix server does not prevent users from opening files that are already open; its default scheme is to allow multiple users to open a file simultaneously, and then resolve file conflicts with `p4 resolve`. To prevent someone else from opening a file once you've opened it, use `p4 lock`. To determine whether or not another user already has a particular file open, use `p4 opened-a` ***`file`***.
- Using an `+S`*`n`* file modifier results in special behavior when you delete and read a file: no file reversions are deleted that were submitted before the add or delete. For example, if a file of type `+S2` is marked as deleted in revision 5, and then re-added with the same file type and modifier, revisions 3 and 4 are not purged.

Examples

`p4 delete //depot/README`	Opens the file called `README` in the depot's top level directory for deletion. The corresponding file within the workspace is immediately deleted, but the file is not deleted from the depot until the default changelist is submitted.

`p4 delete -c 40 file`	Opens *`file`* in the current workspace for deletion. The file is immediately removed from the client workspace, but won't be deleted from the depot until changelist 40 is committed with `p4 submit`.

Related Commands

To open a file for add	`p4 add`
To open a file for edit	`p4 edit`
To copy all open files to the depot	`p4 submit`
To read files from the depot into the client workspace	`p4 sync`
To create or edit a new changelist	`p4 change`
To list all opened files	`p4 opened`
To revert a file to its unopened state	`p4 revert`
To move an open file to a different changelist	`p4 reopen`

p4 delete (graph)

Delete an existing file from the repo.

"Syntax" on page 19

```
p4 delete [-c changelistNumber] file ...
```

Description

Opens a repo file for deletion. If the file is synced in the client workspace, it is removed.

Options

`-c` *`changelistNumber`*	Opens the files for `delete` within the specified changelist. If this option is not provided, the files are linked to the default changelist.
`-n`	Preview which files would be opened for delete, without actually changing any files or metadata.

p4 depot

Create or edit a depot specification.

Warning
A branch, depot, label, and workspace may not share the same name.

"Syntax" on page 19

```
p4 [g-opts] depot [-t type] depotname
p4 [g-opts] depot -d [-f] depotname
p4 [g-opts] depot -o [-t type] depotname
p4 [g-opts] depot -i
```

Note
This command behaves differently for depots of type `graph`. For details, see the section "Working with depots of type graph" on page 153.

Description

The Helix server stores files in shared repositories called depots. By default, there is one `local` depot named `depot` on every Helix server installation.

To create or edit a depot, use `p4 depot depotname` and edit the fields in the depot spec form. Depots can be of type `local`, `stream`, `remote`, `archive`, `spec`, `unload`, `tangent`, or `graph`.

Specifying the `-t` option creates a depot spec for the depot type you specify. For example:

```
$ p4 depot -o -t stream] mystreamdepot
```

Creates:

```
Depot:          mystreamdepot
Owner:          user
Date:           2018/12/21 15:57:50
Description:    Created by user.
Type:           stream
StreamDepth:    //mystreamdepot/1
Map:            mystreamdepot/...
```

You can edit this spec. (To change the `StreamDepth`, see "Working with stream depots" on page 151).

Alternatively, you can pipe the output stream to the `p4 depot` command:

```
$ p4 depot -o [-t stream] mystreamdepot | p4 depot -i
```

Note
A depot created with `p4 depot` is not physically created on disk until files have been added to it with `p4 add`.

Users are not able to access a new depot created with `p4 depot` until the ability to access the depot is granted with `p4 protect`.

Form Fields

Field Name	Type	Description
`Depot:`	Read-Only	The depot name as provided in `p4 depot` ***`depotname`***. Be aware of the "Limitations on characters in filenames and entities" on page 699.
`Owner:`	Writable	The user who owns the depot. By default, this is the user who created the depot. The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Description:`	Writable	A short description of the depot’s purpose. Optional.

Field Name	Type	Description
`Type:`	Writable	`local`, `remote`, `stream`, `spec`, `unload`, `archive`, `tangent`, or `graph`. ▪ A `local` depot is writable, and is the default depot type. Files reside in the server's root directory and are managed directly by the server. By default, there is one `local` depot named `depot` on every Helix server installation. ▪ A `remote` depot references files that reside on other servers, and cannot be written to. See "Working with remote depots" on page 150. ▪ A `stream` depot is also writable, but contains *streams*, a type of branch that includes hierarchy and policy. See "Working with stream depots" on page 151. ▪ The `spec` depot, if present, automatically archives edited forms. See "Working with spec depots" on page 151. ▪ The `unload` depot, if present, holds infrequently-used metadata (about old client workspaces and labels) that has been unloaded with the `p4 unload` command. For more information, see "Unloading infrequently-used metadata" in the *Helix Core Server Administrator Guide*. ▪ An `archive` depot is used in conjunction with the `p4 archive` and `p4 restore` commands to facilitate offline (or near-line) storage of infrequently-accessed revisions, typically large binaries. ▪ A `tangent` depot defines a read-only location that holds tangents created by the `p4 fetch -t` command. The tangent depot named `tangent` is automatically created by `p4 fetch -t` if one does not already exist. ▪ A depot of type `graph` can contain Git repos. ▪ An `extension` depot stores files related to Helix Core Extensions. See *Helix Core Extensions Developer Guide*.
`Address:`	Writable	If the `Type:` is `remote`, the address should be the `P4PORT` address of the remote server. If the `Type:` is `local` or `spec`, this field is ignored.

Field Name	Type	Description
`Suffix:`	Writable	If the `Type:` is `spec`, this field holds an optional suffix for generated paths to objects in the spec depot. See "Working with spec depots" on page 151 for more information. The default suffix is `.p4s`. You do not need a suffix to use the spec depot, but supplying a file extension to your Helix server's versioned specs enables users of GUI client software to associate these specifications with a preferred text editor. If the `Type:` is `local` or `remote`, this field is ignored.
`StreamDepth:`	Writable	The default is one level below the name of the depot. You can set a different value when you create the stream. See "Working with stream depots" on page 151.
`Map:`	Writable	If the `Type:` is `local`, `spec`, or `archive`, set the map to point to the relative location of the depot subdirectory. The map must contain the `...` wildcard; for example, a `local` depot `new` might have a `Map:` of `new/...`. If the `Type:` is `remote`, set the map to point to a location in the remote depot's physical namespace, for example, `//depot/new/rel2/...`. This directory will be the root of the local representation of the remote depot. For more information, see "Providing map information" on the facing page.
`SpecMap:`	Writable	For spec depots, an optional description of which specs should be saved, expressed as a view.

Options

`-d` *`depotname`*	Delete the depot *depotname*. The depot must not contain any files; the Helix server superuser can remove files with `p4 obliterate`. If the depot is `remote`, `p4 obliterate` must still be run: no files are deleted, but any outstanding client or label records referring to that depot are eliminated. The name of a depot may not be the same as the name of a branch, client workspace, or label.
`-f`	By default, when you delete a depot, the directory specified by the `Map:` field (typically under `P4ROOT`) must be empty. Use the `-f` option to remove all files even if the directory is not empty.

`-i`	Read a depot specification from standard input, or from a file in the case of `p4 depot -o -t stream mystreamdepot2 > filename` where content of ***`filename`*** is the depot spec.
`-o` ***`depotname`***	Write a depot specification to standard output.
`-t` ***`type`***	The type of the depot: `local`, `remote`, `spec`, `stream`, `unload`, `archive`, or `tangent`. To support Git workflows, a special type is `graph`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Providing map information

For a local depot, the `Map` field specifies the filesystem location of the archive contents for files in the depot. This location can be either relative or absolute. To store a depot's versioned files on another volume or drive, specify an absolute path in the `Map` field. This path need not be under `P4ROOT`.

- If the location is absolute, for example, `/p4/depots/depot/...`, no further interpretation is needed.
- If the location is relative, for example, `Ace/...`, the location is interpreted relative to the value of `P4ROOT`, unless the `server.depot.root` configurable is set, in which case it is interpreted relative to the value of that variable.

  Take care if you introduce the `server.depot.root` form of addressing in an existing installation. If you want to set it to a value other than `P4ROOT`, you should first update your existing depot `Map` values to make sure they are all absolute. You can then set the `server.depot.root` variable without disrupting anything. After that, you can go back and update your existing depot maps if you so desire.

Working with remote depots

If you are using `remote` depots, the machine that hosts the Perforce service (that is, the machine specified in `P4PORT`) is configured to permit your Helix server application to read files from a different Perforce service. Remote depots are restricted to read-only access; Helix server applications cannot `add`, `edit`, `delete`, or `integrate` files in the depots on the other servers. For more information about remote depots, see the *Helix Core Server Administrator Guide*.

Remote depots are accessed by a virtual user named `remote` (or, if configured, by the service user configured for the service that originates the request), and by default, all files on any Helix server installation can be accessed remotely. To limit or eliminate remote access to a particular server, use `p4 protect` to set permissions for user `remote` (or the accessing site's service user) on that server.

For example, to eliminate `remote` access to all files in all depots on a particular server, set the following permission on that server:

`read user remote * -//...`

Because remote depots can only be used for `read` access, it is not necessary to remove `write` or `super` access.

Neither service users nor the virtual `remote` user consume Helix server licenses.

If your server accesses remote depots by means of a service user, your service user must have a valid ticket for the server that is hosting the remote depot.

See Remote depots and multi-server development in *Helix Core Server Administrator Guide*.

Working with spec depots

The `spec` depot, if present, tracks changes to user-edited forms, such as client workspace specifications, jobs, and branch mappings. There can be only one `spec` depot per server. Files in the spec depot are automatically generated by Helix server, and are represented in Helix server syntax as follows:

`//specdepotname/formtype/objectname[suffix]`

For example, if the spec depot is named `spec` and uses the default suffix of `.p4s`, you can obtain the history of changes to `job000123` by typing:

`p4 filelog //spec/job/job000123.p4s`

After you have created the spec depot, use `p4 admin updatespecdepot` to pre-populate it with the current set of client, depot, branch, label, typemap, group, user, and job forms.

For spec depots, the `SpecMap:` field can be used to control which specs are versioned. By default, all specs (`//spec/...`) are versioned. To exclude the protections table from versioning, configure the spec depot's `SpecMap:` as follows:

```
SpecMap:
    //spec/...
    -//spec/protect/...
```

Adding or changing the spec mapping only affects future updates to the spec depot; files already stored in the spec depot are unaffected.

See the *Helix Core Server Administrator Guide* on Spec Depot.

Working with stream depots

A stream is a special type of branch that has hierarchy and policy. A `stream` depot is the container for a set of streams. To create a stream depot, provide the following information to the Depot spec :

"Syntax" on page 19

- `Depot` (for the name of the stream depot)
- `Owner`
- `Date` (of creation)
- `Type` (must be `stream`)
- `StreamDepth`

About StreamDepth

By default, the files in a stream are stored one (1) level below the depot name. For example:

```
//myStreamDepot/myStream1
//myStreamDepot/myStream2
//myStreamDepot/myStream3
```

To specify a non-default value for the `StreamDepth:` field, use an integer or forward slashes.

By integer:	`Depot:  myStreamDepot`
	`Owner: bruno`
	`Date:  2018/11/21 11:10:32`
	`Description:`
	`Created by bruno.`
	`Type:  stream`
	`Address: local`
	`Suffix:  .p4s`
	`StreamDepth: 2`

By number of forward slashes, which in this case is 2:	`Depot:  myStreamDepot`
	`Owner: bruno`
	`Date:  2018/11/21 11:10:32`
	`Description:`
	`Created by bruno.`
	`Type:  stream`
	`Address: local`
	`Suffix:  .p4s`
	`StreamDepth: //myStreamDepot/foo/bar`

Tip
You might want to create a stream that corresponds to a particular project, group, or location. For example, you can use
`StreamDepth: //deepStream/1/2/3/4`
or
`StreamDepth: 4`

However, for simplicity, we recommend a `StreamDepth` of `3` or less. If you want a StreamDepth of `3`, such as:

```
//myStream/organization/project/mainline
//myStream/organization/project/dev
//myStream/organization/project/release1
//myStream/organization/project/release2
```

specify the value for the **StreamDepth:** field in one of the following two ways:

`StreamDepth: 3`

or

`StreamDepth: //myStream/1/2/3`

Important
A stream's `name` and `StreamDepth` can only be assigned once. After that, you cannot rename a stream or change its depth.

Tip
For a complete explanation of the different types of streams and how to use them, see Streams in the *Helix Core Server User Guide* because the *Helix Core Server Administrator Guide* information on Stream Depots is minimal.

Working with depots of type graph

A depot of type `graph` is used to store Git repos in the Helix server.

To create a depot of type `graph` named `graphDepot1`, issue the following command:

```
p4 depot -t graph graphDepot1
```

To display a list of depots of type `graph`, issue the following command:

```
p4 depots -t graph
```

A default depot of type graph named `repo` is automatically created when the Helix server server is created or upgraded to 2017.1 or later.

After you create a Git repo or a graph depot, `p4 depots` lists all depots, whether or not they are of type `graph`.

Note
A depot of type `graph` does not use the `p4 protect` mechanism at the file level. Instead, a graph depot supports the Git model with a set of permissions for an entire repository (`repo`) of files. For details, see `p4 grant-permission`.

"Syntax" on page 19

For in-depth information on working with depots of type `graph`, see the Helix4Git Administrator Guide.

Related Commands

To list all depots known to the Helix server	`p4 depots`
To populate a new depot with files	`p4 add`
To add mappings from an existing client workspace to the new depot	`p4 client`
To remove all traces of a file from a depot	`p4 obliterate`
To limit remote access to a depot	`p4 protect`
To archive files into an archive depot	`p4 archive`
To restore files from an archive depot	`p4 restore`

p4 depots

Display a list of depots known to the Helix server.

"Syntax" on page 19

```
p4 [g-opts] depots [[-e|-E] nameFilter] -t type]
```

Description

Display a list of depots.

If a depot is excluded in the protections table for a given user, that user does not see the depot in the output of this command.

`Operator` users can run this command. (To learn about `operator` users, see the `p4 user` usage notes.)

Options

`-e` ***nameFilter***	Lists depot specs with a name that matches the *nameFilter* pattern. `p4 depots -e h*x*` finds depots with names like helix and hxadm This option follows the case-sensitivity of the server.
`-E` ***nameFilter***	Makes the matching case-insensitive, even on a case-sensitive server.
`-t` ***type***	List only the depots of the specified *type*
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Examples

To list the depots of all types:

"Syntax" on page 19

```
p4 depots
```

The output might be similar to the following:

```
Depot archive 2017/12/23 archive archive/... 'Created by bruno. '
Depot depot 2018/05/27 local depot/... 'Default depot '
```

To list the depots of a specific type, use the `-t` option. For example:

```
p4 depots -t remote
p4 depots -t local
p4 depots -t stream
p4 depots -t spec
p4 depots -t archive
p4 depots -t graph
```

Related Commands

To create a remote depot or a new local depot	`p4 depot`
To remove all traces of a file from a depot	`p4 obliterate`

p4 describe

Provides information about changelists, as well as files in the changelists, and the path of the open stream, if a stream is open.

"Syntax" on page 19

```
p4 [g-opts] describe [-doptions] [-a -f -I -m -O -s -S]
changelist ...
```

Description

p4 describe displays the details of one or more changelists. For each changelist, the output includes the changelist number, the changelist creator, the client workspace name, the date the changelist was created, and the changelist description.

If a changelist has been **submitted**, the default output also includes a list of affected files and the diffs of those files relative to the previous revision. By default, this command does not perform "Keyword Expansion" on page 711 because keyword differences tend to obscure real differences.

If a changelist is **pending**, it is flagged as such in the output, and the list of open files is shown.

Diffs for **pending** changelists are not displayed because the files have yet to be submitted to the depot.

The **p4 describe** command limits its report depending on whether or not a changelist is public or restricted. Restricted **submitted** or **shelved** changes are not reported unless you either own the change or have **list** permission for at least one file in the change. Restricted **pending** (but unshelved) changes are visible only to the change owner. If you do not have permission to view a restricted changelist, the message "no permission" is displayed in place of a changelist description. Administrators can override this behavior and view restricted changelists by using the **-f** option.

You cannot run **p4 describe** on the default changelist.

The **p4 describe** command uses **p4**'s built-in diff subroutine. The **P4DIFF** variable has no effect on this command.

Options

-a	For text files only (ignores binary files): ▪ For shelved files, shows the content for "open for add" (pending) files. ▪ For submitted files, shows the content of added files.

`-d` *`options`*	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" below for an option listing.
`-f`	Force the display of descriptions for restricted changelists. This option requires `admin` permission.
`-I`	Specifies that the changelist number is the `Identity` field of a changelist.
`-m` *max*	Limits files to the first *max* number of files. The following example alphabetically lists (and diffs) two files affected by changelist 765 and two files affected by changelist 987: `p4 describe -m 2 765 987`
`-O`	If a changelist was renumbered on submit, and you know only the original changelist number, use `-O` and the original changelist number to describe the changelist.
`-s`	Display a shortened output that excludes the diffs of the files.
`-S`	Display the names of files shelved for the specified changelist, including the diff of each file against its previous depot revision.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`read`, `list` for `p4 describe -s`

The diff options supported by `p4 describe` are:

Option	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc [`*`num`*`]`	context output format, showing line number ranges and *num* lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du [`*`num`*`]`	unified output format, showing added and deleted lines with *num* lines of context, in a form compatible with the `patch(1)` utility.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs

Option	Name
`-db`	ignore changes made within whitespace; this option implies `-dl`.
`-dw`	ignore whitespace altogether; this option implies `-dl`.

Examples for files

These examples use `8` to represent a changelist number associated with text (non-binary) files.

Command	Pending Changelist displays	Submitted Changelist displays
`p4 describe 8`	changelist description	changelist description with a list of affected files and any diffs
`p4 describe -S 8`	■ changelist description of the shelved files associated with the specified changelist ■ diff between shelved and depot versions	
`p4 describe -Sa 8`	■ diffs of edited files that are shelved ■ content of new files that will be added	■ diffs of edited files that were shelved ■ content of files that were added
`p4 describe -a 8`	■ diffs of edited files ■ content of shelved files that will be added	■ diffs of edited files ■ content of files that were added

Examples for open stream

`p4 describe 2`

```
Change 2 by bruno@brn123 on 2019/02/04 09:12:23 pending
x
Affected stream: //root/main
Affected files ...
```

`p4 -ztag describe 2`

```
... change 2
... user bruno
... client brn123
... time 1546621943
... desc x
... status pending
... changeType public
... stream //root/main
```

Related Commands

To view a list of changelists	`p4 changes`
To view a list of all opened files	`p4 opened`
To compare any two depot file revisions	`p4 diff2`
To compare a changed file in the client to a depot file revision	`p4 diff`

p4 describe (graph)

Display a commit description.

"Syntax" on page 19

`p4 describe -n //`*`repo`*`/`*`name`* `[-a -d`*`options`* `-s]` *`sha`*

Description

Display information about the specified commit.

- Specify the repo name after `-n`
- sha is the corresponding sha1 of the commit

Options

-a	Display the content of added files in addition to diff of updated files.
`-d`*`options`*	Passes one or more options to the built-in diff routine to modify the output
`-s`	Display a shortened output that excludes the diffs of the files that were updated.

The diff options supported by **`p4 describe`** are:

Option	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc [`*`num`*`]`	context output format, showing line number ranges and *num* lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du [`*`num`*`]`	unified output format, showing added and deleted lines with *num* lines of context, in a form compatible with the `patch(1)` utility.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs
`-db`	ignore changes made within whitespace; this option implies `-dl`.
`-dw`	ignore whitespace altogether; this option implies `-dl`.

Examples

`p4 describe -n //repo/name SHA1`

where *`SHA1`* represents the commit `SHA1` that is created after the submit occurs. A changelist that is pending does not yet have a `SHA1`.

p4 diff

Diff utility for comparing workspace content to depot content. (For comparing two depot paths, see "p4 diff2" on page 168.) Also for stream spec comparison.

"Syntax" on page 19

```
p4 [g-opts] diff [-doptions] [-f -t -Od] [-m max] [-soptions]
[file[rev] ...]
```

```
p4 [g-opts] diff [-doptions] -As [streamname[@change]]
```

Description

`p4 diff` runs a diff program on your workstation that compares files in your workspace to revisions in the depot.

This command takes a file argument, which can contain a revision specifier.

- If a revision specifier is included, the file in the client workspace is diffed against the specified revision.
- If a revision specifier is not included, the client workspace file is compared against the revision currently being edited (usually the head revision).
- In either case, the client file must be open for `edit`, or the comparison must be against a revision other than the one to which the client file was last synced.

If the file argument includes wildcards, all open files that match the file pattern are diffed.

If no file argument is provided, all open files are diffed against their depot counterparts.

Tip
By default, your workstation runs the diff routine built into the `p4` command-line application. You can make your workstation run an external diff program by setting the `P4DIFF` environment variable to point to your external diff program.

To diff stream specs

`p4 [g-opts] diff [-doptions] -As [streamname[@change]]` allows the user to diff a privately edited stream spec against another version of the same stream spec. To use this option, the user's client workspace must be associated with the given stream, and the stream spec must be privately opened for edit.

Options for workspace content

`-d options`	Pass options to the underlying diff routine (see "Usage Notes" below for details).
`-f`	Force the diff (if no revision is specified, against the head revision), even when the client file is not open for `edit`.
`-m max`	Limit output to diffs (or status) of only the first *max* files, unless the `-s` option is used, in which case the `-m` option is ignored.
`-Od`	Limit output to only those files that differ.
`-s options`	Pass display options to the underlying diff routine (see "Usage Notes" below for details).
`-t`	Diff the revisions even if the files are not of type `text`.
g-opts	See "Global options" on page 690.

Options for stream specs

`-d options`	Pass options to the underlying diff routine (see "Usage Notes" below for details).
`-As`	Allows two arbitrary stream specs to be diffed against each other. Can be used with a streamname, or with a streamname at a specific changelist number. **Warning** Limitation: Although this option requires the user have at least the `list` access to the stream path, it ignores any other entry in the protections table, including any minus sign (`-`) that would otherwise block the operation.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	No	`read` or `list` to use the `-As` option

- The `-doptions` supported by `p4 diff` are:

Option	Name
`-db`	ignore changes made within whitespace; this option implies `-dl`.
`-dc [num]`	context output format, showing line number ranges and *num* lines of context around the changes.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs.
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du [num]`	unified output format, showing added and deleted lines with *num* lines of context, in a form compatible with the `patch(1)` utility.
`-dw`	ignore whitespace altogether; this option implies `-dl`.

- The `-soptions` supported by `p4 diff` are:

Option	Name
`-sa`	Show only the names of opened files that are different from the revision in the depot, or are missing.
`-sb`	Show only the names of files opened for integrate that have been resolved, but that have been modified after being resolved.
`-sd`	Show only the names of unopened files that are missing from the client workspace, but present in the depot.
`-se`	Show only the names of unopened files in the client workspace that are different than the revision in the depot.
`-sl file ...`	Every unopened *file* is compared with the depot, and listed with a status of `same`, `diff`, or `missing`. If you use the `-f` option together with the `-sl` option, files that are open for edit are also compared and their status is listed.
`-sr`	Show only the names of opened files in the client workspace that are identical to the revision in the depot.

- To pass more than one option to the diff routine, group them together. For example:

  ```
  p4 diff -dub file
  ```

  specifies a unified diff that ignores changes in whitespace.

"Syntax" on page 19

- The header line of a unified diff produced with the `-du` option for use with `patch(1)` displays filenames in Helix server syntax, not local syntax.
- If a revision is provided in the file specification, the `-s` options compare the file(s) regardless of whether they are opened in a changelist or the workspace has been synced to the specified revision.

Examples for workspace content

`p4 diff file#5`	Compare the client workspace revision of file `file` to the fifth depot revision.
`p4 diff @2017/05/22`	Compare all open files in the client workspace to the revisions in the depot as of midnight on May 22, 2017.
`p4 diff -m 10 @2017/05/22`	Limit to the first 10 files the comparison all open files in the client workspace to the revisions in the depot as of midnight on May 22, 2017.
`p4 diff -du file`	Run the comparison on file `file`, displaying output in a format suitable for the `patch(1)` utility.
`p4 diff -sr | p4 -x - revert`	Revert all open, unchanged files. This differs from `p4 revert` -a (revert all unchanged files, where resolving a file, even if no changes are made, counts as a change), in that it reverts files whose workspace content matches the depot content, including resolved files that happen to be identical to those in the depot. The first command shows all open, unchanged files. The second command (running `p4 -x` and taking arguments, one per line, from standard input, abbreviated as "-") reverts each file in that list. (This UNIX version of this command uses a pipe. Most operating systems have a similar way of performing these operations in series). For more information about the `-x` option to `p4`, see "Global options" on page 690.

Examples for stream specs

`p4 diff -As`
`p4 diff -As myStream`
`p4 diff -As myStream@have`
	diff between the opened stream spec and the have version of the stream spec

`p4 diff -As myStream@head`	diff between the opened stream spec and the head version of the stream spec
`p4 diff -As myStream@1`	diff between the opened stream spec and the version of the stream spec at change `1`
`p4 diff -As myStream@=1`	diff between the opened stream spec and the shelved version of the stream spec at change `1`

Related Commands

To compare two depot revisions	`p4 diff2`
To view the entire contents of a file	`p4 print`

p4 diff (graph)

Diff utility for comparing workspace content to repo content. (For comparing two repo paths, see "p4 diff2 (graph)" on page 172.)

"Syntax" on page 19

```
p4 diff [-doptions -soptions file ...]
```

Description

On the client machine, diff a client file against the corresponding revision in the repo. The file is compared only if the file is opened for edit. If the file specification is omitted, all open files are diffed.

Options

`-doptions`	Pass options to the underlying diff routine.

- The **`-doptions`** supported by **`p4 diff`** are:

Option	Name
`-db`	ignore changes made within whitespace; this option implies **`-dl`**.
`-dc [`*`num`*`]`	context output format, showing line number ranges and ***`num`*** lines of context around the changes.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs.
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du [`*`num`*`]`	unified output format, showing added and deleted lines with ***`num`*** lines of context, in a form compatible with the **`patch(1)`** utility.
`-dw`	ignore whitespace altogether; this option implies **`-dl`**.

- The **`-soptions`** supported by **`p4 diff`** are:

Option	Name
`-sa`	Show only the names of opened files that are different from the revision in the depot, or are missing.
`-sd`	Show only the names of unopened files that are missing from the client workspace, but present in the depot.
`-se`	Show only the names of unopened files in the client workspace that are different than the revision in the depot.
`-sl`	Every unopened ***`file`*** is compared with the depot, and listed with a status of **`same`**, **`diff`**, or **`missing`**.
`-sr`	Show only the names of opened files in the client workspace that are identical to the revision in the depot.

p4 diff2

Diff utility for comparing the content at two depot paths. (For comparing workspace content to depot content, see "p4 diff" on page 162.)

You can specify the source and the target files on the command line or through a branch view.

Also compares two arbitrary stream specs with the `-As` option.

"Syntax" on page 19

```
p4 [g-opts] diff2 [-doptions] [-Od -q -t -u] file1[rev] file2
[rev]
```

```
p4 [g-opts] diff2 [-doptions] [-Od -q -t -u] -b branch [[fromfile
[rev]]
                   tofile[rev]]
```

```
p4 [g-opts] diff2 [-doptions] [-Od -q -t -u] [-S stream] [-P
parent]
                   [[fromfile[rev]] tofile[rev]]
```

```
p4 [g-opts] diff2 [-doptions] -As streamname1[@change1]
streamname2[@change2]
```

Description for depot content

`p4 diff2` uses the Perforce service's built-in diff routine to compare two file revisions from the depot. These revisions are usually two versions of the same file, but they can be revisions of entirely separate files. If no file revision is explicitly provided with the file argument, the head revision is used.

- If you specify no arguments, the current stream is diffed against its parent stream.
- If you specify a filename, it diffs that file in the current stream against its the same file in the parent stream.

Tip
`p4 diff2` ignores the client environment variable `P4DIFF` because it runs on the Helix server.

You can specify file patterns as arguments in place of specific files, with or without revision specifiers. Helix server performs diffs for each pair of files that match the given pattern. If you invoke `p4 diff2` with file patterns, escape the file patterns from the OS shell by using quotes or backslashes.

Helix server presents the diffs in UNIX diff format, prepended with a header. The header is formatted as follows:

```
==== file1 (filetype1) - file2 (filetype2) ==== summary
```

The possible values and meanings of *summary* are:

- `content`: the file revisions' contents are different,
- `types`: the revisions' contents are identical, but the filetypes are different,
- `identical`: the revisions' contents and filetypes are identical.

If either *file1* or *file2* does not exist at the specified revision, the header displays the *summary* as `<none>`.

Options for depot content

`-b branch from [rev] to [rev]`	Use a branch mapping to diff files in two branched codelines. The files that are compared can be limited by file patterns in either the ***from*** or ***to*** file specifications.
`-doptions`	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" on the facing page for a listing of these options.
`-Od`	Limit output to only those files that differ.
`-q`	Quiet diff. Display only the header; if *file1* and *file2* are identical, display only "`file1 - no differing files`" as the output.
`-S stream [-P parent]`	Diff a stream with its parent. To diff the stream with a stream other than its configured parent, specify `-P`.
`-t`	Diff the file revisions even if the file(s) are not of type `text`.
`-u`	Generate unified output format, showing added and deleted lines with sufficient context for compatibility with the `patch(1)` utility. Only those files that differ are included. File names and dates remain in Helix server syntax.
g-opts	See "Global options" on page 690.

Description for stream specs

You can compare any two stream specs. For example, to diff between `myStream` at change `1` and `yourStream` at change `2`:

```
p4 diff2 -As myStream@1 yourStream@2
```

and to diff between shelved stream specs:

```
p4 diff2 -As myStream@=1 yourStream@=2
```

Options for stream specs

Allows the two specified stream specs to be diffed against each other.

`-d` *`options`*	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" below for a listing of these options.
`-As`	Allows two arbitrary stream specs to be diffed against each other. Can be used with a streamname, or with a streamname at a specific changelist number. **Warning** Limitation: Although this option requires the user have at least the `list` access to the stream path, it ignores any other entry in the protections table, including any minus sign (`-`) that would otherwise block the operation.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	No	`read` access necessary for both file revisions or `list` to use the `-As` option

- The diff options supported by `p4 diff2` are:

Option	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc [num]`	context output format, showing line number ranges and *num* lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du [num]`	unified output format, showing added and deleted lines with *num* lines of context, in a form compatible with the `patch(1)` utility.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs
`-db`	ignore changes made within whitespace; this option implies `-dl`.
`-dw`	ignore whitespace altogether; this option implies `-dl`.

- To pass more than one option to the diff routine, group them together. For example:

```
$ p4 diff2 -dub file1file2
```

  specifies a unified diff that ignores changes in whitespace.
- The header line of a unified diff produced with the `-du` option for `patch(1)` use displays the diffed files in Helix server syntax, not local syntax.
- When `p4 diff2` is used to diff `binary` files, the line

```
... files differ ...
```

  is printed if they are not identical.
- The option `-b branch[[fromfile[rev]]tofile[rev]]` allows you to specify a ***fromfile*** file pattern and a ***tofile*** revision, or a ***fromfile*** revision and a ***tofile*** file pattern.
- RCS keywords within files are not expanded with `p4 diff2`.

Examples

`p4 diff2 -ds file#1 file`	Compare the first revision of file `file` to its head revision, and display a summary of what chunks were added to, deleted from, or changed within the file.

`p4 diff2 file@34 file@1998/12/04`	Diff the revision of `file` that was in the depot after changelist 34 was submitted against the revision in the depot at midnight on December 4, 1998.
`p4 diff2 //depot/rel1/... //depot/rel2/...#4`	Compare the head revisions of all files under `//depot/rel1` to the fourth revision of all files under `//depot/rel2`.
`p4 diff2 //depot/rel1/* //depot/rel2/...`	Not allowed. The wildcards in each file pattern must match.
`p4 diff2 -b branch2 //depot/rel2/...#2 @50`	Compare the second revision of the files in `//depot/rel2/...` to the files branched from it by branch mapping `branch2` at the revision they were at in changelist 50.

Related Commands

To compare a client workspace file to a depot file revision	`p4 diff`
To view the entire contents of a file	`p4 print`

p4 diff2 (graph)

Diff utility for comparing the content at two repo paths. (For comparing workspace content to repo content, see "p4 diff (graph)" on page 166.)

"Syntax" on page 19

```
p4 diff2 [options] fromFile[@sha1|@reference] tofile
[@sha1|@reference]
```

Description

`p4 diff2` runs on the server to compare one set of graph repo files (the 'source') to another (the 'target'). Source and target file sets can be specified on the 'p4 diff2' command line.

`fromFile` and `toFile` can include @sha1 or @reference specifiers. By default, the head revisions are diffed.

This command precedes each diffed file pair with a header line of the following form:

`==== source#rev (type) - target#rev (type) ==== summary`

A source or target file shown as `<none>` means there is no file at the specified name or reference to pair with its counterpart. The summary status is one of the following: `identical` means file contents and types are identical, `types` means file contents are identical but the types are different, and `content` means file contents are different.

Options

`-d` *`options`*	Runs the diff routine with one of a subset of the standard UNIX diff options.
`-Od`	Limit output to only those files that differ.
`-q`	Quiet diff. Display only the header; if *file1* and *file2* are identical, display only "*`file1`* `- no differing files`" as the output.
`-u`	Generate unified output format, showing added and deleted lines with sufficient context for compatibility with the `patch(1)` utility. Only those files that differ are included. File names and dates remain in Helix server syntax.

- The diff options supported by `p4 diff2` are:

Option	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc [`*`num`*`]`	context output format, showing line number ranges and *num* lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du [`*`num`*`]`	unified output format, showing added and deleted lines with *num* lines of context, in a form compatible with the `patch(1)` utility.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs
`-db`	ignore changes made within whitespace; this option implies `-dl`.
`-dw`	ignore whitespace altogether; this option implies `-dl`.

- To pass more than one option to the diff routine, group them together. For example:

```
$ p4 diff2 -dub file1 file2
```

specifies a unified diff that ignores changes in whitespace.

- The header line of a unified diff produced with the `-du` option for `patch(1)` use displays the diffed files in Helix server syntax, not local syntax.
- When **`p4 diff2`** is used to diff `binary` files, the line

  ```
  ... files differ ...
  ```

  is printed if they are not identical.
- The option `-b branch[[fromfile[rev]]tofile[rev]]` allows you to specify a ***fromfile*** file pattern and a ***tofile*** revision, or a ***fromfile*** revision and a ***tofile*** file pattern.
- RCS keywords within files are not expanded with **`p4 diff2`**.

Examples

```
p4 diff2 //repo/main/src/...@00662f4
//repo/main/src/...@refs/heads/bugfix
```

where `@00662f4` represents the commit SHA-1 and `refs/heads/bugfix` represents the branch. The output might show differences between the files currently in the branch and files in the commit:

```
==== //repo/main/src/chat.c#1e7637e (text) -
//repo/main/src/chat.c#1e7637e (text) ==== identical
==== //repo/main/src/db.c#6950848 (text) - //repo/main/src/db.c#2ab62af
(text) ==== content
2,3d1
< Additional database code.
< Add Btree code
==== //repo/main/src/main.c#184e90a (text) -
//repo/main/src/main.c#5a8f6ff (text) ==== content
3d2
< Enable additional database code.
```

where *#number* represents the blob SHA-1 of a file associated with the commit SHA-1.

p4 dirs

List the immediate subdirectories of the specified depot directories.

"Syntax" on page 19

```
p4 [g-opts] dirs [-C -D -H] [-S stream] [-i] depot_directories*
[revSpec]
```

Description

Use `p4 dirs` to find the immediate subdirectories of any depot directories provided as arguments.

The depot_directories argument must be provided in depot or local syntax and must end with the `*` wildcard.

`p4 dirs` only lists the immediate subdirectories of the directory arguments.

By default, only subdirectories that contain at least one undeleted file will be returned. To include those subdirectories that contain only deleted files, use the `-D` option.

If you include a revision specifier or revision range as part of a directory argument, the only subdirectories returned are those that contain at least one file revision that matches the given specifier.

Note
This command is meant to be used in scripts rather than from the command line.

Options

`-C`	Display only those directories that are mapped through the current client view.
`-D`	Include subdirectories that contain only deleted files. By default, these directories are not displayed.
`-H`	Include only those directories that contain files on the current client workspace's `p4 have` list.
`-S stream`	List directories mapped for the specified stream.
`-i`	Ignore the case of the directory argument when listing directories in a case-sensitive server.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

Perforce does not track directories in its database. Therefore, the "`...`" wildcard is not supported.

Examples

`p4 dirs //depot/projects/*`	Returns a list of all the immediate subdirectories of `//depot/projects`, such as `//depot/projects/test` and `//depot/projects/prod`
`p4 dirs //depot/projects/*/*`	Returns a list of all the subdirectories two levels below `//depot/projects`, such as `//depot/projects/test/t1` and `//depot/projects/prod/p1`
`p4 dirs //depot/a/* //depot/b/*`	Returns a list of all immediate subdirectories of `//depot/a` and `//depot/b`.
`p4 dirs -i //depot/read*`	Case-insensitive version retrieves //depot/README.TXT, //depot/ReadMe.html, and so on.

Related Commands

To list all the files that meet particular criteria	`p4 files`
To list all depots known to the Helix server	`p4 depots`

p4 dirs (graph)

List repo subdirectories.

"Syntax" on page 19

```
p4 dirs [-i] dir[@reference] ...
```

Description

List directories that match the specified file pattern (`dir`). This command supports wildcards (`'...'`, `'*'`).

By default, all directories containing files are listed. If the `dir` argument includes a reference, only directories containing files at that reference are listed.

Options

`-C`	Display only those directories that are mapped through the current client view.
`-i`	Ignore the case of the directory argument when listing directories in a case-sensitive server. This flag is not compatible with the -C option.

Examples

`p4 dirs //bruno_1666/*/*/*`	The output might be similar to:
	`//depot/projectA/src`
	`//graphDepot1/projectB/src`
	`//graphDepot1/projectB/test`
	See the section "Including Graph Depot repos in your client" on page 113 in the topic "p4 dirs (graph)" on the previous page.

`p4 dirs -i //bruno_1666/*/*/*`	The output might be similar to:
	`//graphDepot1/projectB/test`
	`//Graphdepot1/ProjectB/TEST`
	See the section "Including Graph Depot repos in your client" on page 113 in the topic "p4 dirs (graph)" on the previous page.

p4 diskspace

Display disk space information on the server.

p4 df is an alias for **p4 diskspace**.

"Syntax" on page 19

```
p4 [g-opts] diskspace [P4ROOT | P4JOURNAL | P4LOG | TEMP |
journalPrefix | depot]
```

Description

Shows summary information about the current availability of disk space on the server.

The output of **p4 diskspace** is in the form:

```
name  (type type ) : xxx  GB free, yyy  GB used, zzz  GB total (ff  %
full)
```

Where *name* can be either **P4ROOT**, **P4JOURNAL**, **P4LOG**, **TEMP**, a prefix to a non-default Helix server journal file location, or the name of a Helix server depot. The filesystem *type* is that reported by the operating system.

If no arguments are specified, disk space information is displayed for all objects.

By default, Helix server rejects commands when free space on the filesystems housing the **P4ROOT**, **P4JOURNAL**, **P4LOG**, or **TEMP** falls below 10 megabytes. To change this behavior, set the **filesys.P4ROOT.min** (and corresponding) configurables to your desired limits.

If the user account that runs the Helix server is subject to disk quotas, the **filesys.*.min** configurables reflect those quotas, regardless of how much physical space actually remains on the filesystem(s) in question.

Note
Server releases prior to 16.1 included reserved space in the number displayed for used space in the Size column for Unix filesystems. This has been changed so that the size given now excludes reserved space.

Options

depot	Report disk space available for filesystem holding the specified *depot*.

journalPrefix	Report disk space available for filesystem holding a non-standard journal location.
`P4JOURNAL`	Report disk space available for filesystem holding `P4JOURNAL`.
`P4LOG`	Report disk space available for filesystem holding `P4LOG` (server log).
`P4ROOT`	Report disk space available for filesystem holding `P4ROOT`.
`TEMP`	Report disk space available for filesystem holding temporary files. If not defined, uses `P4ROOT` on Windows, and `/tmp` on Unix.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Related Commands

To configure Perforce's behavior when diskspace is low	`p4 configure`

p4 edit

Opens files in a client workspace for edit, or open the current stream spec.

"Syntax" on page 19

```
p4 [g-opts] edit [-c changelist] [-k -n] [-t type] [--
remote=remote] file...
```

```
p4 [g-opts] edit -So  [-c changelist]
```

Description

Files and p4 edit

`p4 edit` opens files for editing within the client workspace. The specified files are linked to a changelist, but the files are not actually changed in the depot until the changelist is committed with **`p4 submit`**.

Helix server controls the local OS file permissions. When **`p4 edit`** is run, the OS **`write`** permission is turned on for the specified files.

When a file that has been opened for edit with **`p4 edit`** is submitted to the depot, the file revision that exists in the depot is not replaced. Instead, the new file revision is assigned the next revision number in sequence, and previous revisions are still accessible. By default, the newest revision (the *head revision*) is used by all commands that refer to the file.

By default, the specified files are added to the default changelist. Use **`-c`** to specify a different changelist. (Or use the **`p4 change`** command to move files from the default changelist to a numbered changelist.)

To move files already opened for edit from one changelist to another, use **`p4 reopen`**.

Streams and p4 edit

The **`-So`** option opens the stream only, so no list of files is allowed.

Options

`-c` ***`changelist`***	Opens the files for edit within the specified changelist. If this option is not provided, the files are linked to the default changelist.

`-k`	Keep existing workspace files; mark the file as open for edit even if the file is not in the client view. Use **`p4 edit -k`** only in the context of reconciling work performed while disconnected from the shared versioning service.
`-n`	Preview which files would be opened for edit, without actually changing any files or metadata.
`--remote=`*`remote`*	Opens the file for edit in your personal server, and additionally — if the file is of type `+l` — takes a global exclusive lock on the file in the shared server from which you cloned the file. For more information, see the section Support for exclusive locking in the Fetching and Pushing chapter of *Using Helix Core Server for Distributed Versioning*.
`-t` *`type`*	Stores the new file revision as the specified type, overriding the file type of the previous revision of the same file. To forcibly re-detect a file's filetype (that is, to assign a file type as if the file were being newly added) upon editing a file, use **`p4 edit -t auto`**. See "File types" on page 707 for a list of file types.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

Because **`p4 edit`** turns local OS **`write`** permissions on for the specified files, this command should be given before the file is actually edited. The process is:

1. Use **`p4 edit`** to open the file in the client workspace,
2. Edit the file with any editor,
3. Submit the file to the depot with **`p4 submit`**.

To edit an older revision of a file, use **`p4 sync`** to retrieve the previously stored file revision into the client workspace, and then **`p4 edit`** the file. Because this file revision is not the head revision, you must use **`p4 resolve`** before the file can be stored in the depot with **`p4 submit`**.

By default, Helix server does not prevent users from opening files that are already open; its default scheme is to allow multiple users to edit the file simultaneously, and then resolve file conflicts with **`p4 resolve`**. To determine whether or not another user already has a particular file opened, use **`p4 opened-a`** *`file`*.

If you need to prevent other users from working on files you've already opened, you can either use the `p4 lock` command (to allow other users to edit files you have open, but prevent them from submitting the files until you first submit your changes), or you can use the `+l` (exclusive-open) filetype to prevent other users from opening the files for edit at all.

In older versions of Helix server, `p4 edit` was called `p4 open`.

Examples for files

`p4 edit -t text+k doc/*.txt`	Opens all files ending in `.txt` within the current directory's `doc` subdirectory for `edit`. These files are linked to the default changelist; these files are stored as type `text` with keyword expansion.
`p4 edit -t +l //depotname/...`	Implements pessimistic locking (exclusive-open) for all files in a depot. After this changelist is submitted, only one user at a time will be able to edit files in the depot named ***`depotname`***.
`p4 edit -c 14 ...`	Opens all files anywhere within the current working directory's file tree for `edit`. These files are examined to determine whether they are `text` or `binary`, and changes to these files are linked to changelist 14.
`p4 edit status%40jan1.txt`	Open a file named `status@jan1.txt` for edit. For details about how to specify other characters reserved for use as Helix server wildcards, see "Limitations on characters in filenames and entities" on page 699.

Examples for a stream

`p4 edit -So`	Opens the current stream spec to the default changelist.
`p4 edit -So -c 14`	Opens the current stream spec to the specified numbered change list

Related Commands

To open a file for add	`p4 add`
To open a file for deletion	`p4 delete`
To copy all open files to the depot	`p4 submit`
To copy files from the depot into the client workspace	`p4 sync`
To create or edit a new changelist	`p4 change`

To list all opened files	`p4 opened`
To revert a file to its unopened state	`p4 revert`
To move an open file to a different changelist or change its filetype	`p4 reopen`

p4 edit (graph)

Open an existing file for edit.

"Syntax" on page 19

```
p4 edit [-c changelist# -n] file ...
```

Description

Open an existing file for edit. The server records that the current user has opened the file in the current workspace, and changes the file permission from read-only to read/write.

Options

`-c` ***changelist***	Opens the files for edit within the specified changelist. If this option is not provided, the files are linked to the default changelist.
`-n`	Preview which files would be opened for edit, without actually changing any files or metadata.

p4 export

Extract journal or checkpoint records.

"Syntax" on page 19

```
p4 export -c token [-J prefix] [-f] [-l lines] [-F filter]
            [-T tableexcludelist] [-P filterpattern]
```

```
p4 export -j token [-J prefix] [-f] [-l lines] [-F filter]
           [-T tableexcludelist] [-P filterpattern]
```

```
p4 export -j token [-J prefix] -r [-F filter]
           [-T tableexcludelist] [-P filterpattern]
```

Description

This command reports checkpoint and journal metadata from a Helix server server. With no options, the records are reported in tagged form.

Some fields are added to the tagged output to indicate either transactional consistency, or to indicate the end of the journal.

To filter database tables out of the exported data, use the `-T` option with a list of tables whose data you wish to exclude. To exclude data from multiple tables, separate the table names by spaces or commas. The table names must begin with "`db.`", following the naming convention used for database files in the server root directory. If you separate the table exclusion list with spaces, you must enclose the list in quotes.

Options

`-c`	Specifies a checkpoint number or position token of the form ***`checkpointnum#byteoffset`***.
`-f`	Format the output so that non-textual datatypes are formatted appropriately.
`-F` ***`filter`***	Limit output to records that match the specified ***`filter`*** pattern. For example, `-F "table = db.configure"`.

`-j`	Specify a journal number or position token of the form ***`journalnum/byteoffset`***.
`-J` ***`prefix`***	Specifies a filename prefix for the journal, such as that used with `p4d -jc` ***`prefix`***.
`-l` ***`lines`***	Limit output to the specified number of ***`lines`*** of journal records.
`-P` ***`filterpattern`***	Limit output to records that match the specified filter pattern. Multiple filter patterns can be specified with multiple `-P` options. Each ***`filterpattern`*** begins with two characters and a colon, and specifies either a client filter or a depot filter, as well as whether the pattern is to be included or excluded, using the syntax: ▪ `-Pic://client/`***`pattern`*** - client records to include ▪ `-Pxc://client/`***`pattern`*** - client records to exclude ▪ `-Pif://depot/`***`pattern`*** - depot records to include ▪ `-Pxf://depot/`***`pattern`*** - depot records to exclude The first character specifies whether the records are included or excluded ("`i`" or "`x`"), the second character specifies whether the records are client workspace-related or depot-file related ("`c`" or "`f`"), the colon is a separator, and the remainder of the ***`filterpattern`*** denotes either a client workspace view or a depot file path. The mechanism by which this filtering is implemented is the same as that which is used by the `ClientDataFilter:` and `RevisionDataFilter:` fields in the `p4 server` form.
`-r`	Display raw journal output; this argument applies to journals only.
`-T` ***`tableexcludelist`***	Supply a list of database tables (for example, `db.have` and `db.client`) to exclude from export. Limit output to records that match the specified ***`filter`*** pattern. For example, `-T db.have,db.client` or `-T "db.have db.client"`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Compressed journals or checkpoints are not supported.

Examples

`p4 export -T "db.have db.working"`	Run `p4 export`, but ignore records in the `db.have` and `db.working` tables.

Related Commands

To replicate metadata from one server to another	`p4 replicate`
To pull journal records (and file content) from a master server to a replica server	`p4 pull`

p4 extension

Manage the Helix Core Server extensibility mechanism.

"Syntax" on page 19

```
p4 extension --sample extName
```

```
p4 extension --package dir
```

```
p4 extension --install extName.p4-extension [--yes]
```

```
p4 extension --delete extFQN [ --name instName ] [ --revision=n ]
             [ --path filespec ] [--yes]
```

```
p4 extension --configure extFQN [ --revision=n ] [ -o | -i ]
```

```
p4 extension --configure extFQN [ --revision=n ] [ -o | -i
             --name instName
```

```
p4 extension --run instName [ extArguments ]
```

```
p4 extension --list --type type
```

Description

`p4 extension` manages the installation, versioning, and configuration of Helix Core Server Extensions.

Extensions provide a way to customize the behavior of the Helix Core Server with user-supplied logic. Extensions are self-contained packages of third-party code and assets run within the Helix Core Server. Extensions can be used for many use cases, such as change submission validation, form validation, external authentication, external job fix integration, external archive integration, and command policies. For a list of events that Extensions can register for, see the list of trigger types in the output of `p4 help triggers`.

Extensions coexist with triggers (see "p4 triggers" on page 588), but offer more functionality. The Helix Core Server runs Extensions natively, without relying on external processes, so Extensions provide a portable, versioned runtime with automatic replication and a programmatic API.

Extensions are versioned within the Helix Core Server and are stored in a special depot, named `.p4-extensions` by default. The extension depot is only accessible to the super user and by read-only commands. The extension depot is created automatically.

Installing or upgrading an Extension creates a changelist with information about pre- and post-install versions.

An Extension can implement custom commands that a user can run. For example,

```
p4 extension --run myInstanceConfig validateFileSize pathToFile
```

where ***`validateFileSize`*** represents a custom command and ***`pathToFile`*** represents an argument that is relevant to the custom command.

Configuration and access

After installation, Extensions must be configured before use.

Firstly, the super user runs `p4 extension --configure extName` to supply various global details about the Extension's configuration, such as the list of groups whose members can create instances of the Extension, or Extension runtime limits.

Secondly, the super user uses the `--configure` and `--name` options together to create a named instance of the Extension, parameterizing the Extension to be run with specific settings. The `--name` option takes the name of the configuration to create or modify and the `--configure` option takes the name of the Extension.

> **Note**
> For an extension to be active, it must have a global configuration and at least one instance configuration. See the `--configure` option. An Extension can have multiple configurations, depending on the events it registers for. Each configuration of an Extension can point to a different version of the Extension.

"Syntax" on page 19

Naming rules

Extensions are referred to by their Fully-Qualified Name (***`extFQN`***). This is the combination of the Extension's namespace, name, and optional depot file revision. For example,

- a namespace of **`ExampleInc`** and name of **`extName`** would be referred to as **`ExampleInc::extName`**
- with a revision of **`6`**, the combination would be **`ExampleInc::extName#6`**

Extensions are referred to by their fully-qualified name. This is the combination of the namespace, name, and, optionally, the revision of the Extension that is installed on a server (as opposed to the version of the extension code). For example, a namespace of **`ExampleInc`** and a name of **`ExtName`** would be referred to as **`ExampleInc::ExtName`**

Note
For more information about Extensions, see **`p4 help serverextensionintro`** and *Helix Core Extensions Developer Guide*.

Options

`--sample`	Creates a skeleton of a new Extension. The argument is the name of the Extension to create as well as the name of the local directory where the files are placed. The sample Extension has placeholder values that should be replaced when developing the code. (See "Naming rules" above.)
`--package`	Creates a packaged Extension file from the named directory of files, preparing it for **`--install`** on the server. The resulting file has the same name as the directory, but with the **`.p4-extension`** suffix. Packaging is done client-side and requires a version 2020.1 or later command-line client.
`--install`	The **`--install`** flag is passed the full name of a packaged Extension file residing on the client and installs it server-side, where it can then be instantiated with **`--configure`**.
`--name`	The name of the instance configuration to create or modify.
`--revision`	Which depot version of the Extension to configure. The default is the head revision. This option applies to the **`--configure`** option. This flag applies to the **`--configure`** option when used with or without the **`--name`** option.

"Syntax" on page 19

`--configure`	The `--configure` flag is used to supply details about an Extension's configuration. There are two parts to configuring an Extension: ▪ the global configuration ▪ the instance configuration. To create a **global** configuration, use `--configure` with the ***`FQN`*** of the Extension. To create an **instance** configuration, use the `--configure` option with the `--name` option. The `--name` option takes the name of the configuration to create or modify. More than one named instance is allowed for most event types that an Extension can be coded to work with, such as the `change-submit` event. However, the `auth-check` event allows only one named instance.
`--list`	Displays information about installed Extensions. The value of the `--type` flag determines which data is reported. Values for `--type` are: ▪ `extensions` ▪ `configs`, further narrowed by `global` and `instance` Adding the `--path` option filters instance configuration output to instances whose events are path-based and match the specified filespec.

`--delete`	Deletes an extension, its configurations, or both.
	■ To remove the named instance config, include the `--name` option. For example:
	`p4 extension --delete myCompany::fileSizeCheck --name=fileSizeCheck-instance --yes`
	■ To remove *all* revisions of an extension and its configurations, omit the `--name` option. For example:
	`p4 extension --delete myCompany::fileSizeCheck`
	■ To remove one specific revision and its instances, include the `--revision` option. For example:
	`p4 extension --delete myCompany::fileSizeCheck --name=fileSizeCheck-instance --revision=2 --yes`
	■ To delete file-based instances registered under the specified path, use the `--path` option. For example:
	`p4 extension --delete ExampleInc::extD1 --path=//depot/... --yes`

`--run`	Executes the Extension-supplied command. An Extension can implement its own end-user commands that behave similarly to native Helix Core commands in that they can have tagged output, send RPC-level messages, and open forms client-side. However, Extension-supplied commands are restricted to reporting output and cannot access client-side content. The `--run` option takes an instance configuration as its argument, and any other arguments are left to the Extension to interpret.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

p4 failover

Fail over to a standby server.

"Syntax" on page 19

```
p4 failover [-y] [ -m | [-i] -s serverID ] [-w waitForQuiesce][-v
verificationTime][failoverMessage]
```

Description

Allows the administrator of a standby server to initiate the process of failing over from the current master server to that standby server. In this context, the master server is the server from which the standby server makes its journalcopy (see "p4 journalcopy" on page 294). The "master" server can also be an edge server. The "standby" server can be either a "standby" or "forwarding-standby" server.

Failing over to a "mandatory" (see "p4 server" on page 493 `Options:`) standby when the master server is not part of the failover process ensures that none of the downstream replicas will be ahead of the new master server.

- For the local High Availability standby server, we recommend **`mandatory`**, but only after journalcopying is complete.
- For the remote Disaster Recovery standby server, we recommend **`nomandatory`**

For details, see the Failover topic in the Backup and recovery chapter of Helix Core Server Administrator Guide.

Options

`-y`	Perform the failover operation. Without this option, **`p4 failover`** merely reports what it would do. **Important** Carefully consider the report of what failover would do BEFORE you include the **`-y`** option and launch the operation. During the failover process, end-user clients cannot issue commands against the server.
`-v`	Specifies the *verificationTime* as a number of seconds prior to the launch of the failover command. The default value is `300` seconds means that any file content that was updated from 1 - 300 seconds prior to launch of failover will be verified. This is to ensure that this file content will be correct when the failover process completes on the new master server. The failover command proceeds as soon as it has determined that the new master server has received correct file content. The legal range is between `0` (no verification) and `43200` seconds (12 hours).

`-w`	The waiting period in seconds for commands in the master server to complete. At the end of this *quiesceWait* period, all user commands, regardless of their activity, will be stalled. If `-w` is not specified, the default *quiesceWait* value is `60` You can set the waiting period to any integer value between `0` (immediate) and `3600` seconds (1 hour). To cancel the failover during the `quiesceWait` period, press `Ctrl-C`
`-i`	Ignore the original master server, even if it is accessible. This option and the `-m` option are mutually exclusive. Using the `-i` flag when the master server is accessible can result in an undesirable "split-brain" scenario in which two master servers process divergent datasets.
`-m`	The master server is required to participate in failover, which excludes the `-i` option. If the master server cannot be accessed by the standby server, the failover will not occur.
`-s`	If the existing master server participates in failover, the new master automatically gets the `serverID` of the old master and this option is not required. However, if the standby server cannot access the master server, use this option to specify the `serverID` of the master server.

failoverMessage	If the master server participates in the failover operation, *failoverMessage* is the text to display to end-users during the failover operation when end-users attempt to start new commands on the master server. The default message is:
	`Server currently in failover mode, try again after failover has completed`
	The *failoverMessage* must be the final argument specified in the `p4 failover` command. For example,
	`p4 failover -w 90 -v 500 -y The Perforce server is undergoing scheduled maintenance`
	and quotation marks are not required.

g-opts	See "Global options" on page 690

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Related commands

To copy journal data from a master server to the local file system of a standby replica.	`"p4 journalcopy" on page 294`
To copy the file content of the files that have been added or updated (according to the replicated journal data).	**`p4 pull -u`**
To retrieve the journal records from the journalcopy files created by the **`p4 journalcopy`**, and to apply these to the standby's database	**`p4 pull -L`**

p4 fetch

Copy files from a remote server into your local server.

> **Note**
> For distributed version control. See *Using Helix Core Server for Distributed Versioning* (DVCS).

"Syntax" on page 19

```
p4 [g-opts] fetch [-r remotespec] [-m depth] [-v -k] [-n | -t] [-
Ox]
                    [-S stream | filespec]
p4 [g-opts] fetch [-r remotespec] [-v] [-n] [-Ox] -s shelf
```

Description

The **`p4 fetch`** command copies the following items from the specified remote server to the local server:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes associated with the changelists, but only if the job that is linked by the fix is already present in the local server. If it is not, then the fix is not copied.
- all integration records that describe integrations to the files being fetched

A fetch is only allowed if the files being fetched fit cleanly into the server to which you're currently connected, building cleanly on a shared common history.

The second form of the command copies a shelved changelist, rather than one or more submitted changelists, in which case conflicts do not arise; the result is a new shelved change in the local server.

If there are no conflicts, the files and their changelists become new submitted changelists in the local server. Conflict handling is configurable, using the **`-t`** option. If **`-t`** is not specified, and there are any conflicts or gaps, the fetch is rejected. The **`-t`** option specifies that the conflicting changelists should be relocated to the tangent depot, and the remote work is then fetched. After the fetch completes, use **`p4 resubmit`** to resubmit the conflicting local changes.

When the changelists are added to the local server, they are given newly assigned change numbers but they retain the same description, user, date, type, workspace, and set of files. When the files are added to the local server, they are kept in their same changelists, as new revisions starting after the current head. The new revisions retain the same revision number, file type, action, date, timestamp, digest, and file size. Although the changelists are new submitted changelists in the local server, none of the submit triggers are run in the local server.

Note that once a particular revision has been copied to a local server, using `p4 attribute -f` to change the attributes on that revision will only affect the revision on that server, not on any other server to which it may have been copied.

Typically, the `p4 fetch` command specifies a remote spec, and the `DepotMap` field in the remote spec specifies which files are to be fetched. The `p4 fetch` command can also specify a filespec argument to further restrict the files to be fetched. The filespec argument can be one of the following:

- the name of a stream, such as `-S dev`
- a filename pattern, such as `//stream/dev/...`, `//path1/...`, or `//...`

You cannot not specify both a stream and a filename pattern in a single fetch command.

Note
If you use a filespec argument to restrict the files to be fetched, the **LastFetch:** field will not be updated until you issue **p4 fetch** without a filespec argument .

If the remote spec uses differing patterns for the local and remote sides of the `DepotMap`, the filespec argument, if provided, must specify the files using the local filename syntax. If a particular changelist includes some files that match the filespec, and other files that do not, only the matching files are included in the fetch. To ensure that a partial changelist is not fetched, an appropriate filespec should be specified (for example, `//...@change,#head`).

`p4 fetch` behaves differently if the remote spec's `ArchiveLimits:` field is set. This field regulates how many, if any, revisions of file archives are stored on the server you fetch to. For more information, see the section "Configure server to limit storage of archive revisions" in the "Fetching and Pushing" chapter of *Using Helix Core Server for Distributed Versioning*.

When `p4 fetch` copies integration records, they are adjusted in the local server to reflect the resulting changelist numbers and revision numbers of the local server. In order to fetch a set of files, you must have read access to those files in the remote server, and you must have write access to those same files in the local server; your local userid is used as the userid at the remote server and you must already be logged in to both servers prior to running the `p4 fetch` command.

By default, a server does not accept fetch requests from another server. In order to fetch from a server, an administrator of that server must enable fetching by setting "server.allowfetch" on page 806 to `1`.

The `p4 fetch` command is atomic: either all the specified files are fetched, or none of them are fetched.

Files with the filetype modifiers `+k`, `+l`, or `+S` have some special considerations. Files of type `+k` have their digests cleared when fetched. This means certain cross-server merge conflicts are not detected. To re-generate the digests after the fetch, use the `p4 verify` command. When fetching files of type `+l`, the new files are added to the server even if the files are currently open by a pending changelist in the server. When fetching files of type `+S`, old archives which exceed the specified limit are not purged by the fetch command.

The value of the `"rpl.checksum.change" on page 794` configurable determines the level of verification performed for the `p4 fetch` command. See "Configurables" on page 715.

Note
`p4 fetch` automatically performs a `p4 sync` as part of its operations.

Triggering on fetches

The following push trigger types may be invoked during the execution of the `p4 fetch` command:

- The `push-submit` trigger can customize processing during the phase of the `p4 fetch` command when metadata has been transferred but files have not yet been transferred.
- The `push-content` trigger can customize processing during that phase of the `p4 fetch` command when files have been transferred but their contents have not yet been committed.
- The `push-commit` trigger can do any clean up work or other post processing after changes have been committed by the `p4 fetch` command.

For more information, see the section "Triggering on pushes and fetches" in the scripting chapter of *Helix Core Server Administrator Guide*.

Options

With no options specified `p4 fetch` fetches files from the remote server named origin.

`-k`	Suppresses automatic sync of workspace to the head revision.
`-m` *`depth`*	Specifies that Helix server should perform a shallow fetch; only the last number of revisions specified in *depth* are fetched.
`-n`	Performs correctness checks but does not fetch any files or changelists from the remote server. In particular, Helix server checks for conflicts between work that's been done in the local server and work you are trying to fetch from the remote server. This tells you whether your personal server is up to date with the remote server.
`-Oc`	When set, the `p4 fetch` command outputs information about every changelist. The `-v` option must be set for this to take effect.

`-Of`	When set, the `p4 fetch` command outputs information about every file in every changelist. The `-v` option must be set for this to take effect.
`-Oi`	When set, the `p4 fetch` command outputs information about every integration in every file in every changelist. The `-v` option must be set for this to take effect.
`-r` *`remotespec`*	Specifies a remote spec containing the address of the remote server, and a file mapping which is to be used to re-map the files when they are fetched from the remote server. See also `p4 remote`.
`-s`	Specifies a shelved changelist to be fetched, instead of one or more submitted changelists.For more information, see the section "Fetch and push a shelved changelist" in the "Fetching and Pushing" chapter of *Using Helix Core Server for Distributed Versioning*.
`-t`	Specifies that conflicting changelists should be relocated to the tangent depot, automatically creating that depot if it does not exist. The relocated changes can then be resubmitted using `p4 resubmit`. If you don't specify a remote server with the `-r` option, the remote server defaults to `origin`. **Note** `p4 fetch -t` requires `admin` permission.
`-S` *`stream`*	Specifies a particular stream to fetch. If you specify a stream you cannot also specify a file or files.
`-v`	Enables verbose mode, which provides diagnostics for debugging. With verbose mode enabled, you can refine and control the precise level of verbosity. Specifically, you can indicate whether you want information about: ■ every changelist fetched (with the `-Oc` option) ■ every file in every changelist fetched (with the `-Of` option) ■ every integration of every file in every changelist fetched (with the `-Oi` option) You can specify any combination of these options, but must always include the `-O`. The default is to display information about every changelist.
`filespec`	Specifies which files to fetch. If you specify a file or files you cannot specify a stream with the `-S` option.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`read` on the remote server, `write` on the local server.

Examples

`p4 fetch -m 5 -r dev`	Fetch the most recent `5` revisions of each file in the `dev` remote spec.

Related Commands

To push to a remote server	`p4 push`

p4 filelog

Print detailed information about files' revisions.

"Syntax" on page 19

```
p4 [g-opts] filelog [-c change] [-h -i -l -L -t -p -s] [-m max]
FileSpec[revSpec]
```

Description

`p4 filelog` describes each revision of the files provided as arguments. At least one file or file pattern must be provided as an argument. If the file specification includes a revision range, only the specified revisions are listed.

By default, the output consists of one line per revision in reverse chronological order. The format of each line is:

```
... #rev change chnum action on date by user@client (type)
'description'
```

where:

- ***rev*** is the revision number
- ***chnum*** is the number of the submitting changelist
- ***action*** is the operation the file was open for. See the Description of `p4 integrated`.
- ***date*** is the submission date (by default), or date and time (if the `-t` option is used)
- ***user*** is the name of the user who submitted the revision
- ***client*** is the name of the client workspace from which the revision was submitted
- ***type*** is the type of the file at the given revision
- ***description*** is the first 30 characters of the corresponding changelist's description

  If the `-l` option is used, the ***description*** is the full changelist description as entered when the changelist was submitted. If the `-L` option is used, the description is the full changelist description, truncated to 250 characters.

By default, this command is optimized not to display history of a file which is deleted in the most recent commit, or has been removed previously.

To view the full history of a deleted file, specify the `-d` flag.

This option cannot be used with paths containing wildcards.

The `-m max` option displays at most `max` commits.

The `--first-parent` option follows only the first parent of each commit.

Options

`-c` *`change`*	Display only files submitted at the specified changelist number.
`-h`	Display file content history instead of file name history. The revisions that are listed include revisions of other files that were branched/copied (using `p4 integrate` and `p4 resolve` **-at**) to the specified revision. Revisions that were replaced by copying or branching are not displayed, even if they are in the history of the specified revision.
`-i`	Follow file history across branches. If a file was created by integration (`p4 integrate`), Helix server describes the file’s revisions and displays the revisions of the file from which it was branched (back to the branch point of the original file). File history inherited by renaming (`p4 move`) is always displayed, regardless of whether or not the `-i` option is used.
`-l`	List long output, with the full text of each changelist description.
`-L`	List long output, with the full text of each changelist description truncated at 250 characters.
`-m` *`max`*	List only the first *`max`* changes per file output.
`-p`	When used with the `-h` option, do not follow content of promoted task streams. This option is useful when there are many child task streams branched from the supplied *`file`* argument.
`-s`	Display a shortened form of output by ignoring non-contributory integrations (for example, integrations involving "branch into" or copy into" operations are not displayed)
`-t`	Display the time as well as the date.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	No	`list`

- Because `p4 filelog`'s output can be quite large when called with highly non-restrictive file arguments (for example, `p4 filelog //depot/...` displays the revision history for every file in the depot), `p4 filelog` commands may be subject to a `maxresults` limitation as set in `p4 group`.

- If both the `-i` and the `-m` ***`maxrev`*** options are used, and a branch is encountered within the most recent ***`maxrev`*** revisions of the file, the most recent ***`maxrev`*** revisions of the file prior to the branch point are also displayed. `p4 filelog -i` follows branches down to a depth of 50 levels, which should be more than sufficient for any site.
- Old revisions of temporary object files (file type modifier `+S`*`n`*) are displayed with an action of `purge`.

Examples

`p4 filelog //depot/proj1/...`	Display the revision history for every file under the depot's `proj1` directory.
`p4 filelog file1.c@100,@120`	Display the revision history for `file1.c` from changelists 100 through 120.
`p4 filelog file1.c#have,#head`	If you do not have the latest revision of `file1.c`, display revision history since your last sync.
`p4 filelog file1.c file1.h`	Show the revision history for files `file1.c` and `file1.h`, which reside locally in the current working directory.

Related Commands

To read additional information about each file	`p4 files`
To display file information in a format suitable for scripts	`p4 fstat`
To view a list of open files	`p4 opened`
To view a list of files you've synced to your client workspace	`p4 have`

p4 filelog (graph)

List the commit history of the specified file.

"Syntax" on page 19

```
p4 filelog [-d -m max --first-parent] file
```

"Syntax" on page 19

Description

List the commit history of the specified file, from the most recent commit to the initial commit.

By default, this command is optimized to NOT display history of a file that is deleted in the most recent commit, or has been removed previously.

The `--`***`first-parent`*** option follows only the first parent of each commit.

Options

`-d`	Show the full history of a deleted file. This option cannot be used with paths containing wildcards.
`-m` ***`max`***	List only the first ***`max`*** changes per file output.
`--`***`first-parent`***	Follows only the first parent of each commit.

p4 files

Provide information about files in the depot without accessing their contents.

"Syntax" on page 19

```
p4 [g-opts] files [-a -A -e] [-i][-m max] [[FileSpec][revSpec]]
p4 [g-opts] files -U unloadFileSpec
```

Description

This command lists each file that matches the file patterns provided as arguments. If a revision specifier is given, the files are described at the given revision. One file is listed per line, and the format of each line is:

```
depot-file-location#rev - action change changelist (filetype)
```

where:

- ***depot-file-location*** is the file's location relative to the top of the depot,
- ***rev*** is the revision number of the head revision of that file,
- ***action*** is the action taken at the head revision: **add**, **edit**, **delete**, **branch**, **move/add**, **move/delete**, **integrate**, **import**, **purge**, or **archive**,
- ***changelist*** is the number of the changelist in which the revision was submitted, and
- ***filetype*** is the Helix server file type of this file at the head revision.

Unlike most Helix server commands, **p4 files** reports on any file in the depot. I t is not limited to only those files that are visible through the client view. If a file pattern on the command line is given in client syntax, only files in the client workspace are shown.

Options

-a	For each file, list all revisions within a specified revision range, rather than only the highest revision in the range.
-A	Limit output to files in archive depots.
-e	Exclude deleted, purged, or archived files; the files that remain are those available for syncing or integration.
-i	Ignore the case of the file argument when listing files in a case sensitive server.

`-m` *`max`*	Limit output to the first *`max`* files.
`-U` *`unloadfile`*	List only files in the unload depot. See `p4 unload` for details.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

- The specified revision can be a revision range. In this case, only those files with revisions within the specified range are listed, and by default, only the highest revision in that range is listed. (To display information for all files within a revision range, use `p4 files -a`.)
- Because the output of `p4 files` can be quite large when called with highly non-restrictive file arguments (for example, `p4 files //depot/...` prints information about all the files in the depot), it may be subject to a `maxresults` limitation as set in `p4 group`.

Examples

`p4 files //depot/...`	Provides information about all files in the depot.
`p4 files //clientname/...`	Provides information about all depot files visible through the client view.
`p4 files -i //CLIENTName/...`	Case-insensitive version of `p4 files //clientname/...`
`p4 files @2011/03/10`	Provides information about all depot file revisions that existed on March 10, 2011.
`p4 files @2011/03/31:08:00,@2011/03/31:17:00`	Lists all files and revisions changed during business hours on March 31, 2011.
`p4 files //depot/proj2/...@p2lab`	Lists files and revisions under the directory `//depot/proj2/...` tagged by label `p2lab`.

`p4 files //depot/file.c`	Show information on the head revision of `//depot/file.c`. (that is, the *highest* revision in the implied range of `#1,#head`).
`p4 files -a //depot/file.c`	Show information on every revision of `//depot/file.c` (that is, *all* revisions in the implied range of `#1,#head`).
`p4 files -A //arch/depot/proj/...`	If an administrator has used `p4 archive` to transfer `//depot/proj/...` to an archive depot named `arch`, displays information about the files in the archived project.

Related Commands

To list the revision history of files	`p4 filelog`
To see a list of all currently opened files	`p4 opened`
To see a list of the file revisions you've synced to	`p4 have`
To view the contents of depot files	`p4 print`

p4 files (graph)

List the files in the repo.

"Syntax" on page 19

```
p4 files [-m max] file ...
```

Description

List the files in the repo. If client syntax is used to specify the file argument, the client view mapping is used to determine the corresponding repo files.

Options

`-m max`	Limit output to the first ***max*** files.

"Syntax" on page 19

Examples

`p4 files //bruno_1666/...`

The depot for **`projectA`** is not of type **`graph`**, so we see metadata, such as "add change 6 (text)". The depot for **`projectB`** is of type **`graph`**, so we see only the file name:

```
//depot/projectA/README.md#1 - add change 6 (text)
//depot/projectA/src/feature.cc#1 - add change 6 (text)
//depot/projectA/src/main.cc#1 - add change 6 (text)
//repo/projectB/Hero/IgnoreSubviewModPreprocessor.swift
//repo/projectB/Hero/MatchPreprocessor.swift
//repo/projectB/README.md
```

p4 fix

Link jobs to the changelists that fix them.

"Syntax" on page 19

```
p4 [g-opts] fix [-d] [-s status] -c changelistjobName ...
```

Description

The **p4 fix** command links jobs (descriptions of work to be done) to a changelist (a set of changes to files that does the work described by a job).

If the changelist has not yet been submitted, the job appears on the **p4 submit** or **p4 change** form for the changelist to which it's linked, and under normal circumstances, the status of the job is changed to **closed** when the changelist is submitted. If the changelist has already been submitted when you run **p4 fix**, the job's status is changed to a default status (typically **closed**) immediately.

To change a job status to something other than the default status (typically **closed**) when you submit a changelist, supply the **-s** option to **p4 fix**, **p4 submit**, or **p4 change**.

Because described work can be fixed over multiple changelists, one job can be linked to multiple changelists. Because a single changelist might fix ten bugs, multiple jobs can be linked to the same changelist. You can do this in one command execution by providing multiple jobs as arguments to **p4 fix**.

Options

-c ***changelist***	The changelist to mark as fixed.
-d	Delete the fix record for the specified job at the specified changelist. The job's status will not change.

`-s` ***`status`***	Upon submission of the changelist, change the job's status to *status*, rather than the default value `closed` (or some other value as defined in the `Presets:` of field 102 of the `p4 jobspec` form). If the changelist to which you're linking the job been `submitted`, the status value is immediately reflected in the job's status. If the changelist is `pending`, the job status is changed on submission of the changelist, provided that the `-s` option is also supplied to `p4 submit` and the desired status appears next to the job in the `p4 submit` form's `Jobs:` field. To leave a job unchanged, use the special status of `same`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`

- Because the format of jobs can be changed from site to site, it is possible that the jobs on your system no longer have a `Status:` field. If so, you can still link jobs to changelists with `p4 fix`, but Helix server will not change any of the job fields' values when the changelist is submitted.
- You can change a fixed or unfixed job's status at any time by editing the job with `p4 job`.
- Another way to fix (or unfix) a job is to add it to (or delete it from) the `Jobs:` field of an unsubmitted changelist's `p4 submit` or `p4 change` form.
- You can't `p4 fix` a job to the default changelist; instead, add the job to the `Jobs:` field of the default changelist's `p4 submit` form when submitting it to the depot.
- If you use `p4 fix -s` ***`status`*** on a job, and then use the `-s` option with `p4 submit` or `p4 change`, the `Jobs:` field of the changelist's form will also require a status value (the default value being the one specified by `p4 fix -s` ***`status`***). The job(s) will be assigned the specified *status* upon successful submission of the changelist. If no status value is specified in the form, the error message:

  ```
  Wrong number of words for field 'Jobs'.
  ```

  is displayed.

  `p4 fix -s` ***`status`***, `p4 submit -s`, and `p4 change -s` are intended for use in conjunction with defect tracking systems.

Under normal circumstances, end users do not use these commands, and use `p4 submit` and `p4 change` without the `-s` option. In this case, only the job number is required in the `Jobs:` field, and each job's status is set to a default value (typically `closed`) on completion of the submit.

Examples

`p4 fix -c 201 job000141 job002034`	Mark two jobs as being fixed by changelist 201. If changelist 201 is still `pending`, the jobs' status is changed to `closed` when the changelist is submitted.
`p4 fix -c 201 -s suspended job002433`	Mark `job002433` as `suspended`, rather than `closed`, when changelist 201 is submitted. Requires use of the `-s` option with `p4 submit`.

Related Commands

To add or delete a job from a pending changelist	`p4 change`
To add or delete a job from the default changelist	`p4 submit`
To view a list of connections between jobs and changelists	`p4 fixes`
To create or edit a job	`p4 job`
To list all jobs, or a subset of jobs	`p4 jobs`
To change the format of jobs at your site (superuser only)	`p4 jobspec`
To read information about the format of jobs at your site	`p4 jobspec -o`

p4 fixes

List jobs and the changelists that fix them.

"Syntax" on page 19

```
p4 [g-opts] fixes [-i] [-m max] [-j job] [-c changelist]
[FileSpec[revRange]...]
```

Description

After a job has been linked to a particular numbered changelist with `p4 fix`, `p4 change`, or `p4 submit`, the job is *fixed* by the changelist (even if the changelist is still pending). The `p4 fixes` command lists changelists and the jobs they fix.

If invoked without arguments, `p4 fixes` displays all fix records. Fix records are displayed in the following format:

```
jobname fixed by change changelist on date by user (status)
```

You can limit the listed fixes by combining the following options when calling `p4 fixes`:

- Use the `-c` ***changelist*** option to list only the jobs fixed by that pending or submitted changelist.
- Use the `-j` ***job*** option to list only those pending or submitted changelists that fix that job.
- Provide one or more file pattern arguments. If you provide a file argument, only submitted changelists affecting files that match the file patterns are listed. Pending changelists are not included. If a revision specifier or revision range is included, only submitted changelists that affected files at the given revisions are listed. You can use the `-i` option with a file pattern argument to include fixes made by changelists that were integrated into the specified files.
- Use the `-m` ***max*** option to limit the output to the first `max` fixes.

Note
This command runs in lockless mode if "db.peeking" on page 737 is set to `r`

Options

`-c` ***changelist***	Limit the displayed fixes to those that include the specified changelist.

`-i`	Include fixes made by changelists that affected files integrated into the specified files.
`-j jobname`	Limit the displayed fixes to those that include the specified job.
`-m max`	List only the first ***max*** fixes.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

Examples

`p4 fixes //depot/proj1/...`	Display all fixes made by submitted changelists that included any files under `//depot/proj1`.
`p4 fixes file.c`	Display all fixes made by submitted changes that included any and all revisions of `file.c`.
`p4 fixes file.c#5`	Display all fixes made by submitted changes that included revisions 1 through 5 of `file.c`.
`p4 fixes file.c#5,5`	Display only those fixes associated with the changelist in which `file.c#5` was submitted.
`p4 fixes -c 414`	Display all jobs fixed by pending or submitted changelist 414.

Related Commands

To create or edit an existing job	`p4 job`
To list all jobs known to the system	`p4 jobs`
To attach a job to a particular changelist; the job is fixed by that changelist	`p4 fix`
To change the format of jobs at your site (*superuser only*)	`p4 jobspec`
To read information about the format of jobs at your site	`p4 jobspec` -o

p4 flush

Update a client workspace's have list without actually copying any files.

"Syntax" on page 19

```
p4 [g-opts] flush [-f -L -n -q] [[FileSpec][revSpec] ...]
```

Description

> **Warning**
> `p4 flush` can cause commands to behave in unexpected ways, so use it only when the situation is appropriate. See the two "Examples" on the next page.

The `p4 flush` command performs only Step 2 of `p4 sync` *`FileSpec`*two-step operation:

Step 1: The file revisions in the *`FileSpec`* are copied from the depot to the client workspace.

Step 2: The workspace's *have list* (which tracks which file revisions have been synced, and is managed by the Perforce service) is updated to reflect the new client workspace contents.

Under most circumstances, this is not desirable, because a client workspace's have list should always reflect the workspace's true contents. However, if the workspace's contents are already out of sync with the have list, `p4 flush` can sometimes be used to bring the have list in sync with the actual contents. Because `p4 flush` performs no actual file transfers, this command is much **faster** then the corresponding `p4 sync`.

Options

`-f`	Force the flush. Helix server performs the flush even if the client workspace already has the file at the specified revision.
`-L`	For scripting purposes, perform the flush on a list of valid file arguments in full depot syntax with a valid revision number.
`-n`	Display what the results of the flush would be without actually performing the flush. This lets you make sure that the flush does what you think it will do before you do it.
`-q`	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`read`

- Because `p4 flush` updates the have list without copying files, and `p4 sync` -f updates the client workspace to match the have list, `p4 flush` ***`files`*** followed by `p4 sync -f` ***`files`*** is almost equivalent to `p4 sync`***`files`***. This means that a bad flush can be almost entirely fixed by following it with a `p4 sync` -f of the same file revisions that were originally flushed.

  Unfortunately, this is not a complete remedy, because any file revisions that were deleted from the have list by `p4 flush` will remain in the client workspace even after the `p4 sync` -f. In this case, you will need to manually remove deleted file revisions from the client workspace.
- `p4 flush` is an alias for `p4 sync -k`.

Examples

- Ten users at the same site need to set up new, identical client workspaces from the same depot at a remote location over a slow link. Typically, each user would run identical `p4 sync` commands, but if bandwidth is limited, there's a faster way:
  - One user runs `p4 sync` ***`files`*** from the client workspace `firstworkspace`.
  - The other users copy the newly synced files from the first user's client workspace into their own client workspaces using their local OS file-copying commands.
  - The other users run `p4 flush` ***`files`*** `@`***`firstworkspace`***, which brings their client workspaces' have lists into sync with the files copied into the client workspaces in the last step.

    Because `p4 flush` moves no files across the slow link, the process can be much faster then running the same `p4 sync` command ten separate times.
- Joe has a client workspace called `joe` that has a `Root:` of
  `/usr/joe/project1/subproj`
  and a `View:` of
  `//depot/joe/proj1/subproj/... //joe/...`
  Joe decides that all the files under `/usr/joe/project1` need to be included in the workspace, and uses `p4 client` to change the `Root:` to `/usr/joe/project1` and the `View:` to `//depot/joe/proj1/... //joe/...`.

  This keeps his current client workspace files in the same place while extending the scope of the workspace to include other files. But when Joe runs his next `p4 sync`, he is surprised to see that Helix server deletes every non-open file in the client workspace and replaces it with an identical copy of the same file!

"Syntax" on page 19

Helix server behaves this way because:

- the have list describes each file’s location relative to the client root
- the physical location of each file is determined when each Helix server command is run.

Therefore:

- Helix server thinks that each file has been relocated
- `p4 sync` deletes the file from its "old" location and copies it into its "new" location

The efficient solution in this case is to use **`p4 flush #have`** to update the client workspace’s have list to reflect the "new" locations of the files without actually copying any files.

Related Commands

`p4 flush` is an alias for `p4 sync` **-k**	`p4 sync` **-k**
To copy files from the depot to the client workspace	`p4 sync`
To bring the client workspace in sync with the have list after a bad **`p4 flush`**	`p4 sync` **-f**

p4 fstat

Dump file info in format suitable for parsing by scripts.

"Syntax" on page 19

```
p4 [g-opts] fstat [-F filter -L -T fields -m max -r] [-c|-e
change]
                  [-Ox -Rx -Sx] [-A pattern ] [-U] file[rev]...
```

Description

The **p4 fstat** command dumps information about each file, with information for each field on a separate line. The output is best used within a Helix C/C++ API application where the items can be accessed as variables, but is also suitable for parsing by scripts.

The only argument required for the **p4 fstat** command, is the ***file[rev]*** argument. All other options relate to limiting the set of files operated on or controlling the amount and display of information for the selected files.

- To change the field on which output is sorted, use one of the **-Sx** options, and to reverse sort order, use the **-r** option.
- Use the **-m *max*** option to limit the output to the first ***max*** files.
- To filter the output on some function of the form fields (for example, all files larger than a certain size and with a certain filetype), use the **-F *filter*** option.
- To limit output to the set of fields specified in a ***fields*** argument, use the **-T *fields*** option. The list of field names can be separated by spaces or commas.

The head type fields, for example, **headTime**, return information for the file revision provided for the file argument. If no specific revision is given, it returns information for the head revision.

Form Fields

The fields shown will vary with the selected file.

Field Name	Description	Example/Notes
attr-*name*	Attribute value for name	**attr-myAttr critical**
attrProp-*name*	Set if **attr-*name*** is a propagating attribute	**attrProp-myAttr**

Field Name	Description	Example/Notes
`clientFile`	Local path to file (in local syntax by default, or in Helix server syntax with the `-Op` option) For files containing the special characters `@`, `#`, `*`, and `%`, the `clientFile` displays the special character.	`/staff/userid/src/file.c` (or `//workspace/src/file.c` in Helix server syntax)
`depotFile`	Depot path to file. For files containing the special characters `@`, `#`, `*`, and `%`, the filename is displayed containing the ASCII expression of the character's hexadecimal value.	`//depot/src/file.c`
`movedFile`	Name in depot of moved to/from file	`//depot/src/file.c`
`path`	Local path to file	`//workspace/src/file.c`
`isMapped`	Set if the file is mapped to current client workspace	`isMapped`
`shelved`	Set if file is shelved	`shelved`
`headAction`	Action taken at head revision, if in depot	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`.
`headChange`	Head revision changelist number, if in depot	`124`
`headRev`	Head revision number, if in depot	`124`
`headType`	Head revision type, if in depot	`text`, `binary`, `text+k`, etc. (see "File types" on page 707.)
`headCharset`	Head charset	for unicode files

Field Name	Description	Example/Notes
`headTime`	Head revision changelist time, if in depot. Time is measured in seconds since 00:00:00 UTC, January 1, 1970	`919283152` is a date in early 1999
`headModTime`	Head revision modification time (the time that the file was last modified on the client before submit), if in depot.	`919280483` is a date in early 1999
`movedRev`	Head revision of moved file	`157`
`haveRev`	Revision last synced to workspace, if on workspace	`23`
`desc`	Changelist description (if using `-e` ***`changelist`*** and if the file was part of ***`changelist`***)	A Helix server changelist
`digest`	MD5 digest of a file (requires `-Ol` option)	A 32 hexadecimal digit string. Based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.
`fileSize`	File length in bytes (requires `-Ol` option)	`63488` Based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.

Field Name	Description	Example/Notes
`action`	Open action, if opened in your workspace	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`.
`type`	Open type, if opened in your workspace	A Helix server file type
`charset`	Open charset	(for unicode files)
`actionOwner`	User who opened the file, if open	A Helix server username
`workRev`	open revision, if opened	
`change`	Open changelist number, if opened in your workspace	`75331`
`resolved`	The number, if any, of resolved integration records	`5`
`unresolved`	The number, if any, of unresolved integration records	`2`
`reresolvable`	The number, if any, of re-resolvable integration records	`1`

`otherOpen`	The number of other users who have the file open, blank if no other users have the file open	`1`, `2`, `3`… ***n***, preceded by ***n*** records listing the users (`0` through ***n***-`1`) with `otherOpen`***n***, `otherAction`***n***, and `otherLock`***n*** fields as applicable. For example:
		`... otherOpen 3`
		`...... otherOpen0`
		`user1@ws1`
		`...... otherOpen1`
		`user2@ws2`
		`...... otherOpen2`
		`user3@ws3`

"Syntax" on page 19

Field Name	Description	Example/Notes
`otherOpen`*n*	For each user with the file open, the workspace and user with the open file	`user123@workstation9`
`otherLock`	Present and set to null if another user has the file locked, otherwise not present	`otherLock`
`otherLock`*n*	For each user with the file locked, the workspace and user holding the lock	`user123@workstation9` Because only one user at a time can lock a file, if *n* is set, *n* is always `0`.
`otherAction`*n*	For each user with the file open, the action taken	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`.
`otherChange`*n*	For every changelist with the file open, the changelist	75612
`openattr-`*name*	For every changelist with the file open, the attribute value for name	`attr-`*name*
`openattrProp-`*name*	Set if `attr-`*name* is a propagating attribute	`attrProp-`*name*
`ourLock`	Present and set to null if the current user has the file locked, otherwise not present	`ourLock`
`resolveAction`*n* `resolveBaseFile`*n* `resolveBaseRev`*n* `resolveFromFile`*n* `resolveStartFromRev`*n* `resolveEndFromRev`*n*	Pending integration action, base file, base revision number, from file, starting, and ending revision, respectively.	For pending integration record information, use the `-Or` option.

Field Name	Description	Example/Notes
`totalFileCount`	The number of files examined, if sorted.	Appears in the first file's output when you use the `-m` ***max*** option in conjunction with one of the `-Sx` or `-r` sorting options.

Options

`-A` ***pattern***	Restrict displayed attributes to those that match the specified ***pattern***. For example, for the selected files, `-A` ***foo**** displays only attributes whose name starts with ***foo***.
`-c` ***change***	Display only files affected after the given changelist number. This operation is much faster than using a revision range on the affected files.
`-e` ***change***	Display only files affected by the given changelist number. This option is much faster than using a revision range on the affected files. When used with the `-Ro` option, only pending changes are considered to ensure that files opened for add are included. This option also displays the change description.
`-F` ***filter***	List only those files that match the criteria specified by ***filter***. See "Usage Notes" on page 222 for a discussion of filters. Filtering is not optimized with indexes for performance.
`-L`	For scripting purposes or automated reporting processes: report file information on a list of valid file arguments in full depot syntax with a valid revision number. File specifications that do not meet these requirements are silently ignored.
`-m` ***max***	Produce fstat output for only the first ***max*** files.
`-Oa`	Output attributes set by `p4 attribute`.
`-Ob`	Output the path, revision, type, full and relative local paths of the server archive file. Requires `admin` privilege.
`-Od`	Output the digest of an attribute.
`-Oe`	Output attribute values encoded as hex.
`-Of`	Output all revisions for the given files, suppressing the `other[...]` and `resolve[...]` fields.
`-Ol`	Output a `fileSize` field displaying the length of the file and a digest field for each revision. `p4 fstat -e` ***shelvedChange*** `-Rs -Ol` reports the file size and digest of files shelved at the specified change.

`-Op`	Display the `clientFile` in Helix server syntax, as opposed to local syntax.
`-Or`	Display pending integration record data for files open in the current workspace.
`-Os`	Shorten output by excluding client workspace data (for instance, the `clientFile` field).
`-r`	Sort the output in reverse order.
`-Rc`	Limit output to files mapped into the current workspace.
`-Rh`	Limit output to files on your have list; that is, to files synced to the current workspace.
`-Rn`	Limit output to files opened at revisions not at the head revision.
`-Ro`	Limit output to open files in the current workspace. You must use this option to get pending changes.
`-Rr`	Limit output to open files that have been resolved.
`-Rs`	Limit output to shelved files. Requires `-e` ***`changelist`*** option. `p4 fstat -e` ***`shelvedChange`*** `-Rs -Ol` reports the file size and digest of files shelved at the specified change.
`-Ru`	Limit output to open files that are unresolved.
`-Sd`	Sort by date.
`-Sh`	Sort by have revision.
`-Sr`	Sort by head revision.
`-Ss`	Sort by filesize.
`-St`	Sort by filetype.
`-T` ***`fields`***	List only those fields that match the field names specified by ***`fields`***. The list of field names can be separated by spaces or commas.
`-U`	Include files in the unload depot when displaying data. See `p4 unload` for details.
`g-opts`	See "Global options" on page 690. The `-s` global option (which prefixes each line of output with a tag describing the type of output as `error`, `warning`, `info`, `text`, or `exit`) can be particularly useful when used with `p4 fstat`.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

Filters

Use **`p4 fstat -F`** ***`filter`*** to limit the list of files to those that meet certain criteria.

You can use logical operators on any of the form fields displayed by **`p4 fstat`**. The usual comparison operators (**`=`**, **`>`**, **`<`**, **`>=`**, and **`<=`**) are available. Regular expression matching is supported by the regular expression matching operator (**`~=`**).

The following filter expression filters for files of a certain size whose **`headType`** field is set to **`text`**.

```
-F "fileSize > 100000 & headType=text"
```

Filters used for **`fstat`** are case-sensitive. All alphanumeric strings (including words including embedded punctuation) separated by whitespace are indexed as words.

Spaces between search terms in a filter are treated as boolean AND operations. To find files that contain any of the key/value pairs (boolean OR), separate the terms with the "|" character.

Ampersands (**`&`**) can be used as boolean ANDs as well; the boolean operators bind in the order **`&`**, **`|`**, space (highest precedence to lowest precedence). Use parentheses to change the grouping order.

Additionally, you can use the NOT operator (**`^`**) to negate the sense of some comparisons.

Search results can be narrowed by matching values within specific fields with the filter syntax "***`fieldname=value`***". The ***`value`*** must be a single token, including both alphanumeric characters and punctuation.

The wildcard "**`*`**" allows for partial word matches. The filter "***`fieldname=`*****`string*`**" matches "**`string`**", "**`stringy`**", "**`stringlike`**", and so on.

Date fields can be matched by expressing the filter date as ***`yyyy/mm/dd`*** or ***`yyyy/mm/dd:hh:mm:ss`***. If a specific time is not provided, the equality operator (**`=`**) matches the entire day.

To search for text containing characters that are filter expression operators, escape the characters with a backslash (**`\`**) character. To match the backslash character, escape it with an additional backlash (**`\\`**). Using backslashes to escape search queries has two special cases: you can escape the Helix server "**`...`**" wildcard with **`\...`**, and you can search for empty fields with **`\0`**.

The behavior of comparison operators depends on the type of field you're comparing against. All fields that `fstat` processes are text fields. The equality operator (`=`) or case-insensitive equality operator (`~=`) matches the file if the word given as the value is found anywhere in the specified field. The relational operators are of limited use here, because they match the file if *any* word in the specified field matches the provided value. Relational operators are always case-sensitive. For example, if a changelist has a `text` field `desc` that contains the phrase `bug not fixed`, and the filter is "`desc<fixed`", the file matches the filter, because `bug<fixed`.

Examples

`p4 fstat file.c`	Displays information on `file.c`.
`p4 fstat //....c@20,@now`	Displays information on all `.c` files after the checking-in of files under changelist 20.
`p4 fstat -Os file.c`	No client workspace information lines (`clientFile`) are displayed.
`p4 fstat -Osl file.c`	No client workspace information lines are displayed, but the `fileSize` and `digest` lines are displayed.
`p4 fstat -Os -Ol file.c`	Equivalent to `p4 fstat -Osl`.
`p4 fstat -F "clientFile=c:\\ws\\file.c" //depot/main/...`	If a path contains backslashes, escape them with backslashes.
`p4 fstat -F "clientFile~=c:\\ws\\[Ff]ile.c" //depot/main/...`	Use the `~=` regular expression modifier to specify a regexp that matches `File.c` and `file.c`.
`p4 fstat -Ol -F "fileSize < 1024 & headType=text" //depot/main/...`	Display information on all text files under `//depot/main/...` that are smaller than 1024 bytes in length.
`p4 fstat -T 'depotFile, headRev' file.c`	Display only the `depotFile` and `headRev` fields for `file.c`.

Related Commands

To read additional information about each file	`p4 files`
To display file information including change descriptions	`p4 filelog`

p4 fstat (graph)

List file info.

"Syntax" on page 19

`p4 fstat [-F` ***`filter`*** `-T` ***`fields`*** `-m` ***`max`***`]` ***`file`*** `...`

Description

For files in a depot of type `graph`, information is limited to what Git provides. For details, see the final example in the section "Examples" on page 226.

This command:

- lists information about files, one line per field
- is intended for use in Perforce API applications, where the output can be accessed as variables, but its output is also suitable for parsing from the client command output in scripts.

Fields that p4 fstat displays

Field	Description
`depotFile`	name in depot
`clientFile`	local path (host or Perforce syntax)
`isMapped`	set if file is mapped in the client
`headCommit`	the commit for the file at head rev
`headBlob`	the sha for the file at head rev
`headType`	head rev type, if in depo
`haveBlob`	the sha for the have rev on client
`haveCommit`	the commit for the have rev on client
`otherOpen#`	list of user@client with file opened
`otherAction#`	open action, if opened by someone else
`otherChange#`	changelist, if opened by someone else
`otherLock#`	user@client with file locked
`otherOpen`	set if someone else has it open

Field	Description
`otherLock`	set if someone else has it locked
`repo`	the name of the repo (.git suffix)
`repoName`	the name of the repo

If the file is opened in the current workspace, additional fields are displayed:

Field	Description
`action`	open action, if opened
`change`	open changelist#, if opened
`type`	open type, if opened
`actionOwner`	user who opened file, if opened
`workBlob`	sha of open revision, if opened
`ourLock`	set if this user/client has it locked

Options

`-F` ***`filter`***	The -F flag lists only files satisfying the filter expression. This filter syntax is similar to the one used for "p4 jobs" on page 283 **`-e jobview`** and is used to evaluate the contents of the fields in the preceding list. For example: **`-F "headType=binary & repoName=//repo/git"`** **Note** ■ Filtering is case-sensitive. ■ For performance reasons, filtering is not optimized with indexes.
`-m` ***`max`***	Limits output to the specified number of files.
`-T` ***`fields`***	List only those fields that match the field names specified by ***`fields`***. The list of field names can be separated by spaces or commas. For example: **`-T "depotFile, clientFile, headCommit"`**

"Syntax" on page 19

Examples

`p4 fstat //graphDepot1/repo1/... *`	Each file has its own Git SHA-1 value as a binary large object (blob).
	Both files have the same Git commit SHA-1 because they were in the same commit.
	`p4 fstat //graphDepot1/repo1/triggers.txt`
	`... depotFile //graphDepot1/repo1/triggers.txt`
	`... clientFile /opt/perforce/servers/17100/graph_ws/graphDepot1/repo1/triggers.txt`
	`... isMapped`
	`... headCommit 754462e860e40b4e9ce638fec2af308f06e90216`
	`... headBlob 8539b99903d69a65caca1d67a5771ebca3d75758`
	`... headType text`
	`... haveBlob 8539b99903d69a65caca1d67a5771ebca3d75758`
	`... haveCommit 754462e860e40b4e9ce638fec2af308f06e90216`
	`... repo //graphDepot1/repo1.git`
	`... repoName //graphDepot1/repo1`

p4 grant-permission (graph)

Assign the specified permission for the specified depot of type `graph` (or a repo in that depot) to the specified user or group.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] grant-permission -d graphDepot1 -g group [-r ref] -p
permission
p4 [g-opts] grant-permission -d graphDepot1 -u user [-r ref] -p
permission
p4 [g-opts] grant-permission -n //graphDepot1/reponame -g group
[-r ref] -p permission
p4 [g-opts] grant-permission -n //graphDepot1/reponame -u user [-
r ref] -p permission
p4 [g-opts] grant-permission -n //graphDepot1/reponame -g group -
r ref -p restricted-ref
p4 [g-opts] grant-permission -n //graphDepot1/reponame -u user -r
ref -p restricted-ref
```

Description

Note
An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

The user who creates a depot is the `owner` of that depot and always has full `admin` rights to the depot. An `admin` user can grant permissions, including the `admin` permission, to groups and users of the depot or a repo it contains.

This command does not verify the existence of the specified user or group. Nor does this command verify the existence of the specified reference to a branch or tag. Therefore, you can use those options before or after creating the user, group, or reference to a branch or tag.

Permissions are additive. For example, you can give a user the `create-repo` permission, and later also give that user the `delete-repo` permission. After you have granted a permission, you can remove that permission with `p4 revoke-permission`.

> **Note**
> Certain permissions imply multiple permissions. Any such permissions are included automatically and cannot be revoked unless the permission that contains them is revoked. See "Permissions" below.

Options

`-d`	Applies to the depot and its repos. **Note** The Helix server superuser can specify `-d *` to grant the permission for all repos in all graph depots.
`-n`	Applies to the repo with the specified name.
`-g`	Applies to the specified group.
`-u`	Applies to the specified user.
`-r`	Optional for the `create-ref`, `delete-ref`, and `write-ref` permissions to specify the *ref*, a reference that corresponds to a repository's branch, such as *refs/heads/release*, or a tag, which might represent a release number, such as *refs/heads/rel-2.1.14*. See https://git-scm.com/book/en/v2/Git-Internals-Git-References. Required for the `restricted-ref` permission.
`-p`	Applies the specified permission.
g-opts	See "Global options" on page 690.

Permissions

When you grant a permission to a user or group, that user or group receives:

- the specified permission
- any permissions that are implied (implicitly included) with the explicit permission

The implied permissions associated with each assigned permission:

"Syntax" on page 19

Assigned Permission	Implied Permission								
	admin	force-push	delete-ref	create-ref	delete-repo	create-repo	write-all	write-ref	read
admin	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩
force-push		🟩	🟩						
delete-ref			🟩						
create-ref				🟩					
delete-repo					🟩				
create-repo						🟩			
write-all							🟩	🟩	🟩
write-ref								🟩	🟩
read									🟩

The capability associated with each permission:

`admin`	Grant and revoke permissions for the repo or depot specified.
`force-push`	Force an overwrite to the branch.
`delete-ref`	Delete the repository's branch or tag specified by `-r`*ref*.
`create-ref`	Create and set the initial value of `-r`*ref* for the specified repository's branch or tag. **Note** Does not include the `write-ref` permission.
`delete-repo`	Delete a repo in the specified depot of type `graph`.
`create-repo`	Allows a user to create a new repo on the graph depot through the `git push` command. For details, see *Helix4Git Administrator Guide*, which explains how to create and view repos.

`write-all`	Read and update files and references of the branch or tag specified by **`-r`***ref*. This permission allows a user to clone, pull, and push a repo that already exists in the graph depot. Does not include the **`create-repo`** permission. **Note** Modified files are not checked against the protection table nor for Git LFS locks.
`write-ref`	Read and update the repository's branch or tag specified by **`-r`***ref*. **Note** Modified files are checked against the protection table and for Git LFS locks. This is the sole permission that applies the protection setting in the protections table for a file or directory. See **`p4 protect`** and "Authorizing access" in *Helix Core Server Administrator Guide*.
`read`	Read the files in the specified depot or repo. This allows the user to clone and pull from the repo.

The following permission is exclusive and has no implied permissions:

`restricted-ref`	The specified user or group can update the ref, (branch or tag) specified by the **`-r`** option. If set, only users with this permission can perform an update. This prevents other users with write-ref or write-all from updating the specified reference. (See the final example.)

Examples

To assign the **`read`** permission for the specified depot of type **`graph`** (and its repos) to the specified user:

```
p4 grant-permission -p read -d graphDepot1 -u bruno
```

To limit the assignment to a specific repo within a specific graph depot:

```
p4 grant-permission -n //graphDepot1/repo8 -u bruno -p read
```

To limit a reference-related assignment (**`write-ref`**, **`write-all`**, **`create-ref`**, **`delete-ref`**, **`force-push`**) to a specific branch or tag:

```
p4 grant-permission -n //graphDepot1/repo8 -r "refs/heads/rel-2.1.14" -u
bruno -p create-ref
```

To make the reference-related assignment apply to more than one branch or tag, use the asterisk (*) wildcard:

```
p4 grant-permission -n //graphDepot1/repo8 -r "refs/heads/rel-*" -u bruno
-p delete-ref
```

To restrict a particular branch to a specified user or group, such as to restrict the master branch to the devops team:

```
p4 grant-permission -n //repo/test -g devops -p restricted-ref -r
refs/heads/master
```

Related Commands

To list the permissions currently granted	**`p4 show-permission`**
To remove a permission	**`p4 revoke-permission`**

p4 graph lfs-lock (graph)

Create a git lfs-lock.

"Syntax" on page 19

```
p4 graph lfs-lock [-u user ] -n repo -r refpath file
```

Description

The file path must be a fully-qualified repo path or a path relative to the root of the repo. See "Git LFS" in the *Helix4Git Administrator Guide*.

Options

`-n`	The name of the repo.
`-r`	The branch reference.
`-u`	The user that owns the lock.

Related Commands

To list the git LFS locks in the repo	"p4 graph lfs-locks (graph)" below
To remove a git lfs-lock	"p4 graph lfs-unlock (graph)" below

p4 graph lfs-locks (graph)

List the git lfs-locks in the repo.

"Syntax" on page 19

```
p4 graph lfs-locks -n repo [ -r refpath ] [ -p file ]
```

Options

`-n`	The name of the repo.
`-r`	The branch reference.
`-p`	The repo path.

Related Commands

Create a git lfs-lock	"p4 graph lfs-lock (graph)" on the previous page
To remove a git lfs-lock	"p4 graph lfs-unlock (graph)" below

p4 graph lfs-unlock (graph)

Remove a git lfs-lock.

"Syntax" on page 19

```
p4 graph lfs-unlock [ -f ] [ -u user ] [ -n repo -r ref file ] |
[ lockid ]
```

Options

`-f`	Allows removal of a lock owned by another user.
`-u`	The user that owns the lock.
`-n`	The name of the repo.
`-r`	The branch reference.

Related Commands

Create a git lfs-lock	"p4 graph lfs-lock (graph)" on page 231
To list the git lfs-locks	"p4 graph lfs-locks (graph)" on the previous page

p4 graph log (graph)

List commits.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 graph log -n //repo/name [options] commit...
```

Description

Lists the specified commits from the specified repo.

To show the full set of options:

```
p4 graph log -n repo [ -u user ] [ -A date ] [ -B date ] [ -p ] [
-N N ]
 [ -X N ] [ --oneline[=tree] [--no-abbrev] ] [ -a | -m N ] [
commit... ]
```

Options

`-a`	Display all commits (default is first 1000).
`-m` *`max`*	Display no more than max commits.
`-A` *`date`*	Display commits created after this date.
`-U` *`author`*	Display commits created by this author.
`-N minParents`	Display commits having at least this many parents.
`-X maxParents`	Display commits having at most this many parents.
`-p`	Display commits following first parent only.
`--oneline [=tree]`	Display one line per commit.
	The optional `tree` value for the `--oneline` option can be entered as: `--oneline=tree` to add a column in the output for tree-SHA-1 values
	The optional `no-abbrev` value for the `--oneline` option can be entered as: `--oneline --no-abbrev` to display SHA-1 values in the original 40 characters instead of the default 7 characters abbreviation.
	`--oneline=tree --no-abbrev` combines the two sub-options.

p4 graph rebase (graph)

Replay local history onto the target's new base.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 graph rebase [-r ref ] [-f] --repo repo --target target source
```

Description

Rebase moves the base of the target branch from its current location in the source to the source's HEAD. The source branch is not altered.

Rebase:

1. Determines the common base between the target and source, saving the local commits of the target branch committed after the base.
2. Resets the target branch HEAD to the source branch HEAD.
3. Applies the saved local changes as new commits on the target.

The target branch reference is updated to the last replayed commit.

Rebase requires `force-push` permissions. (See "Permissions" on page 228)

If conflicts are detected when replaying the diffs, the rebase fails, and any intermediary commits will be orphaned.

If the common base cannot be found, the rebase fails.

Options

`--repo`	Specifies the repo.
`--target`	Specifies the branch reference whose history would be changed by rebasing its local changes onto its new base.
`-r`	Allows an alternative reference to be updated which can be tested without having updated the target. If the ref exists, `force-push` permissions are required. If the ref does not exist, `create-ref` permissions are required.
`-f`	Required if the `--target` is the default branch of the repo.

p4 graph show-ref (graph)

Display reference values.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 graph show-ref [ -n //repo/name ] [-a -u user -m max -t ref-
type ] [[-e|-E] nameFilter]
```

Options

`-n`	Specifies the repo name. If a repo name is specified, you have to have read permission for that repo. Otherwise you must be a super user.
`-a`	Displays all references from all repos.
`-u`	Displays only references readable by the user argument.
`-m`	Specifies the maximum references to display.
`-t`	Specifies the type of reference, tag or branch.
`-e` *`nameFilter`*	Lists references with a name that matches the *nameFilter* pattern. For example: `-e refs/heads/dev...`
`-e`	Uses the server's normal case-sensitivity rules.
`-E`	Makes the matching case-insensitive, even on a case-sensitive server.

p4 graph tag (graph)

Tag a commit with a name.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 graph tag -n //repo/name [-c comment] [-f] tag sha
p4 graph tag -n //repo/name -d tag
p4 graph tag [-n //repo/name] -l [-o] [-m max]
```

Description

Create, update, delete, or list tags.

Options

`-n`	Specify the repo.
`-f`	Force an update to an existing tag.

`-c`	Add a comment to a tag.
`-d`	Delete a tag.
`-l`	List the tags. If the repo-name is not entered, list tags in all repos.
`-o`	Detailed format.
`-m` ***`max`***	Displays at most ***`max`*** tags.

p4 graph tags (graph)

List tagged commits in all repos.

Note
For depots of type `graph` only.

"Syntax" on page 19

`p4 graph tags [-o] [-m max]`

Description

List tagged commits in all repos.

Options

`-o`	Detailed format.
`-m` ***`max`***	Displays at most ***`max`*** tags.

p4 grep

Print lines in files (or revisions of files) that match a pattern.

"Syntax" on page 19

```
p4 [g-opts] grep [-a -i -n -s -t] [-v | -l | -L] [-F | -G] [-A
num]
                        [-B num] [-C num]  -e patternfile[revRange] ...
```

Description

The **p4 grep** command searches for lines that match a given regular expression.

By default, **p4 grep** operates on the head revision. If the file argument specifies a revision, all files as of that revision number are searched. If the file argument has a revision range, only those files selected by that revision range are searched, and the highest revision in that range is used for each file.

By default, **p4 grep** searches at most 10,000 revisions. This limit is controlled by the `"dm.grep.maxrevs" on page 740` configurable.

The following example shows you can find all occurrences of a whole word:

```
$ p4 grep -e "readme" //depot/main/myDir/...
```

Options

-a	Search all revisions within the specified range, rather than only the highest revision in the range.
-A *num*	Display *num* lines of trailing context after matching lines.
-B *num*	Display *num* lines of trailing context before matching lines.
-C *num*	Display *num* lines of output context.
-e *pattern*	The ***pattern***s used by **p4 grep** are regular expressions comparable to those used in UNIX. See the syntax in the output of **p4 help grep**.
-F	Interpret the pattern as a fixed string.
-G	Interpret the pattern as a regular expression.
-i	Perform case-insensitive pattern matching. (By default, matching is case-sensitive.)

`-L`	Display the name of each selected file from which no output would normally have been displayed; scanning stops at the first match.
`-n`	Display a matching line number after the file revision number.
`-v`	Display files with non-matching lines.
`-l`	Display the name of each selected file from which output would have been displayed; scanning stops at the first match.
`-s`	Suppress error messages from files with more than 4096 characters in a single line. (By default, **`p4 grep`** abandons these files and reports an error)
`-t`	Treat binary files as text. (By default, only files of type text are selected for pattern matching.)
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`read`

p4 group

Add or delete users from a group, or set the `maxresults`, `maxscanrows`, `maxlocktime`, and `timeout` limits for the members of a group.

"Syntax" on page 19

```
p4 [g-opts] group [-a | -A] groupname
p4 [g-opts] group -d [-a | -F] groupname
p4 [g-opts] group -o groupname
p4 [g-opts] group -i [-a | -A]
```

Description

A *group* is a list of Helix server users. Use groups to set access levels in the `p4 protect` form, to limit the maximum amount of data that can be retrieved from Helix server by particular users with a single command, to set the timeout period for `p4 login` tickets, and to provide information for the `p4 ldapsync` command.

To delete a group, use `p4 group -d groupname`, or call `p4 group groupname` and remove all the users from the resulting form. Use the `-F` option with the `-d` option to force deletion and to remove the group from the protections table and from all groups.

Form Fields

Field Name	Type	Description
`Group:`	Read-only	The name of the group, as entered on the command line.
`MaxResults:`	Writable	The maximum number of results that members of this group can access from the service from a single command. The default value is `unset`. See "Usage Notes" on page 243 for more details.
`MaxScanRows:`	Writable	The maximum number of rows that members of this group can scan from the service from a single command. The default value is `unset`. See "Usage Notes" on page 243 for more details.

Field Name	Type	Description
`MaxLockTime:`	Writable	The maximum length of time (in milliseconds) that any one operation can lock any database table when scanning data. The default value is `unset`. See "Usage Notes" on page 243 for more details.
`MaxOpenFiles:`	Writable	The maximum number of files that a member of a group can open using a single command. See "Usage Notes" on page 243 for more details.
`Timeout:`	Writable	The duration (in seconds) of the validity of a session ticket created by `p4 login`. The default value is 43,200 seconds (12 hours). To create a ticket that does not expire, set the `Timeout:` field to `unlimited`.
`PasswordTimeout:`	Writable	The length of time (in seconds) for which passwords for users in this group remain valid. To disable password aging, use a value of `unset`.
`LdapConfig`	Writable	The LDAP configuration to use when populating the group's user list from an LDAP query. For more information, see `p4 ldapsync`.
`LdapSearchQuery`	Writable	The LDAP query used to identify the members of the group. For more information, see `p4 ldapsync`.
`LdapUserAttribute`	Writable	The LDAP attribute that represents the user's username. For more information, see `p4 ldapsync`.
`Subgroups:`	Writable, multi-line	Names of other Helix server groups. To add all users in a previously defined group to the group you're presently working with, include the group name in the `Subgroups:` field of the `p4 group` form. Note that user and group names occupy separate namespaces, and thus, groups and users can have the same names. Every member of any previously defined group you list in the `Subgroups:` field will be a member of the group you're now defining.

Field Name	Type	Description
`Owners:`	Writable, multi-line	Names of other Helix server users. Group owners without `super` access are permitted to administer this group, provided that they use the `-a` option. Group owners are not necessarily members of a group; if a group owner is to be a member of the group, the userid must also be added to the `Users:` field. The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Users:`	Writable, multi-line	The Helix server usernames of the group members. Each user name must be typed on its own line, and should be indented.

Options

`-a`	Allow a (non-superuser) group owner to administer the group. The user must be listed in the `Owner:` field of the group.
`-A`	Allow a user with `admin` access to add a new group. Existing groups cannot be modified when this option is used.
`-d` *`groupname`*	Delete group *groupname*. The members of the group are affected only if their access level or `maxresults` value changes as a result of the group's deletion.
`-F` *`groupname`*	Used only with the `-d` option, forces the deletion of the specified group, and also removes the group from the protections table and from all groups.
`-i`	Read the form from standard input without invoking the user's editor. The new group specification replaces the previous one.
`-o`	Write the form to standard output without invoking the user's editor.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super` (`admin` for `p4 group -A`) (`list` for `p4 group -o` or `-a`)

- Referring to a (nonexistent) user in a group definition does not create the user, nor does it consume a license; use the `p4 user` command to create users.
- Ticket `Timeout` and `PasswordTimeout` values for users who belong to multiple groups are calculated the same way as `maxresults` values: the largest `timeout` value for all the groups of which the user is a member (including `unlimited`, but ignoring `unset`). Users in no groups have the default ticket `Timeout` value of `43200` and `PasswordTimeout` value of `unset`. To create a ticket that does not expire, set the `Timeout` to `unlimited`.
- If you are using the `PasswordTimeout:` field to implement password aging, a 30-day timeout is 2,592,000 seconds.
- As the number of files in the depot grows, certain commands can significantly slow down the service if called with no parameters, or if called with non-restrictive arguments. For example, `p4 print //depot/...` will print the contents of every file in the depot on the user's screen, and `p4 filelog //depot/...` will attempt to retrieve data on every file in the depot at *every revision*.

  The Helix server superuser can limit the amount of data that Helix server returns to the user by setting the `MaxResults` value for groups of users. The superuser can also limit the amount of data scanned (whether returned to the user or not) by setting the `MaxScanRows` value, and the length of time any database table can be locked in by any single operation by setting the `MaxLockTime` value. Equally, the `MaxOpenFiles` field can be set to specify the maximum number of files that a group member can open at any given time.

  If any of the `MaxResults`, `MaxScanRows`, `MaxLockTime`, or `MaxOpenFiles` limits are violated, the request fails and the user is asked to limit his query.

  If a user belongs to multiple groups, the service computes her `MaxResults` value to be the maximum of the `MaxResults` for all the groups of which the user is a member (removing the limit if it encounters a setting of `unlimited`, but ignoring any settings still at the default value of `unset`). If a particular user is not in any groups, her `MaxResults` value is `unset`. (The user's `MaxScanRows`, `MaxLockTime`, and `MaxOpenFiles` limits are computed in the same way.)

  The speed of most hardware should make it unnecessary to ever set a `MaxResults` value below 10,000, a `MaxScanRows` value below 50,000, or a `MaxLockTime` value below 1,000.

A user can also set these limits by specifying them on a per-command basis for some commands. Values set for individual commands, override values set using **p4 group**. To disable overriding **p4 group** settings, set "server.commandlimits" on page 807`=2`.

- To unload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough that users will not need to ask for assistance with `p4 unload` or `p4 reload` operations.
- To display a group's `maxresults`, `maxscanrows`, `maxlocktime`, and `timeout` limits, use `p4 groups -v groupname`.
- Use `p4 help maxresults` to obtain the list of commands that are affected by any of the four limiting values.
- See also the following topics in *Helix Core Server Administrator Guide*:
  - Limiting database queries
  - Limiting simultaneous connections

Related Commands

To modify users' access levels	`p4 protect`
To view a list of existing groups	`p4 groups`
To synchronize LDAP and Helix server groups	`p4 ldapsync`

p4 groups

List groups of users.

"Syntax" on page 19

```
p4 [g-opts] groups [-m max] [-v] [group]
p4 [g-opts] groups [-m max] [-i [-v]] user | group
p4 [g-opts] groups [-m max] [-g | -u | -o] name
```

Description

Shows a list of all current groups of users as created by `p4 group`. Only the group names are displayed.

If the optional ***`user`*** argument is provided, only the groups containing that user are listed. If the optional ***`group`*** argument is provided, only groups containing the named group are listed.

Use the `-i` option to include groups to which the user (or group) belongs by means of being a member of a subgroup. If a group argument is given, only groups that contain the named group are displayed.

Use the `-v` option to display the `MaxResults`, `MaxScanRows`, `MaxLockTime`, and `Timeout` values for the named group, or, if no group is specified, for all groups.

Use the `-m` ***`max`*** option to limit the output to the first ***`max`*** groups.

Options

`-g` ***`name`***	List groups with the specified name.
`-i`	Display groups to which the ***`user`*** or ***`group`*** is an indirect member (that is, by means of inclusion in a subgroup).
`-m` ***`max`***	List only the first ***`max`*** groups.
`-o` ***`name`***	List groups owned by the named user.
`-u` ***`name`***	List groups for whom the specified user is a member.

`-v`	Display verbose output: include `MaxResults`, `MaxScanRows`, `MaxLockTime`, and `Timeout` values.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- To see all the members of a particular group, use `p4 group-o` *`groupname`*. This variation of `p4 group` requires only `list` access.

Examples

`p4 groups bob`	Display the names of all groups of which user `bob` is a member.

Related Commands

To create or edit an existing group of users	`p4 group`
To view a list of all the members and specifications of a particular group	`p4 group-o` *`groupname`*
To set Helix server access levels for the members of a particular group	`p4 protect`

p4 have

List files and revisions that are synced to the client workspace.

"Syntax" on page 19

```
p4 [g-opts] have [--graph-only] [file...]
```

Description

List the files and revisions that are copied to the client workspace with " p4 sync" on page 574.

If file patterns are provided, the list is limited to the files that meet both criteria:

- match one of the patterns
- mapped to the client view

`p4 have` lists the files, one per line, in the format:

`depot-file#revision-number - local-path`

- ***`depot-file`*** is the path to the file in *depot syntax*.
- ***`revision-number`*** is the *have revision*, the revision presently in the current client workspace
- ***`local-path`*** is the path as represented in terms of the local filesystem (that is, in *local syntax*).

Options

`file ...`	List revision numbers of the currently-synced files. If file name is omitted, list all files synced to this client workspace.
`--graph-only`	For hybrid workspaces, limit the report to graph depots only.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`

- Some Helix server documentation refers to a client workspace's *have list*. The **have list** is the list of files reported by `p4 have`, and is the list of file revisions that are most recently synced from the depot.

  The have list excludes:

  - files that exist in your client workspace but not in the depot
  - files at deleted revisions

  For instance, if you use `p4 add` to open a newly created file in your client workspace for add, or if you use `p4 integrate` to create a group of files in your client workspace, but they are not yet submitted, the new files do not appear in the output of `p4 have`.

  The set of all files in your client workspace is the union of the set of files listed by `p4 have` with the set of files listed by `p4 opened`.

- For files containing the special characters `@`, `#`, `*`, and `%`, the ***depot-file*** field shows the ASCII expression of the character's hexadecimal value, and the ***local-path*** shows the special character. For example:

  ```
  //depot/status/100%25.txt#1 - /staff/status/100%.txt
  ```

Attempting to sync an open file to an earlier version

If a file is open for edit, `p4 have` reflects the most recent attempt at `p4 sync`. For example, if you open a file for edit that is at version #3, and then attempt to sync that file to an earlier version, such as #1, the sync attempt fails. This is a feature to prevent you from accidentally overwriting your work at version #3. However, `p4 have` optimistically assumes " p4 sync" on page 574 succeeded and lists the file at version #1. If you submit version #3, which increments the version to #4, and run `p4 have` again, `p4 have` lists version #4.

Examples

`p4 sync //depot/name/...` `p4 have //depot/name`
 `p4 sync //depot/name/...#4` `p4 have //depot/name`
	In each of these two pairs of commands:
	 The first `p4 have` shows that the highest revision of the file has been copied to the client workspace.
	 The second `p4 have` shows that the fourth revision is the revision currently in the client workspace.

Related Commands

To copy file revisions from the depot to the client workspace	`p4 sync`

List the commit most recently synced to the current workspace	"p4 have (graph)" below

p4 have (graph)

List the commit most recently synced to the current workspace

"Syntax" on page 19

```
p4 [g-opts] have [--graph-only] [file...]
```

Description

Lists the current branch and the currently-synced commit for this client workspace.

Options

`file ...`	List revision numbers of the currently-synced files. If file name is omitted, list all files synced to this client workspace.
`--graph-only`	For hybrid workspaces, limit the report to graph depots only.
`g-opts`	See "Global options" on page 690.

Examples

`p4 sync` `p4 have`	(depot of type `graph`) Suppose you `p4 sync` a repo, and then issue the `p4 have` command, the output for that repo is two lines, such as: `//graphDepot1/repo1 workspace1 b78d7fb2e8c50e refs/heads/master` which includes the repo name, the workspace name, the value of the commit SHA, and the branch name.

Note

If, in addition to the two lines, you also see "`File specific revisions:`", such as:

```
//graphDepot1/repo1 workspace1 refs/heads/master
b78d7fb2e8c50e workspace1
File specific revisions:
workspace1  readme.txt  3a16b5c563e01a
workspace1  index.html  3a16b5c563e01a
workspace1  eiffel-tower.png  f2d7b9b83a49d
```

the files listed after `File specific revisions` are NOT part of the commit on the master branch. Instead, for those files that are NOT on the master branch, we see the workspace, file name, and commit SHA. In this case, `readme.txt` and `index.html` files belong to the same commit SHA, and the file named `eifel-tower.png` belongs to a different commit SHA.

Possible causes that these three files are not on the master branch include:

- disk or network issue preventing the sync of a file
- the workspace is configured as `noclobber` and file specific revisions have occurred on files that been changed from `read-only` to `write` (see `noclobber` under `p4 client` "Usage Notes" on page 109)

Related Commands

To copy file revisions from the depot to the client workspace	`p4 sync`
(classic) List files and revisions that are synced to the client workspace.	"p4 have" on page 247

p4 heartbeat

Monitor the responsiveness of a target server, such as a standby server monitoring its master server to help the administrator determine if a failover might be needed.

"Syntax" on page 19

```
p4 [g-opts] heartbeat [-i interval] [-w wait] [-m
missingInterval]
     [-r missingWait] [-c count]
```

```
p4 [g-opts] heartbeat -t target [-i interval] [-w wait] [-m
missingInterval]
    [-r missingWait] [-c count]
```

Description

The `p4 heartbeat` command can be used to monitor a server. Both the server running `p4 heartbeat` and the server being monitored must be at 20.1 or later.

A typical use case is that a standby server uses the heartbeat to monitor the responsiveness of its master server, which can help the administrator determine if a failover might be needed (see "p4 failover" on page 190).

Heartbeat sends requests to a target server at regular intervals and monitors the responses. The server that is monitoring the target server can be set up to alert administrators about a change in the target server's status.

If the target server ...	the monitoring server can fire a trigger or extension of type ...
misses a response	`heartbeat-missing`
resumes its response	`heartbeat-resumed`
misses consecutive responses that reach the maximum count	`heartbeat-dead`

See Triggering on heartbeat in *Helix Core Server Administrator Guide*.

Note

- If you want the standby server to monitor the master server: on the standby server, configure the heartbeat thread as a startup background thread by setting the "startup.N" on page 816 configurable as follows: `p4 configure standby#startup.4=heartbeat`
- This command was designed for a standby server to monitor a master server for a possible failover scenario (see Failover in *Helix Core Server Administrator Guide*), but we recommend a human being, rather than a script, make any decision to perform a failover operation.
- This command can be used to monitor any server.
- Logging:
  - If the heartbeat command is started with `-t target`, the timestamp is logged in `commands.csv` and `all.csv`
  - Both first missing heartbeat and the maximum count of missing heartbeats exceeded are logged in `errors.csv`
  - To get verbose logging that includes every missing heartbeat, start Helix server with the following option: `-vheartbeat=3`
- This command increases network traffic between the target server and the monitoring server.
- The heartbeat thread runs until any of the following occur:
  - the heartbeat thread detects that the target server has died
  - the heartbeat thread is terminated explicitly
  - the heartbeat thread is terminated at server shutdown

  This is true whether the heartbeat thread runs as a startup process, like the startup processes for "p4 pull" on page 420, or you run `p4 heartbeat` in the foreground.

For an overview of Helix Core replication, see Replication in *Helix Core Server Administrator Guide*.

Options

`-t`	Specifies the target server to be monitored. If not specified, the command uses the value of "P4TARGET" on page 683. If neither `P4TARGET` nor the `-t` option is set, the command will fail. (See also `-t host:port` in Helix Core server (p4d) Reference.)
`-i`	Specifies the time in milliseconds to wait before sending another heartbeat request. If not specified, this interval defaults to the "net.heartbeat.interval" on page 770 configurable.
`-w`	Specifies the time in milliseconds to wait before the response is timed out. If not specified, this defaults to the "net.heartbeat.wait" on page 771 configurable.

`-m`	Specifies the time in milliseconds to wait before sending another heartbeat request after the previous response was missing. If not specified, this defaults to the "net.heartbeat.missing.interval" on page 771 configurable.
`-r`	Specifies the time in milliseconds to wait before the response times out after the previous response was missing. If not specified, this defaults to the "net.heartbeat.missing.wait" on page 772 configurable.
`-c`	Specifies the maximum number of consecutive missed heartbeats before the heartbeat is is considered non-responsive or dead.. If not specified, this defaults to the "net.heartbeat.missing.count" on page 773 configurable.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super` or `operator`

Related Commands

Fail over to a standby server.	"p4 failover" on page 190

p4 help

Provide command-line help for Helix server.

"Syntax" on page 19

```
p4 [g-opts] help
p4 [g-opts] help keyword
p4 [g-opts] help command
```

Description

`p4 help`

- displays a topic describing the named *command* or *keyword*
- is similar to this manual, but more concise and without hyperlinks

`p4 help` with no arguments lists all the available `p4 help` options.

`p4 help command` provides help on the named *command*.

`p4 help keyword` takes the following keywords as arguments:

Command and Keyword	Meaning	Equivalent Topic, Chapter, or Manual
`p4 help administration`	Help on specialized administration topics.	`p4 admin` topic
`p4 help charset`	Describes how to control Unicode translation.	`P4CHARSET` topic
`p4 help commands`	Lists all the Perforce commands.	"Commands" on page 45 - alphabetical list
`p4 help configurables`	Describes all of the server configuration variables.	"Configurables" on page 715 chapter
`p4 help dvcs`	Describes decentralized version control with Perforce.	*Using Helix Core Server for Distributed Versioning* manual

"Syntax" on page 19

Command and Keyword	Meaning	Equivalent Topic, Chapter, or Manual
`p4 help environment`	Lists the Perforce environment variables and their meanings.	"Environment and registry variables" on page 637 chapter
`p4 help extension`	Mentions `p4 help serverextensionintro` and `p4 help clientextensionintro`	"p4 extension" on page 186 topic and *Helix Core Extensions Developer Guide*
`p4 help filetypes`	Lists the Perforce filetypes and their meanings.	"File types" on page 707 chapter
`p4 help graph-depot`	Lists the commands for graph depots. A depot of type graph is used to store Git repos in the Helix server.	"Graph depot commands" on page 29 topic and *Helix4Git Administrator Guide*
`p4 help jobview`	Describes Perforce jobviews.	`p4 jobs` topic
`p4 help legal`	Lists the third-party software licenses that the **server** uses.	"License Statements" on page 854 topic
`p4 help -l legal`	Lists the third-party software licenses that the **local client** (such as P4V) uses.	
`p4 help networkaddress`	Help on network address syntax.	"P4PORT" on page 677 topic
`p4 help replication`	Describes specialized replication topics.	Deployment architecture in *Helix Core Server Administrator Guide*
`p4 help revisions`	Describes Perforce revision specifiers.	"File specifications" on page 695 chapter
`p4 help simple`	Provides short descriptions of the eight most basic Perforce commands.	Basic tasks in *Helix Core Server User Guide*
`p4 help streamintro`	Explains what streams are and the commands typically used with streams.	streams-based Tutorial and Streams chapter in *Helix Core Server User Guide*
`p4 help usage`	Lists the six options available with all Perforce commands.	"Global options" on page 690 chapter
`p4 help views`	Describes the meaning of Perforce views.	"Views" on page 702 chapter

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`none`

Related Commands

To view information about the current Helix server configuration	`p4 info`
To review a list of commands for depots of type graph for the git data model	"p4 help-graph (graph)" below

p4 help-graph (graph)

Provide on-line help for Helix server commands for depots of type graph.

"Syntax" on page 19

```
p4 [g-opts] help-graph
p4 [g-opts] help-graph keyword
p4 [g-opts] help-graph command
```

Description

To see ...	type ...
a list of graph depot commands	`p4 help-graph`

help for a specific command	`p4 help-graph command`
	
	Examples:
	`p4 help-graph add`
	`p4 help-graph switch`

help for graph depot administrators	`p4 help-graph administration`

"Syntax" on page 19

Related Commands

For help with "classic" commands, that is, commands unrelated to graph depots	"p4 help" on page 254
To view information about the current Helix server configuration	`p4 info`

p4 ignores

Displays the ignore mappings computed from the rules in "P4IGNORE" on page 660 files.

"Syntax" on page 19

```
p4 [g-opts] ignores [-v] [path ...]
p4 [g-opts] ignores -i [-v] path ...
```

Description

It's not always easy to tell why a file is being ignored during add and reconcile operations. To help with that, **`p4 ignores`** lets you see the mappings in Perforce syntax that are applied for any specified path.

If you do not specify a path, the mappings for the current working directory are displayed.

Using the `-v` option outputs additional information about the source of each set of mappings: the rule that generated them and the line number in the file where the rule is defined.

The `-i` option checks whether a filepath would be ignored; if it would, the path is returned. With the `-v` option, a more verbose message is returned explaining whether the filepath would be ignored and if it matched any ignore rules.

Options

`-i`	Test to see if the specified filepath would be ignored.
`-v`	Report additional information about the `P4IGNORE` rules in effect. When listing mappings, include the rules from which they were generated. When testing a filepath, include the rule that affected the output.
`path`	The path to be tested. Specify the path using local syntax.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`none`

Examples

Note that in the following table, the commands in the lefthand column were run in the `c:\workspace` directory.

`p4 ignores`	Display a list of mappings generated from `P4IGNORE` rules.
	`.../.p4root/...`
	`.../.p4root`
	`.../.p4config/...`
	`.../.p4config`
	`c:/workspace/.../builds/...`
	`c:/workspace/builds/...`
	`c:/workspace/.../builds`
	`c:/workspace/builds`

`p4 ignores -v`	Display a list of mappings generated from `P4IGNORE` rules, including the rules that they were generated from.
	`#FILE - defaults`
	`#LINE 2:**/.p4root`
	`.../.p4root/...`
	`.../.p4root`
	`#LINE 1:**/.p4config`
	`.../.p4config/...`
	`.../.p4config`
	`#FILE c:\workspace\.p4ignore`
	`#LINE 1:builds`
	`c:/workspace/.../builds/...`
	`c:/workspace/builds/...`
	`c:/workspace/.../builds`
	`c:/workspace/builds`

`p4 ignores -i file1 builds/file2`	Test to see whether `file1` and `file2` will be ignored. If a file is ignored, it will be returned. Otherwise, there will be no output. `c:\workspace\builds\file2 ignored`

`p4 ignores -i -v file1 builds/file2`	Test to see whether **`file1`** and **`file2`** will be ignored. If a file is ignored then it will be returned with information about the **`P4IGNORE`** rule that it matched. If it is not, it will be returned with a message saying that it is not ignored; if it matched a **`P4IGNORE`** exclusionary rule, information about that rule will also be returned.
	`c:\workspace\file1 not ignored`
	`c:\workspace\builds\file2 ignored by`
	`  c:\workspace\.p4ignore:1:builds`

p4 info

Display information about the current Helix server application and the shared versioning service.

"Syntax" on page 19

```
p4 [g-opts] info [-s]
```

Description

The `p4 info` command displays information about the Helix server application and the shared versioning service.

Here's an example of the output from `p4 info`. If the server were a replica of another server, that information would be supplied.

```
Client name: myserver-24-n102
Client host: myserver-24-n102.dhcp.perforce.com
Client unknown.
Current directory: /Users/jbujes
Peer address: 10.0.102.24:52492
Client address: 10.0.102.24
Server address: someaddress.perforce.com:1999
Server root: /depots/p4-1999
Server date: 2015/07/13 14:52:59 -0700 PDT
Server uptime: 147:34:34
Server version: P4D/LINUX26X86_64/2015.2.MAIN-TEST_ONLY/1199094
(2015/07/07)
ServerID: Master1999
Server services: standard
Server license: 500 users (expires 2017/01/31)
Server license-ip: qaplay.perforce.com
Case Handling: sensitive
```

To obtain the version of the Helix server application (`p4`), use `p4 -V`.

"Syntax" on page 19

Note
The output of `p4 -ztag info` includes `unloadSupport enabled` if the administrator has created a depot of type `unload`. If not, the output includes `unloadSupport disabled`.

Tip
To hide sensitive output from unauthenticated users, use the "dm.info.hide" on page 740 configurable.

Options

`-s`	Shortened output: exclude information (for example, the workspace root) that requires a database lookup.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`none`

Related Commands

To read Helix server's help files	`p4 help`
To display Helix Proxy connection information	`p4 proxy`
To view version information for your Helix server application	`p4 -V`

p4 init

Initializes a new Helix server.

Note
For distributed version control only. See *Using Helix Core Server for Distributed Versioning* (DVCS).

"Syntax" on page 19

```
p4 [-u user] [-d dir] [-c client] init [-h -q] [-c stream] [-Cx]
[-xi -n] [-p]
```

Description

Initialize a new personal (local) Helix server.

In order to run `p4 init`, you must have up-to-date and matching versions of the `p4` and `p4d` executables in your operating system path. You can download these executables from www.perforce.com.

Helix server stores its database files in the directory named `.p4root`. Helix server stores configuration settings in the `P4CONFIG` and `P4IGNORE` files at the top level of your directory. It is not necessary to view or update these files, but you should be aware that they exist.

After initializing your new server, run `p4 reconcile` to mark all of your source files to be added to Helix server, then `p4 submit` to submit them.

Options

`-c` *`stream`*	Specifies the stream to use as the mainline stream instead of the default `//stream/main`.
`-Cx`	Sets the case sensitivity of the installation. If *x* is set to `0`, your installation is case-sensitive, if set to `1` your installation is case-insensitive. Your client must match the case sensitivity of the server you're fetching from or pushing to.
`-d` *`directory`*	Specifies the directory in which Helix server initializes the server. Without this option, Helix server initializes the server in the current directory.
`-h`	Display help for this command, as it operates on the client.

`-n`	Configures the installation without unicode support. This is useful because the unicode capability of the local server must match that of the server you fetch from and push to.
`-q`	Suppresses informational messages.
`-p`	Specifies the address of a remote server whose case sensitivity and unicode settings you want to discover. Specifying this information makes your local server compatible with the remote server.
`-u` *`username`*	Specifies your Helix server user name.
`-xi`	Configures the installation with unicode support.
`g-opts`	See "Global options" on page 690.

Without `-xi` or `-n`, unicode support is detected by finding a `P4CHARSET` setting.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	N/A

Examples

`p4 init`	Initializes a new Helix server personal server.

Related Commands

Clone a new server	`p4 clone`

p4 integrate

Open files for branching or merging.

`p4 integrate` can be abbreviated as **`p4 integ`**.

"Syntax" on page 19

```
p4 [g-opts] integrate [options] fromFileSpec[revRange] toFile
p4 [g-opts] integrate [options] -b branch [-r] [toFileSpec
[RevRange] ...]
p4 [g-opts] integrate [options] -b branch -s fromFileSpec
[revRange] [toFile ...]
p4 [g-opts] integrate [options] -S stream [-r] [-P parent] [file
[revRange] ...]
```

For ***[options]***, you can use:

```
-c changelist  -Di  -f -h -O[b][r]  -n -m max -R[b][d][s] -q -v
```

Description

When you've made changes to a file that need to be propagated to another file, start the process with **`p4 integrate`**. The command includes four syntax variants, depending on whether the source and target files are specified using files, branches, or streams.

The simplest syntax variant is **`p4 integrate`** ***`fromFile toFile`***, which lets the versioning service know that changes in ***`fromFile`*** need to be propagated to ***`toFile`***, and has the following effects:

- If ***`toFile`*** does not yet exist, ***`fromFile`*** is copied to ***`toFile`***, then ***`toFile`*** is opened for **`branch`** in the client workspace.
- If ***`toFile`*** exists, and shares a common ancestor with ***`fromfile`*** as above, then ***`toFile`*** is opened for **`integrate`**. You can then use **`p4 resolve`** to propagate all of, portions of, or none of the changes in ***`fromFile`*** to ***`toFile`***.

  The **`p4 resolve`** command uses ***`fromFile`*** as ***`theirs`***, ***`toFile`*** as ***`yours`***, and the file with the most edits in common as the base.
- If ***`fromFile`*** was deleted at its last revision (and all previous changes have already been integrated between ***`fromFile`*** and ***`toFile`***), ***`toFile`*** is opened for **`delete`** in the client workspace.

- Whether you move files using `p4 move`, or whether you use native OS commands to rename files within your workspace (using `p4 reconcile` or `p4 status` to update your changelist to reflect the moves you made), `p4 integrate` automatically detects these actions, adjusts the source-to-target mappings appropriately, and schedules a filename resolve for each remapped file pair.

(Some of the available options modify this behavior. See "Options" on the facing page for details.)

The process is complete when you `p4 submit` ***`toFile`*** to the depot.

Note
If you integrate from a classic branch or other stream depot to a task stream, the files are not copied up to the parent unless they are edited and submitted first.

To specify multiple files, use wildcards in ***`fromFile`*** and ***`toFile`***. Any wildcards used in ***`fromFile`*** must match identical wildcards in ***`toFile`***. Helix server compares the ***`fromFile`*** pattern to the ***`toFile`*** pattern, creates a list of ***`fromFile`*** and ***`toFile`*** pairs, and performs an integration on each pair.

The syntax `p4 integrate` ***`fromFiles toFiles`*** requires you to specify the mapping between ***`fromFiles`*** and ***`toFiles`*** each time changes need to be propagated from ***`fromFiles`*** to ***`toFiles`***. Alternatively, use `p4 branch` to store the mappings between ***`fromFiles`*** and ***`toFiles`*** in a ***`branch view`***, and then use `p4 integrate -b` ***`branchview`*** whenever you need to propagate changes between ***`fromFiles`*** and ***`toFiles`***.

By default, files that have been opened for `branch` or `integrate` with `p4 integrate` are read-only in the client workspace. You can edit these files before submitting them using `p4 edit` to reopen the file for `edit`.

Whenever a ***`toFile`*** is integrated from a ***`fromFile`***, Helix server creates an *integration record* in its database that describes the effect of the integration. The integration record includes the names of the ***`fromFile`***, and ***`toFile`***, the revisions of ***`fromFile`*** that were integrated into ***`toFile`***, the new revision number for ***`toFile`***, and the action that was taken at the time of the integration. See `p4 integrated` for a full description of integration actions.

In most cases, `p4 integrate` performs a lazy copy. The contents of the file are not duplicated on the server because the integration record contains sufficient information to reproduce the file.

Tip
Alternatives to `p4 integrate`:

- `"p4 populate" on page 393` to quickly branch files without opening them in a workspace
- `"p4 copy" on page 129` to open files to be copied without scheduling any resolves. Consider whether the `-v` option of `p4 copy` is appropriate for your use case.
- `"p4 merge" on page 366` to open files to be merged and schedule resolves for all changes

Options

Basic Integration Options

`-b` ***`branchname`*** `[`***`toFiles`*** `...]`	Integrate the files using the ***`sourceFile`***/***`targetFile`*** mappings included in the branch view of ***`branchname`***. If the ***`toFiles`*** argument is included, include only those target files in the branch view that match the pattern specified by ***`toFiles`***. If a revision range is supplied with ***`toFiles`***, the range refers to source revisions, not target revisions.
`fromFiles` ***`toFiles`***	***`fromFiles`*** are called the *source files*; ***`toFiles`*** are called the *target files*. Any ***`toFiles`*** that **`p4 integrate`** needs to operate on must be included in the **`p4 client`** view.
`-n`	Display the integrations this command would perform without actually performing them.
`-v`	This option was deprecated because "p4 populate" on page 393 makes it obsolete. ~~Performs a "virtual" integration that does not modify client workspace files unless target files need to be resolved. After submitting a "virtual" integration, `p4 sync` can be used to update the workspace.~~
`-c` ***`changelist`***	Open the ***`toFiles`*** for **`branch`**, **`integrate`**, or **`delete`** in the specified pending changelist. If this option is not provided, the files are opened in the default changelist.
`-q`	Quiet mode, which suppresses normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Advanced Integration Options

`-b` ***`branchname`*** **`-s`** ***`fromFile`*** **`[`*****`revRange`*****`]`** **`[`*****`toFiles`*** **`...]`**	In its simplest form, **`p4 integrate -b`** ***`branchname`*** **`-s`** ***`fromFile`*** allows you to integrate files using the source/target mappings included in the branch view of ***`branchname`***, but include only those source files that match the patterns specified by ***`fromFile`***. In its more complicated form, when both ***`fromFile`*** and ***`toFile`*** are specified, integration is performed bidirectionally in two steps: 1. From ***`fromFile`*** to **`toFile`** 2. From ***`toFile`*** to **`fromFile`** This variation of **`p4 integrate`** was written to provide some needed functionality to graphical Helix server applications; it is unlikely that you'll need to use this more complex form.
`-b` ***`branchname`*** **`-r`** **`[`*****`toFiles`*** **`...]`**	Reverse the mappings in the branch view, integrating from the target files to the source files.
`-Di`	The **`-Di`** option modifies the way deleted revisions are treated. If the source file has been deleted and re-added, revisions that precede the deletion will be considered to be part of the same source file. By default, re-added files are considered to be unrelated to the files of the same name that preceded them. When the source file has been moved or renamed, the move/add and move/delete revisions are propagated as **`branch`** and **`delete`** revisions instead.
`-f`	Force the integration on all revisions of ***`fromFile`*** and ***`toFile`***, even if some revisions have been integrated in the past. Best used with a revision range.
`-h`	Don't automatically sync target files to the head revision before integrating. Use the have revision instead.
`-m` ***`max`***	Limit the command to integrating only the first ***`max`*** files.
`-Ob`	The **`-Ob`** option outputs the base revision for the merge (if any).
`-Or`	The **`-Or`** option outputs the resolves that are being scheduled.
`-Rb`	The **`-Rb`** option schedules a branch resolve instead of branching the target files automatically.
`-Rd`	The **`-Rd`** option schedules a delete resolve instead of deleting the target files automatically.

`-Rs`	The `-Rs` option skips cherry-picked revisions that have already been integrated. Using this option can improve merge results, but can also cause multiple resolves per file to be scheduled.
`-S` ***`stream`*** `[-P` ***`parent`***`]`	Integrates a stream to its parent. To reverse the direction of the mapping, use the `-r` option. To override the configured parent and integrate to a different target stream, specify `-P`. To submit integrated stream files, the current client must be switched to the target stream or to a virtual child stream of the target stream.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`open`

Examples

`p4 integ //depot/dev/... //depot/rel2/...`	Branch or merge all files in `//depot/dev/...` to the corresponding files in `//depot/rel2/...`. If there is no corresponding file in `//depot/rel2/...`, this creates it.
`p4 integ -b rel2br`	Branch or merge all ***`fromFiles`*** contained in the branch view `rel2br` into the corresponding ***`toFiles`*** as mapped through the branch view.
`p4 integ -b rel2br //depot/rel2/headers/...`	Branch or merge those ***`fromFiles`*** contained in the branch view `rel2br` that map to the ***`toFiles`***`//depot/rel2/headers/...`
`p4 integ -b rel2br -r //depot/rel2/README`	Branch or merge ***`fromFile`***`//depot/rel2/README` from its ***`toFile`*** as mapped through the branch view `rel2br`.

Related Commands

To create or edit a branch mapping	`p4 branch`

"Syntax" on page 19

To view a list of existing branch mappings	`p4 branches`
To view a list of integrations that have already been performed and submitted	`p4 integrated`
To propagate changes from one file to another after opening files with **`p4 integrate`**	`p4 resolve`
To view a history of all integrations performed on a particular file	`p4 filelog`

p4 integrated

Show integrations that have been submitted.

"Syntax" on page 19

```
p4 [g-opts] integrated [-b branchname [-r]] [-s change] [--into-only] [file ...]
```

Description

The **`p4 integrated`** command shows the integration history of the selected files, and uses this format:

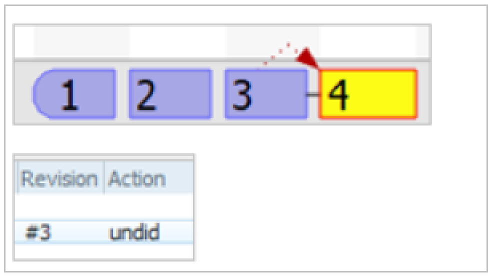
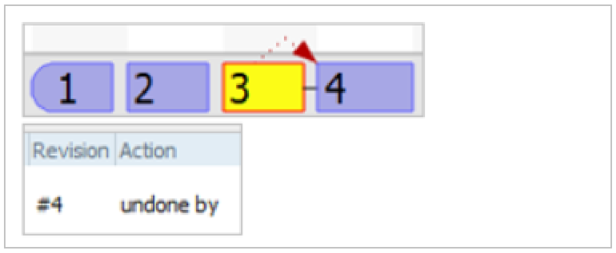
`file#revision-range - integrate-action partner-file#revision-range`

where:

- ***`file`*** is the file argument provided to **`p4 integrated`**
- ***`partner-file`*** is the file it was integrated from or into
- ***`integrate-action`*** describes what the user did during the **`p4 resolve`** process, and is one of the following:

Integrate Action	Resolve Process
`add into`	***`file`*** was integrated into previously nonexistent ***`partner-file`***, and ***`partner-file`*** was reopened for **`add`** before submission.
`add from`	***`file`*** was integrated from a deleted ***`partner-file`***, and ***`partner-file`*** was reopened for **`add`** (that is, someone restored a deleted file by syncing back to a pre-deleted revision and adding the file).
`branch from`	***`file`*** did not previously exist; it was created as a copy of ***`partner-file`***.
`branch into`	***`partner-file`*** did not previously exist; it was created as a copy of ***`file`***.
`merge from`	***`file`*** was integrated from ***`partner-file`***, accepting ***`merge`***.

Integrate Action	Resolve Process
`merge into`	***`file`*** was integrated into ***`partner-file`***, accepting ***`merge`***.
`moved from`	***`file`*** was integrated from ***`partner-file`***, accepting ***`theirs`*** and deleting the original.
`moved into`	***`file`*** was integrated into ***`partner-file`***, accepting ***`theirs`*** and creating ***`partner-file`*** if it did not previously exist.
`copy from`	***`file`*** was integrated from ***`partner-file`***, accepting ***`theirs`***.
`copy into`	***`file`*** was integrated into ***`partner-file`***, accepting ***`theirs`***.
`ignored`	***`file`*** was integrated from ***`partner-file`***, accepting ***`yours`***.
`ignored by`	***`file`*** was integrated into ***`partner-file`***, accepting ***`yours`***.
`delete from`	***`file`*** was integrated from ***`partner-file`***, and ***`partner-file`*** had been previously deleted.
`delete into`	***`file`*** was integrated into ***`partner-file`***, and ***`file`*** had been previously deleted.
`edit from`	***`file`*** was integrated from ***`partner-file`***, and ***`file`*** was edited within the `p4 resolve` process. This allows you to determine whether the change should ever be integrated back; automated changes (`merge from`) needn't be, but original user edits (`edit from`) performed during the resolve should be.
`edit into`	***`file`*** was integrated into ***`partner-file`***, and ***`partner-file`*** was reopened for `edit` before submission.

Integrate Action	Resolve Process
`undid`	the resulting revision has "undid" the edited changes (content and history) from the previously submitted revision(s). Revision #4 "undid" Revision #3:  Revision / Action: #3 undid This is one of the two actions of "p4 undo" on page 596.
`undone by`	the previous revision(s) have been "undone by" the resulting revision, known as the "undone" revision. Revision #4 was "undone" by Revision #3:  Revision / Action: #4 undone by This is one of the two actions of "p4 undo" on page 596.

Note
`Add w/ Edit`, `Merge w/ Edit` and `Undone w/Edit` are possible, but best practice is only one action per changelist.

If a file ***`toFile`*** was ever integrated from a file ***`fromFile`***, and both ***`toFile`*** and ***`fromFile`*** match the `p4 integrated`***`filepattern`*** argument, each integrated action is listed twice in the `p4 integrated` output: once in its ***`from`*** form, and once in its ***`into`*** form, as described above.

If the optional `-b` ***`branch`*** option is used, only files integrated from the source to target files in the branch view are shown.

If the optional `-r` option is provided, the mappings in the branch view are reversed. This option requires the use of the `-b` ***`branch`*** option.

"Syntax" on page 19

Tip
To make it easier to show where a change has been integrated to, consider using the `-s` and `--into-only` options.

Options

`-b` ***branchname***	Lists only files integrated from the source to target files in the branch view. Qualified files are listed, even if they were integrated without using the branch view.
`-r`	Reverses the mappings in the branch view, swapping the target files and source files. The `-b` branch flag is required.
`-s` ***change***	Shows integrations from the specified change forward toward the present.
`--into-only`	Shows only integrations from this path into other paths, not integrations from other paths into this one, or changes ignored by the target path.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`

- When using tagged output with **`p4 integrated`** be warned that the `toFile` and `fromFile` values will be the opposite of how they are presented in the non-tagged output. The following example shows how the output varies:

```
$ p4 integrated //depot/main/revisions.h
//depot/main/revisions.h#1 - add into //depot/dev/revisions.h#1

$ p4 -ztag integrated //depot/main/revisions.h
... toFile //depot/main/revisions.h
... fromFile //depot/dev/revisions.h
... startToRev #none
... endToRev #1
... startFromRev #none
... endFromRev #1
... how add into
... change 12345
```

See "Global options" on page 690 for information on how to enable tagged output.

Related Commands

To see a list of integrations that have not yet been resolved	**`p4 resolve`** **-n**
To view a list of integrations that have been resolved but not yet submitted	**`p4 resolved`**
To perform an integration	**`p4 integrate`**
To view the actions taken for all revisions of a particular file (including all the files from which that particular file was integrated)	`p4 filelog` **`[-i]`** ***`file`***

p4 interchanges

Report changes not yet integrated.

"Syntax" on page 19

```
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user]
                                 fromFileSpec[revSpec] toFile
```

```
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] -b branchname
                                 [toFileSpec[revSpec] ...]
```

```
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] -b branchname
-s
                                 fromFileSpec[revSpec] [toFileSpec]
```

```
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] -S stream [-P
parent]
                                 [FileSpec[revSpec]] [toFileSpec]
```

Description

The **`p4 interchanges`** command lists changes that have not been integrated from a set of source files to a set of target files. The command also reports changes that consist solely of ignored integrations if those changes have not yet been integrated into the target.

Options

`-b branchname`	Use the source and target as defined by the specified branch specification.
`-b branchname -s fromFile[revSpec] [toFileSpec]`	Preview bidirectional integrations (used by Helix server applications; see `p4 integrate` for details.)
`-f`	List files that require integration. For partially integrated changelists, files might be listed even if they were integrated individually.

`-F`	Used with `-S`, ignores a stream's expected flow. It can also force it to generate a branch view based on a virtual stream; the mapping itself refers to the underlying real stream.
`-l`	Long form: include full text of the changelist description.
`-r`	Reverse source and target (that is, reverse the direction of the integration).
`-S` *`stream`* `[-P` *`parent`*`]`	Display integrations pending between the stream and its parent. To treat another stream as the parent, specify `-P`.
`-t`	Display full date and time that changelist was submitted. By default, only the date is displayed.
`-u` *`user`*	Limit results to those submitted by the specified user.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

p4 istat

Check for integrations needed for a stream.

"Syntax" on page 19

```
p4 [g-opts] istat [-a | -r ] [-c | -C] -s] stream
```

Description

Check for integrations that are needed with respect to the parent stream. (Primarily for Helix server applications that check this status in order to render it in human-readable format.)

In a distributed environment, this command is run directly against an edge server. It is not forwarded to the commit server.

Options

`-a`	Check for all integrations, to and from the parent stream
`-c`	Clear cached information before checking integration history. Intended for diagnostic use.
`-C`	Same as -c but also clears the stream's record of the highest merged changelist.
`-r`	Check for integrations required from the parent stream.
`-s`	Display the status of a stream and generate cache data without executing database queries.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`

- The `-c` option is intended for diagnostic and cache consistency checks associated with P4V, the Helix Visual Client.

"Syntax" on page 19

Related Commands

To display changes/sync status for the current workspace.	`p4 cstat`

p4 job

Create or edit an instance of a job, such as a defect or enhancement request.

"Syntax" on page 19

```
p4 [g-opts] job [-f] [jobName]
p4 [g-opts] job -d jobName
p4 [g-opts] job -o [jobName]
p4 [g-opts] job -i [-f]
```

Description

A *job* is a written-language description of work that needs to be performed on files in the depot. It might be a description of a bug (for instance, "the scroll mechanism is not working correctly") or an enhancement request (for instance, "please add a flag that forces a certain operation to occur") or anything else requiring a change to some files under Helix server control.

Jobs are similar to changelist descriptions in that they both describe changes to the system as arbitrary text, but whereas changelist descriptions describe completed work, jobs tell developers what work needs to be done.

Jobs are created and edited in forms displayed by **`p4 job`**. The user enters the textual description of the job into the form, along with information such as the severity of the bug, the developer to whom the bug is assigned, and so on. Because the Helix server superuser can change the fields in the job form with **`p4 jobspec`**, the fields that make up a job may vary from one Helix server installation to another.

When **`p4 job`** is called with no arguments, a new job named **`job`*`NNNNNN`*** is created, where ***NNNNNN*** is a sequential six-digit number. You can change the job's name within the form before quitting the editor. If **`p4 job`** is called with a ***`jobname`*** argument, a job of that name is created; if that job already exists, it is edited.

Once a job has been created, you can link the job to the changelist(s) that fix the job with **`p4 fix`**, **`p4 change`**, or **`p4 submit`**. When a job is linked to a changelist, under most circumstances the job's status is set to **`closed`**. (See "Usage Notes" on the next page for more information).

Form Fields

These are the fields as found in the default job form. Because the fields that describe a job can be changed by the Helix server superuser, the form you see at your site may vary.

Field Name	Type	Description
`Job:`	Writable	The job's name. For a new job, this is `new`. When the form is closed, this is replaced with the name `job`***`NNNNNN`***, where ***`NNNNNN`*** is the next six-digit number in the job numbering sequence. You can change the text in this field. Be aware of the "Limitations on characters in filenames and entities" on page 699.
`Status:`	Writable Value	The value of this field must be `open`, `closed`, or `suspended`. When the job is linked to a changelist, the value of this field is set to `closed` when the changelist is submitted.
`User:`	Writable	The name of the user who created the job.
`Date:`	Writable	The date the job was modified.
`Description:`	Writable	An arbitrary text description of the job.

Options

`-d jobname`	Delete job ***`jobname`***, but only if it has no associated pending or submitted fixes.
`-f`	Force option. Allows Helix server administrators to edit read-only fields.
`-i`	Read the job form from standard input without invoking an editor.
`-o`	Write the job form to standard output without invoking an editor.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`

- If the Helix server superuser has eliminated field ID# `102` (the `Status:` field) with `p4 jobspec`, Helix server is unable to close jobs when the changelists to which they are linked are submitted. See the `p4 jobspec` command and Customizing Helix server: job specifications in the *Helix Core Server Administrator Guide* .
- After a job has been created or changed, Helix server indexes the job so that `p4 jobs` **-e** can locate the job quickly. The index keys are ***`word`***, ***`fieldname`*** where ***`word`*** is a case-

insensitive alphanumeric word. Values in date fields are stored as the number of seconds since January 1, 1970, 00:00:00.

Examples

`p4 job`	Create a new job; by default, its name is of the form **`job`*`NNNNNN`***.
`p4 job job000135`	Edit job **`job000135`**.

p4 jobs

List jobs known to the Helix Core server.

"Syntax" on page 19

```
p4 [g-opts] jobs [-e jobview] [-i -l -r] [-m max] [file[rev] ...]
p4 jobs -R
```

Description

When called without any arguments, **`p4 jobs`** lists all jobs stored in Helix server. You can limit the output of the command by specifying various criteria with options and arguments. If you specify a file pattern, the jobs listed will be limited to those linked to changelists affecting particular files. The **`-e`** option can be used to further limit the listed jobs to jobs containing certain words.

Jobs are listed in alphanumeric order (or, if you use the **`-r`** option, in reverse alphanumeric order) by name, one job per line. The format of each line is:

```
jobname on date by user *status* description
```

The *description* is limited to the first 31 characters, unless the **`-l`** (long) option is used.

If any of the **`date`**, **`user`**, **`status`**, or **`description`** fields have been removed by the Helix server superuser with **`p4 jobspec`**, the corresponding value will be missing from each job's output.

To limit the list of jobs to those that have been fixed by changelists that affected particular files, use **`p4 jobs`** ***`filespec`***. The files or file patterns provided can contain revision specifiers or a revision range.

Options

`-e` ***`jobview`***	List only those jobs that match the criteria specified by *jobview*. See "Job Views" on the next page.
`-i` ***`files`*** **`...`**	Include jobs fixed by changelists that affect files integrated into the named files.
`-l`	Output the full description of each job.
`-m` ***`max`***	Include only the first ***`max`*** jobs, sorted alphanumerically. If used with the **`-r`** option, the last ***`max`*** jobs are included.
`-r`	Display jobs in reverse alphabetical order by job name.

`-R`	Rebuild the job table and re-index each job. Re-indexing the table is necessary either when upgrading from version 98.2 or earlier, or when upgrading from 99.1 to 2001.1 or higher and you wish to search your body of existing jobs for strings containing punctuation.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

Job Views

Use `p4 jobs -e` *`jobview`* to limit the list of jobs to those that contain particular words. You can specify that the search terms be matched only in particular fields, or anywhere in the text of the job. You can use job views to match jobs by values in date fields, though there are fewer options for dates than there are for text. Job fields of type `bulk` are not indexed for searching.

Text matching is case-insensitive. All alphanumeric strings (including words including embedded punctuation) separated by whitespace are indexed as words.

The job view `'`*`word1 word2 ... wordN`*`'` can be used to find jobs that contain the complete set of *word1* through *wordN*.

Examples:

- `p4 jobs -e "ReportedDate=2018/09/14 OwnedBy=mgaria FixVerifiedBy=jsmith"`
- `p4 jobs -e "ReportedDate=2018/09/14 mgaria jsmith"`
- `p4 jobs -e "25* path QA closed job004* 2014* limit*"`

To find jobs that contain all of the terms (boolean AND), separate the terms with the ampersand (`&`) or a space character.

To find jobs that contain any of the terms (boolean OR), separate the terms with the “`|`” character.

Boolean operator precedence	
highest	whatever is in parentheses `()`
	`&` for AND
	`|` for OR
lowest	space for AND

Search results can be narrowed by matching values within specific fields with the job view syntax "***fieldname=value***". The ***value*** must be a single token, including both alphanumeric characters and punctuation.

The wildcard "`*`" allows for partial word matches. The job view "***fieldname=string****" matches "`string`", "`stringy`", "`stringlike`", and so on.

Date fields can be matched by expressing the job view date as ***yyyy/mm/dd*** or ***yyyy/mm/dd:hh:mm:ss***. If a specific time is not provided, the equality operator (`=`) matches the entire day.

The usual comparison operators (`=`, `>`, `<`, `>=`, and `<=`) are available.

Additionally, you can use the NOT operator (`^`) to negate the sense of some comparisons. (See Limitations below for details).

Regular expression matching is supported by the regular expression matching operator (`~=`).

To search for words containing characters that are job search expression operators, escape the characters with a backslash (`\`) character. To match the backslash character, escape it with an additional backlash (`\\`).

The behavior of these operators depends on the type of job field you're comparing against:

Field Type	Use of Comparison Operators in Job Views
`word`	The equality operator (`=`) must match the value in the word field exactly. The relational operators perform comparisons in ASCII order.
`text`	The equality operator (`=`) matches the job if the word given as the value is found anywhere in the specified field. The relational operators are of limited use here, because they match the job if *any* word in the specified field matches the provided value. For example, if a job has a `text` field `ShortDescription` that contains only the phrase `gui bug`, and the job view is "`ShortDesc<filter`", the job matches the job view, because `bug<filter`.
`line`	As for field type `text`, above.
`select`	The equality operator (`=`) matches a job if the value of the named field is the specified word. The relational operators perform comparisons in ASCII order.
`date`	Dates are matched chronologically. If a specific time is not provided, the operators `=`, `<=`, and `>=` match the entire day.

If you're not sure of a field's type, run `p4 jobspec` **-o**, which outputs the job specification used at your site. The `p4 jobspec` field called `Fields:` contains the job fields' names and datatypes. See `p4 jobspec` for a discussion of the different field types.

"Syntax" on page 19

Other Usage Notes

- The `p4 user` form has a `JobView:` field that allows a job view to be linked to a particular user. After a user enters a job view into this field, any changelists he creates automatically list jobs that match the job view in this field. The jobs that are fixed by the changelist can be left in the form, and the jobs that are not should be deleted.
- `p4 jobs` sorts its output alphanumerically by job name, which also happens to be the chronological order in which the jobs were entered. If you use job names other than the standard Helix server names, this ordering may not help much.
- The `-m` ***`max`*** `-r` construct displays the last ***`max`*** jobs in alphanumeric order, not the ***`max`*** most recent jobs, but if you are using the default Helix server job naming scheme (jobs numbered like `job001394`), alphanumeric job order is identical to order by entry date.
- You can use the `*` wildcard to determine if a text field contains a value or not by checking for the job view "***`field`***`=*`"; any non-null value for ***`field`*** matches.
- When querying for jobs using the `-e` ***`jobview`*** option, be aware of your operating system and command shell's behavior for parsing, quoting, and escaping special characters, particularly when using wildcards, logical operators, and parentheses.

Limitations

- Job views cannot be used to search for jobs containing null-valued fields. In other words, if a field has been deleted from an existing job, then the field is not indexed, and there is no job view that matches this "deleted field" value.
- The job view NOT operator (`^`) can be used only after an AND within the job view. Thus, the job views "`gui ^name=joe`" and "`gui&^name=joe`" are valid, while the job views "`gui|^name=joe`" and "`^name=joe`" are not.
- The `*` wildcard is a useful way of getting around both of these limitations.

  For instance, to obtain all jobs without the string "`unwanted`", query for '`job=* ^unwanted`". All jobs will be selected by the first portion of the job view and logically ANDed with all jobs NOT containing the string "unwanted".

  Likewise, because the job view "***`field`***`=*`" matches any *non*-null value for ***`field`***, (and the `job` field can be assumed not to be null), you can search for jobs with null-valued fields with "`job=* ^`***`field`***`=*`"
- You cannot currently search on space-delimited fields with conditionals. For example, instead of using `p4 jobs -e "field=word1 word2"`, you must use `p4 jobs -e "field=word1 field=word2"`.

"Syntax" on page 19

Examples

`p4 jobs //depot/proj/file#1`	List all jobs attached to changelists that include revisions of `//depot/proj/file`.
`p4 jobs -i //depot/proj/file`	List all jobs attached to changelists that include revisions of `//depot/proj/file` or revisions of files that were integrated into `//depot/proj/file`.
`p4 jobs -e gui`	List all jobs that contain the word `gui` in any field.
`p4 jobs -e "gui Submitted-By=joe"`	List all jobs that contain the word `gui` in any field and the word `joe` in the `Submitted-By:` field.
`p4 jobs -e "gui ^Submitted-By=joe"`	List all jobs that contain the word `gui` in any field and any value *other than* `joe` in the `Submitted-By:` field.
`p4 jobs -e "window*"`	List all jobs containing the word "`window`", "`window.c`", "`Windows`", in any field. The quotation marks are used to prevent the local shell from expanding the "`*`" on the command line.
`p4 jobs -e window.c`	List all jobs referring to `window.c` in any field.
`p4 jobs -e "job=* ^unwanted"`	List all jobs not containing the word `unwanted` in any field.
`p4 jobs -e "(fast|quick)&date>1998/03/14"`	List all jobs that contain the word `fast` or `quick` in any field, and have a `date:` field pointing to a date on or after `3/14/98`.
`p4 jobs -e "fast|quick" //depot/proj/...`	List all jobs that have the word `fast` or `quick` in any field, and that are linked to changelists that affected files under `//depot/proj`.

Related Commands

To create or edit an existing job	`p4 job`
To attach a job to a particular changelist, indicating that the job is fixed by that changelist	`p4 fix`
To list all jobs and changelists that have been linked together	`p4 fixes`

To view all the information about a particular changelist, including the jobs linked to the changelist	`p4 describe`
To change the format of the jobs used at your site (superuser only)	`p4 jobspec`
To read information about the format of jobs used on your site (any user)	`p4 jobspec` -o
To set a default job view that includes jobs matching the job view in all new changelists	`p4 user`

p4 jobspec

Edit the jobs template to change, add, or remove spec fields for job forms.

"Syntax" on page 19

```
p4 [g-opts] jobspec
p4 [g-opts] jobspec [-i]
p4 [g-opts] jobspec -o
```

Description

The **`p4 jobspec`** command presents the Helix server administrator with a form in which job fields can be edited, created, deleted, and refined.

"Syntax" on page 19

Form Fields

"Syntax" on page 19

Field Name	Description
`Fields:`	A list of field definitions for your site's jobs, one field per line. Each line is of the form *code name datatype length fieldtype*. **Important** When adding a new field in 2019.2 or later, the administrator can enter its code by using the optional placeholder value `NNN`. If the administrator choses this option, the server will assign an appropriate value. Upon saving the jobspec, the next available code value in the `101-119` range will be automatically generated and saved. The values `101` through `105` are reserved for Helix Core, so the effective range is currently `106-199`. If the number of fields exhausts the range, new codes are assigned unique values greater than or equal to `10000`. Alternatively, the administrator can specify a numeric value. When the administrator saves the spec, a message might indicate that the specified value is not available. ■ *code*: a unique integer that identifies the field internally to Helix server. ■ *name*: the name of the field. This can be changed at any time, while the code should not change once jobs have been created. Field names must not contain spaces. ■ *datatype*: the datatype of the field. Possible values are: • `word`: a single arbitrary word (a string with no spaces) • `date`: a date/time field • `select`: one of a fixed set of words • `line`: one line of text • `text`: a block of text, starting on the line underneath the fieldname. • `bulk`: like `text`, but not indexed for searching with `p4 jobs` **-e**. ■ *length*: recommended length for display boxes in GUI clients accessing this field. Use a value of `0` to let a Helix server application choose its own value. ■ *fieldtype*: does the field have a default value? Is it required? Is it read-only? Possible values are: • `optional`: field can take any value or be erased. • `default`: a default value is provided; it can be changed or erased. • `required`: a default value is provided; it can be changed but the user must enter a value.

Field Name	Description
	• `once`: read-only; the field value is set once to a default value and is never changed. • `always`: read-only; the field's value is set to a new default when the job is edited. This is useful only with the `$now` and `$user` variables; it allows you to change the date a job was modified and the name of the modifying user.
`Values:`	Contains a lists of fields and valid values for `select` fields. Enter one line for each field of datatype `select`. Each line must contain the fieldname, a space, and the list of acceptable values separated by slashes. For example: `JobType bug/request/problem`.
`Presets:`	Contains a list of fields and their default values for each field that has a fieldtype of `default`, `required`, `once`, or `always`. Each line must contain the field name and the default value, separated by a space. For example: `JobType bug` Any one-line string can be used, or one of three built-in variables: ■ `$user`: the user who created the job ■ `$now`: the current date ■ `$blank`: the phrase `<enter description here>` When users enter jobs, any fields in your jobspec with a preset of `$blank` must be filled in by the user before the job is added to the system. See "Usage Notes" on the facing page for special considerations for field 102.
`Comments:`	Textual comments that appear at the top of each `p4 job` form. Each line must begin with the comment character `#`. See "Usage Notes" on the facing page for special considerations for these comments if your users need to enter jobs through P4V, the Helix Visual Client.

Options

`-i`	Read the jobspec form from standard input.
`-o`	Write the jobspec form to standard output.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`admin`, or `list` to use the `-o` option

- Do not attempt to change, rename, or redefine fields 101 through 105. These fields are used by Helix server and should not be deleted or changed. Only use `p4 jobspec` to add new fields to your jobs.

  Field 101 is required by Helix server and cannot be renamed or deleted.

  Fields 102 through 105 are reserved for use by Helix server applications. Although it is possible to rename or delete these fields, it is highly undesirable to do so. Helix server applications might continue to set the value of field 102 (the `Status:` field) to `closed` (or some other value defined in the `Presets:` for field 102) upon changelist submission, even if the administrator has redefined field 102 to for use as a field that does not contain `closed` as a permissible value, leading to unpredictable and confusing results.
- The information in the `Comments:` fields is the only information available to your users to tell them how to fill in the job form, and is also used by P4V, the Helix Visual Client, to display ToolTips.
- The `Presets:` entry for the job status field (field 102) has a special syntax for providing a default fix status for `p4 fix`, `p4 change` -s, and `p4 submit` -s.

  By default, a job's status is set to `closed` after you use `p4 fix`, `p4 change`, or `p4 submit`. To change the default fix status from `closed` to some other *`fixStatus`* (assuming that you have defined the *`fixStatus`* as a valid `select` setting in the `Values:` field), use the special syntax of *`jobStatus`*`,fix/`*`fixStatus`* in the `Presets:` field for field 102 (job status). To change the behavior of `p4 fix`, `p4 change`, and `p4 submit` to leave job status unchanged, use the special *`fixStatus`* of `same`.
- See the example of a customized jobspec in the "Customizing Perforce: Job Specifications" chapter of the *Helix Core Server Administrator Guide*.

Related Commands

To create, edit, or view a job	`p4 job`
To attach a job to a changelist	`p4 fix`
To list jobs	`p4 jobs`
To list jobs attached to specific changelists or changelists attached to specific jobs	`p4 fixes`

p4 journalcopy

Copies journal data from a master server to the local file system of a standby replica.

"Syntax" on page 19

```
p4 [g-opts] journalcopy -l
p4 [g-opts] journalcopy -i N [-b wait]
```

Description

The **`p4 journalcopy`** command has two syntax variants:

- To get a report of the current copy position from the master's journal to the replica's journalcopy, use **`p4 journalcopy -l`**

  where the output of the **`p4 journalcopy -l`** command includes the sequence number, which indicates the byte offset position in the journal that the journalcopy has reached:

  ```
  Current replica persisted journal state is: Journal 2,
  Sequence 6510347
  ```

- To copy journal data (the journalcopy) to the local file system of a standby replica, use **`p4 journalcopy -i`** ***`N`*** with or without the **`-b`** option.

An operator or superuser can confirm the state of a replica by running the **`p4 journalcopy -l`**, `"p4 pull" on page 420` **`-l -j`**, and **`p4 pull -l -s`** commands.

Options

`-b` ***`wait`***	Wait the specified number of seconds before retrying the **`p4 journalcopy`** command after a failed attempt. Setting this option overrides the default value of `60` seconds. For example, to set the value of the startup.*N* configurable so that the *wait* time is `30` seconds: `$ p4 -p depot_master_p4port configure set standby#startup.1="journalcopy -i 0 -b 30"`

`-i N`	Repeat the **`p4 journalcopy`** command every ***N*** seconds. ▪ If you do not use this option, the command runs once. ▪ If you set *N* to `0`, the command runs as frequently as possible rather than sleeping between runs. This is useful for a high availability (HA) standby server.
`-l`	Report the current position in the copied journal.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
NA	NA	`super`

Examples

`p4 journalcopy -l`	Show information about the current copy position from the master's journal to the replica's journal.

Related Commands

To make a copy of the master's versioned files.	`p4 pull -u`
To retrieve the journal records from the journalcopy files created by the **`p4 journalcopy`** and to apply these to the standby's database	`p4 pull -L`
To get information about replication status from the point of view of the master server, use the `-J` option of the `p4 servers` command.	`p4 servers`
To fail over to a new master server	"p4 failover" on page 190

p4 journaldbchecksums

Write journal notes with table checksums.

"Syntax" on page 19

```
p4 [g-opts] journaldbchecksums [-t tableincludelist | -T
tableexcludelist]
                               [-l level]
p4 [g-opts] journaldbchecksums -u filename -t tablename
                               [-v version] [-z]
p4 [g-opts] journaldbchecksums -s -t tablename
                               [-b blocksize] [-v version]
p4 [g-opts] journaldbchecksums -c changelist
```

Description

The **`p4 journaldbchecksums`** command provides a set of tools for ensuring data integrity across a distributed or replicated installation.

The Perforce service automatically performs an integrity check whenever you use the `"p4 admin" on page 58` **`checkpoint`** or **`p4 admin journal`** commands, or when you use **`p4 journaldbchecksums`** to manually perform an integrity check.

To use this command, structured logging (see **`p4 logparse`**) must be enabled, and at least one structured log must be capturing events of type **`integrity`**.

When an integrity check is performed, the Perforce service writes records to the journal that contains the checksums of the specified tables (or, if no tables are specified, for all tables). Replica servers, upon receiving these records, compare these checksums with those computed against their own database tables, as they would with **`p4 dbstat`**. Results of the comparisons are written in the replica’s log.

You can control which tables are checked, either by including and excluding individual tables with the **`-t`** and **`-T`** options, or by using one of three levels of verification.

Verification levels are controlled by the `"rpl.checksum.auto" on page 793` configurable or the **`-l`** ***`level`*** option.

- Level 1 corresponds to the most important system and revision tables.
- Level 2 includes all of level 1 as well as certain metadata that is not expected to differ between replicas.

- Level 3 includes all metadata, including metadata that is likely to differ between replicas, particularly build farms and edge servers.

When checking individual changelists and individual tables, the `"rpl.checksum.change" on page 794` and the `"rpl.checksum.table" on page 794` configurables control when events are written to the log.

For more information, including a list of database tables associated with each level of verification, see Verifying replica integrity in the *Helix Core Server Administrator Guide*.

Options

`-b` ***`blocksize`***	When scanning tables, scan *blocksize* records per block. The default is 5,000. For each block, the server computes a block checksum and writes it as a journal note. Replica servers automatically verify these blocks when processing these notes. This option can be used with large tables on a production system as the table is unlocked between each block. Inspecting the results of the block verifications will reveal the location of damage that affects only part of a database table. See the example of "Database Table Block Checksums" on page 300.
`-c` ***`changelist`***	Compute a checksum for an individual submitted changelist. The checksum is written as a journal note, and replica servers automatically verify the checksum of the change when they process these notes. See the example of "Changelist Checksums" on page 299.
`-l` ***`level`***	Specify a level for checksumming; each level corresponds to a larger set of tables. These levels correspond to the levels used by the **`rpl.checksum.auto`** configurable.
`-s -t` ***`tablename`***	Scan the specified database table. See the example of "Database Table Unload" on page 300.
`-t` ***`tables`***	Specify the table(s) for which to compute checksums. To specify multiple tables, double-quote the list and separate the table names with spaces. The table names must start with "**`db.`**". Table names can also be separated by commas. See the example of "Database Table Checksums" on the facing page.
`-T` ***`tableexcludelist`***	Compute checksums for all tables except those listed.
`-u` ***`filename`*****`-t`** ***`tablename`***	Unload the specified database table to a file. This command also writes a journal note that documents this action, and instructs replica servers to automatically unload the same table to the same file when processing these notes.

`-v` ***`version`***	When unloading or scanning tables, specify the server version number to use. If no server version number is specified, the current server version is used.
`-z`	Compress the file when unloading a table.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`operator` `super`

For more about administering Perforce in distributed or replicated environments, see *Helix Core Server Administrator Guide*.

Examples

Database Table Checksums

`p4 journaldbchecksums [-t tableincludelist | -T tableexcludelist] [-l N]`

causes the server to write journal notes containing table checksum information:

```
p4 journaldbchecksums -t db.rev
@nx@ 12 1487712216 @41@ 9 -933920831 0 4 0 @db.rev@ @@ @@ @@ @@
```

Edge/Replica servers automatically verify the table checksums when processing these notes, writing the results to the server log and optionally an integrity structured log if configured:

```
Table db.rev checksums match. 2017/02/21 13:23:36 version 9: expected
0xC8557FC1, actual 0xC8557FC1
"p4 logparse" on page 356 -m1 -F f_table=db.rev -T 'f_date f_results'
integrity.csv
... f_date 2017/02/21 13:23:36 219149298
... f_results match
```

The results of table checksum comparison will result in one three possible results:

- `match`
- `DIFFER`
- `empty`

```
Table db.have checksums DIFFER. 2017/02/21 13:08:38 version 3: expected
0x3BB210EE, actual 0xB1BF3E83

p4 logparse -F f_results=DIFFER -T 'f_date f_table' integrity.csv
... f_date 2017/02/21 13:08:38 203821071
... f_table db.have
```

```
Table db.ldap checksums empty. 2017/02/24 11:33:54
version 0: expected 0x0, actual 0x0.

p4 logparse -F f_results=empty -T f_table integrity.csv
... f_table db.ldap
```

The table checksums might be reported as `DIFFER` if the database structure diverged as the result of:

Software upgrades:	Some upgrades are performed against a database table when data in a table is accessed.
Replaying a checkpoint or journal:	When administrators replay journal data or journal patches using `p4d -jr`, the transactions replayed into the database are not journaled. This might generate differing checksums. When replaying journal data in a distributed environment, always use `p4d -s -jr` so the replayed transactions are journaled, thus enabling downstream edge/replica servers to replay them. Be aware that `p4d -jr` run against a replica server only updates the replica's database files and therefore might generate differing checksums.
Journal filtering:	When filtering is active in your replication process, not all journal checksums are expected to match.

To remedy unexpected checksum differences, restore the edge/replica server database from a new checkpoint on the commit/master server.

Changelist Checksums

`p4 journaldbchecksums -c change`

causes the server to compute a checksum of an individual submitted changelist. This checksum is written as a journal note:

```
p4 journaldbchecksums -c 12073
@nx@ 15 1487961638 @41@ 12073 1 0 0 0 @46B19358420B468668781A002BA0AC15@
@@ @@ @@ @@
```

Replica servers automatically verify the checksum of the change when processing these notes and write the results to the integrity structured log:

```
p4 logparse -F f_change=12073 -T f_results integrity.csv
... f_results match
```

Server behavior is dependent on the "rpl.checksum.change" on page 794 configurable.

Database Table Block Checksums

`p4 journaldbchecksums -s -t tablename [ -b blocksize ][-v N]`

causes the server to scan the specified database table. The table is scanned in blocks. The number of records in a block is specified by the `-b` flag, which defaults to `5,000`. For each block, the server computes a block checksum and writes it as a journal note:

```
p4 journaldbchecksums -s -t db.have
@nx@ 17 1487964567 @41@ 3 1 313 0 0 @db.have@
@@@//Talkhouse/build/jar/Talkhouse.jar@@
 @ @@@//Jam/MAIN/src/glob.c@@ @ @2BCDA450287C03DE3433AEB6278EA4AA@ @@
```

Replica servers automatically verify these blocks when processing these notes and write output to the integrity structured log if configured:

```
p4 logparse -F 'f_table=db.have' -T 'f_results f_checkSum f_
checkSum2' integrity.csv
... f_checkSum 2BCDA450287C03DE3433AEB6278EA4AA
... f_checkSum2 D41D8CD98F00B204E9800998ECF8427E
... f_results failed
```

This command can be used with large tables on a production system because the table is unlocked between each block. Inspecting the results of the block verifications reveals the location of any damage, which affects only part of a database table.

Database Table Unload

`p4 journaldbchecksums -u filename -t tablename [-v N] [-z]`

causes the server to unload the specified database table to the specified file. The command also writes a journal note describing this action:

```
p4 journaldbchecksums -u working.txt -t db.working
@nx@ 16 1487964861 @41@ 10 0 0 0 0 @db.working@ @working.txt@ @@ @@ @@
```

Replica servers automatically unload the same table to the same file when processing these notes. If only a file name is specified with `-u`, as in the example above, the unload files are created in the "P4ROOT" on page 680 directory of both servers. Any relative path specified with `-u` is relative to the "P4ROOT" on page 680 directory. Absolute paths to the unload file can also be used. Ensure any referenced directory paths exists on both master and replica prior to running the unload.

For a time-consistent comparison of the contents of the table, unload the tables in this way. This command is recommended only for tables that are small. The `-z` flag specifies that the file should be compressed.

p4 journals

Display history of checkpoint and journal activity for the server.

"Syntax" on page 19

```
p4 [g-opts] journals [-F filter] [-T fields] [-m max]
```

Description

Use the **`p4 journals`** command to display information from the **`db.ckphist`** table, which holds historical information about checkpoint and journal activity. A server uses this table to record the following checkpoint and journal events:

- the server takes a checkpoint
- the server rotates a journal
- the server replays a journal
- a replica schedules a checkpoint

Each server in a multi-server installation has its own, unique **`db.ckphist`** table. That is, the table is not replicated to replicas. This table is not part of the main server database; it's not journaled, and it does not need to be backed up. It is not included in checkpoints. If anything goes wrong, it can be thrown away.

Here's an example of the output from **`p4 journals`**.

```
mbp-jbujes:~ jbujes$ p4 -p qaplay:20141 journals
... start 1381278576
... startDate 2013/10/08 17:29:36
... end 1381278576
... endDate 2013/10/08 17:29:36
... pid 19960
... type checkpoint
... flags
... jnum 19
... jfile checkpoint.19
... jdate 1381278576
... jdateDate 2013/10/08 17:29:36
... jdigest E4EB1FF5B589D05E9F5A8EE1F8183A86
... jsize 27183115
... jtype text
... failed 0
... errmsg

... start 1381278576
... startDate 2013/10/08 17:29:36
... end 1381278576
... endDate 2013/10/08 17:29:36
... pid 19960
... type checkpoint
... flags
... jnum 18
... jfile journal.18
... jdate 1381278575
... jdateDate 2013/10/08 17:29:35
... jdigest 00000000000000000000000000000000
... jsize 15737
... jtype text
... failed 1
... errmsg

... start 1374629669
... startDate 2013/07/23 18:34:29
... end 1374629669
... endDate 2013/07/23 18:34:29
... pid 14700
... type replay
... flags -r . -j r
... jnum -1
... jfile basis.ckp
... jdate 1366076427
... jdateDate 2013/04/15 18:40:27
... jdigest 00000000000000000000000000000000
... jsize 27181640
... jtype text
... failed 1
... errmsg
```

This command displays full error message text for a failed checkpoint.

Use the global **-F** option to format the output from the **p4 journals**; for example:

p4 -F "%jfile% %jnum%" journals -F type=checkpoint

The meaning of each field is described in the following table. Output entries are listed from newest event to oldest event.

start	Starting Unix timestamp of the command that ran. See type field to determine which command was executed.
startDate	Human-readable form of start value.
end	Ending Unix timestamp of the command that ran.
endDate	Human-readable form of end value.
pid	The process id of the command whose execution produced this record. This value can be useful in searching for related entries in other logs.
type	Indicates the command whose execution produced this record. Types include the following: ■ **journal**: refers to the **p4 -J** command or the **p4 admin journal** command. ■ **checkpoint**: refers to the **p4 -jc** command or **p4 admin checkpoint** command. ■ **replay**: refers to the **p4d -jr** command ■ **schedule**: refers to taking a checkpoint on a replicated server. This produces two records: one for the checkpoint and another for the schedule.
flags	Flags passed to the command implied by the type filed.
jnum	The checkpoint number. A value of -1 indicates that the journal number is unknown.
jfile	The name of a journal or checkpoint file that was input to the command implied by the type field.
jdate	Unix timestamp when the filed specified by **jfile** was created.
jdateDate	Human-readable format of **jdate** value.
jdigest	The digest of the checkpoint file.
jsize	The size of the file specified by **jfile**.
jtype	The type of the file specified by **jtype**.

Options

`-F` *`filter`*	List only the records that satisfy the filter expression. For instructions on constructing the filter expression, see "Job Views" on page 284.
`-m` *`max`*	Limit output to the specified number of records.
`-T` *`fields`*	Limit output to the specified fields. Separate fields using a comma or a space.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`superuser` or `operator`

Related Commands

To create a checkpoint.	`p4 admin checkpoint`
To create a journal.	`p4 admin journal`

p4 key

Display, set, or delete a key/value pair.

"Syntax" on page 19

```
p4 [g-opts] key name
p4 [g-opts] key [-v] name value
p4 [g-opts] key [-d] name
p4 [g-opts] key [-i -v] name
p4 [g-opts] key [-m] [pair list]
p4 [g-opts] key --from=oldvalue --to=newvalue name
```

Description

Keys allow you to store name-value pairs for use in scripts. These user-managed keys are stored in a table named `db.nameval`.

The command includes the following variants:

- `p4 key name` returns the value of key *name*.
- `p4 key name value` sets the value of the key *name* to *value*, and if *name* does not already exist, it is created.

  Specifying `-v` displays the previous value of the specified key after the key has been set or incremented.
- `p4 key -d name` deletes the specified key.
- `p4 key -i name` increments a numeric key.

  Specifying `-v` displays the previous value of the specified key after the key has been set or incremented.
- `p4 key [-m] pair list` defines multiple set and delete operations to be performed. Each operation is defined by a value pair in the pair list. To set a key, use a name and value, to delete a key, use a `-` (hyphen) followed by the name. See "Examples" on the next page.

  This variant is useful in multi-server environments where running individual commands is likely to introduce unwanted latency.
- `p4 key --from oldvalue --to newvalue` sets the specified key to the new value if the current value of the key is *oldvalue*. A key that is not set or that has been deleted cannot be set using this syntax variant.

If a key does not exist, its value is returned as zero. Key names are not stored until set to a nonzero value.

For the minimal access level required to display and set keys, see "Usage Notes" below and "Examples" below.

Options

`-d` ***name***	Delete key *name* from the Perforce service.
`-i` ***name***	Increment key *name* by 1 and return the new value. This option can only be used with numeric keys.
`-m` ***namevalue*** `...`	Perform multiple key value operations in one command. See "Examples" below.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list` to display a key's value; (`admin` if `dm.keys.hide` is set to 2) `review` to set a new value

Examples

`p4 key` ***mykey*** `12`	Set the value of ***mykey*** to `12`. If ***mykey*** does not exist, it is created. If ***mykey*** does exist, its value is changed to the newly-specified value. Requires `review` access.
`p4 key` ***mykey***	Display the value of ***mykey***. If ***mykey*** does not exist, its value is displayed as `0`. Requires `list` access.

`p4 key -m` ***`mykey`*** `5` ***`mynewkey`*** `4`	Set two keys. Requires **`review`** access.
`p4 key -m -` ***`mykey`*** `-` ***`mynewkey`***	Delete two keys. Requires **`review`** access.
`p4 key -m` ***`mykey`*** `6 -` ***`mynewkey`***	Set one key; delete one key. Requires **`review`** access.
`p4 key --from=4 --to=6 mykey`	Set a different value for the key if the **`--from`** value is correct.

Related Commands

To list all keys and their values	**`p4 keys`**

p4 keys

Display list of known key/value pairs.

"Syntax" on page 19

```
p4 [g-opts] keys [-e nameFilter] [-m max]
```

Description

The Helix Core server holds a user-accessible store of key/value pairs. These user-managed keys are stored in a table named `db.nameval`.

If the `dm.keys.hide` configurable is set to 1 or 2, `admin` access is required.

`p4 keys` provides the current list of keys, along with their values.

Options

`-e` *`nameFilter`*	List keys with a name that matches the *`nameFilter`* pattern, for example `p4 keys -e 'mycounter-*'`.
`-m max`	List only the first *`max`* keys.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list` (`admin` if `dm.keys.hide` is set to 1 or 2)

Related Commands

To view or change the value of a key	`p4 key`

p4 label

Create or edit a label specification and its view.

"Syntax" on page 19

```
p4 [g-opts] label [-f -g] [-t template] labelname
p4 [g-opts] label -d [-f -g] labelname
p4 [g-opts] label -o [-t template] labelname
p4 [g-opts] label -i [-f -g]
```

Description

Use `p4 label` to create a new label specification or edit an existing label specification. A ***`labelname`*** is required.

Running `p4 label` allows you to configure the mapping that controls the set of files that are allowed to be included in the label. After configuring the label, use `p4 labelsync` or `p4 tag` to tag files with the label.

Labels can be either automatic or static. Automatic labels refer to the revisions provided in the `View:` and `Revision:` fields. Static labels refer only to those specific revisions tagged by the label by means of either the `p4 labelsync` or `p4 tag` commands.

Only the `Owner:` of an `unlocked` label can use `p4 labelsync` or `p4 tag` to tag files with that label. The owner of a group may be a single user or a group.

Warning
A branch, depot, label, and workspace may not share the same name.

Automatic labels

Automatic labels refer to the revisions provided in the `View:` and `Revision:` fields of the label specification. To create an automatic label, fill in the `Revision:` field of the `p4 label` spec with a revision specifier. When you sync a workspace to an automatic label, the contents of the `Revision:` field are applied to every file in the `View:` field.

Example **Using an automatic label as an alias for a changelist number**

Bruno is running a nightly build process, and has successfully built a product as of changelist 1234. Rather than having to remember the specific changelist for every night's build, he types `p4 label nightly20111201` and uses the label's `Revision:` field to automatically tag all files as of changelist 1234 with the `nightly20111201` label:

```
Label:  nightly20111201
Owner:  bruno
Description:
        Nightly build process.
Options:     unlocked noautoreload
View:
        //depot/...
Revision:
        @1234
```

The advantage to this approach is that it is highly amenable to scripting, takes up very little space in the label table, and provides a way to easily refer to a nightly build without remembering which changelist number was associated with the night's build process.

Example **Referring specifically to the set of files submitted in a single changelist**

A bug was fixed by means of changelist 1238, and requires a patch label that refers to only those files associated with the fix. Bruno types **`p4 label patch20111201`** and uses the label's **`Revision:`** field to automatically tag only those files submitted in changelist 1238 with the **`patch20111201`** label:

```
Label:  patch20111201
Owner:  bruno
Description:
        Patch to 2011/12/01 nightly build.
Options:     unlocked noautoreload
View:
        //depot/...
Revision:
        @1238,1238
```

This automatic label refers only to those files submitted in changelist 1238.

Example **Referring to the first revision of every file over multiple changelists**

You can use revision specifiers other than changelist specifiers. In this example, Bruno specifies to the first revision (#1) of every file in a branch. Depending on how the branch was populated, these files could have been created through multiple changelists over a long period of time:

```
Label:  first2.2
Owner:  bruno
Description:
```

"Syntax" on page 19

```
        The first revision in the 2.2 branch
Options:     unlocked noautoreload
View:
        //JamCode/release/jam/2.2/src/...
Revision:
        "#1"
```

Because Helix server forms use the `#` character as a comment indicator, Bruno has placed quotation marks around the `#` to ensure that it is parsed as a revision specifier.

Form Fields

Field Name	Type	Description
`Label:`	Read-only	The label name as provided in the invoking command. Be aware of the "Limitations on characters in filenames and entities" on page 699.
`Owner:`	Writable, optional	The label's owner. By default, the user who created the label. Only the owner of a label can update which files are tagged with the label. The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Update:`	Read-only	The date the label specification was last modified.
`Access:`	Read-only	The date and time the label was last accessed, either by running `p4 labelsync` on the label, or by otherwise referring to a file with the label revision specifier `@label`. (Note: Reloading a label with `p4 reload` does not affect the access time.)
`Description:`	Writable, optional	An optional description of the label's purpose.

Field Name	Type	Description
`Options:`	Writable	Options to control behavior and storage location of labels ▪ `locked` or `unlocked`. If the label is `locked`, the list of files tagged with the label cannot be changed with `p4 labelsync`. ▪ `autoreload` or `noautoreload`. For static labels, if `noautoreload` is set, the label is stored in `db.label`, and if `autoreload` is set, it is stored in the unload depot. This option is ignored for automatic labels. Storing labels in the unload depot can improve performance on sites that make extremely heavy use of labels.
`Revision:`	Writable	An optional revision specification for an automatic label. If you use the `#` character to specify a revision number, you must use quotes around it in order to ensure that the `#` is parsed as a revision specifier, and not as a comment field in the form.
`View:`	Writable	A list of depot files that can be tagged with this label. No files are actually tagged until `p4 labelsync` is invoked. Unlike client views or branch views, which map one set of files to another, label views consist of a simple list of depot files. See "Views" on page 702 for more information.
`ServerID:`	Writable, optional	If set, restricts usage of the label to the named server. If unset, this label may be used on any server.

Options

`-d` [`-f`]	Delete the named label if it's `unlocked`. The `-f` option forces the deletion even if the label is `locked`. (Deleting a `locked` label requires `admin` or `super` access.)
`-f`	Allow the `Update:` field's date to be set. Can be used with either the `-i` option or the `-t` option for the same purpose.
`-g`	In multi-server environments, use the `-g` option to control whether the label is local to an edge server, or globally available from the commit server.
`-i`	Read the label definition from standard input without invoking the editor.
`-o`	Write the label definition to standard output without invoking the editor.

`-t` *`template`*	Copy label *`template`*'s view and options into the `View:` and `Options:` fields of this label. You can specify a default label template using the `template.label` configure variable. If you do so, you do not have to specify this option.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`

- To create an automatic label, fill in the `Revision:` field of the `p4 label` form with a revision specifier. When you sync a workspace to an automatic label, the contents of the `Revision:` field are applied to every file in the `View:` field.
- With a multi-server Perforce service, labels may be local or global. Local labels are restricted to a single edge server, and cannot be used on other servers. Global labels are created and updated on the commit server, and are visible to all servers. However, global labels may only be used with global (unbound) client workspaces.

Local Label	Global Label
`"rpl.labels.global" on page 800` is unset, which is `0`	`"rpl.labels.global" on page 800` set to `1`
by default, labels are **local** to your edge server	by default, labels are **global**
`-g` option provides access to **global** labels on the commit server	`-g` option allows the updating of **local** labels If a label exists on an edge server before `rpl.labels.global` is set, and you want that label to be available on both the edge server and the commit server, unloaded the label from the edge server and reload it on the commit server

Examples

`p4 files @`*`labelname`*	List the file revisions tagged by *`labelname`*.

"Syntax" on page 19

Related Commands

To synchronize a label with the client workspace.	"p4 labelsync" on page 319
To list all labels known to the system	"p4 labels" on page 316
To create a label and tag files with the label	"p4 tag" on page 584

p4 labels

Display the list of defined labels.

"Syntax" on page 19

```
p4 [g-opts] labels [-t] [-u user | --me] [[-e|-E] filter] [-m
max]
                       [FileSpec[revSpec]]
p4 [g-opts] labels [-t] [-u user | --me] [[-e|-E] filter] [-m
max]
                       [-a | -s serverID]
p4 [g-opts] labels -U
```

Description

p4 labels lists all the labels known to the Perforce service in the form:

```
Label labelname date description
```

Use the **-t** option to display the time of the last update to the label.

```
Label labelname date time description
```

To see a list of loaded static labels that tag specific files, specify a file pattern, with an optional revision range. (Because automatic labels refer to all files in the label view at a specified revision range, automatic labels are not shown when you use **p4 labels** with a file pattern.)

Use the **-m *max*** option to limit the output to the first ***max*** labels.

Use the **-e** or **-E *filter*** options to limit the output to labels whose name matches the ***filter*** pattern. The **-e** option is case-sensitive, and **-E** is case-insensitive.

Use the **-u *user*** option to limit the output to labels owned by the named user.

Options

-a	List all labels, not just labels bound to this server. This option may not be used with a file specification.
-e *filter*	List only labels matching ***filter*** (case-sensitive).

`-E` ***`filter`***	List only labels matching ***`filter`*** (case-insensitive).
`-m` ***`max`***	List only the first ***`max`*** labels.
`-s` ***`serverID`***	List only those labels bound to the specified ***`serverID`***. This option may not be used with a file specification.
`-t`	Display the time as well as the date of the last update to the label.
`-u` ***`user`***	List only labels owned by ***`user`***.
`--me`	Equivalent to `-u $P4USER`.
`-U`	List labels in the unload depot. For details, see `p4 unload`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- To see a list of files tagged by a particular label, use `p4 files@`***`labelname`***.
- In a multi-server environment, users connected to an edge server receive only those labels that are bound to their edge server, unless they explicitly request otherwise by specifying the `-a` or `-s` ***`serverID`*** options.

Examples

To list all labels in the system	`p4 labels`
To list all labels that contain any revision of `file.c`	`p4 labels file.c`
To list only labels containing revisions `#3` through `#5` of `file.c`	`p4 labels file.c#3,5`

Related Commands

To create a label and tag files with the label	"p4 tag" on page 584
To create or edit a label specification	"p4 label" on page 310

To add, delete, or change the files included in a label	"p4 labelsync" on page 319
To view a list of files included in a label	"p4 files" on page 203 @Labelname

p4 labelsync

Synchronize a label with the contents of the current client workspace.

"Syntax" on page 19

```
p4 [g-opts] labelsync [-a -d -g -n -q] -l labelname [[FileSpec]
[revSpec]]
```

Description

p4 labelsync causes the named label to reflect the current contents of the client workspace by tagging the last revision of each file synced into the workspace with the label name. The label name can subsequently be used in a revision specification as `@label` to refer to the revision of the file that was tagged with the label.

Without a file argument, **p4 labelsync** causes the label to reflect the contents of the client workspace by adding, deleting, and updating the set of files tagged with the label.

- If a file is given, **p4 labelsync** updates the tag for only that named file.
- If the file argument includes a revision specification, the client view is ignored. The specified revision is used instead of the revision existing in the workspace.
- If the file argument includes a revision range, only the highest revision in that range is used.

Only the `Owner:` of an `unlocked` label can use **p4 labelsync** to tag files with that label.

A label that has its `Options:` field set to `locked` cannot be updated with **p4 labelsync**.

Options

`-a`	Add the label to files that match the file pattern arguments; no files are deleted from the label.
`-d`	Delete the label tag from the named files.
`-g`	In multi-server environments, use the `-g` option to specify whether the label being applied is local to an edge server, or is globally available from the commit server.
`-l` ***labelname***	Specify the label to be applied to file revisions.

`-n`	Display what `p4 labelsync` would do without actually performing the operation.
`-q`	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`open`

- By default, `p4 labelsync` operates on the revisions of files last synced to your client workspace. To tag the head revisions of files (or the highest revision in a specified range), use `p4 tag`.
- To see which files are tagged by the label, use `p4 files @`*`labelname`*.
- With a multi-server Perforce service, `p4 labelsync` works with a label local to the edge server (to which you are sending a request). The `-g` option can be used to apply a global label, but only with an unbound (global) client workspace.

  By default, labels are local to your edge server, and you use the `-g` option to access global labels on the commit server. If your administrator has set `rpc.labels.global` to `1`, labels are global by default, and the meaning of the `-g` option is inverted to allow updating of local labels.

Related Commands

To create or edit a label	`p4 label`
To list all labels known to the system	`p4 labels`
To create a label and tag files with the label	`p4 tag`

p4 ldap

Create, edit, or delete an LDAP configuration specification, or test an existing LDAP configuration.

This command cannot be run from a read-only or build replica.

"Syntax" on page 19

```
p4 [gopts] ldap configname
p4 [gopts] ldap -i
p4 [gopts] ldap -o configname
p4 [gopts] ldap -d configname
p4 [gopts] ldap -t username configname
```

Description

The **`p4 ldap`** command includes five syntax variants:

- The first variant allows you to create or edit an LDAP configuration.
- The **`p4 ldap -i`** command allows you to read an LDAP configuration from standard input.
- The **`p4 ldap -o`** command allows you to display the specified LDAP configuration.
- The **`p4 ldap -d`** command allows you to delete the specified LDAP configuration.
- The **`p4 ldap -t`** command allows you to test an existing LDAP configuration.

Creating an LDAP Configuration

The LDAP configuration you create with the **`p4 ldap`** command defines an Active Directory or other LDAP server against which the Helix servercan authenticate users.

To create an LDAP configuration specification, you provide values that specify the host and port of the AD/LDAP server, bind method information, and security parameters. Bind methods can be one of the following:

- **Simple**: Uses a template based on the user’s name to produce a distinguished name that the Helix server attempts to bind against, validating the user’s password. For example:

  ```
  uid=%user%,ou=users,dc=example,doc=org
  ```

- **Search**: Uses an LDAP search query to locate the user record. The search relies on a known base DN and an LDAP search query. You provide these using the `SearchBaseDN`, `SearchFilter`, and `SearchScope` fields of the LDAP configuration specification. This method might also required the full distinguished name and password of a known read-only entity in the directory. You supply these using the `SearchBindDN` and `SearchPasswd` fields of the LDAP configuration. Here is a sample search query:

  ```
  BaseDN: ou=users,dc=example,dc=org
  LDAP query: (uid=%user%)
  ```

- **SASL**: If the AD/LDAP server supports `SASL DIGEST-MD5`, this method defers the user search to the AD/LDAP server and does not require a distinguished name to be discovered before the bind is attempted. The user provides a user name, a password, and an optional realm.

In addition to creating the LDAP configuration, you must use the following configurables to enable the configuration and to further define the authentication process:

- `auth.ldap.order.N` - enables an AD/LDAP server and specifies the order in which it should be searched.
- `auth.default.method` - specifies whether new users should be authenticated by Helix server or using LDAP.
- `auth.ldap.userautocreate` - specifies whether new users should be automatically created on login when using LDAP authentication.
- `auth.ldap.timeout` - time to wait before giving up on a connection.
- `auth.ldap.cafile` - the path to a file used for certification when the AD/LDAP server uses SSL or TLS.
- `auth.ldap.ssllevel` - level of SSL certificate validation.

For more information, see "Configurables" on page 715.

Note
LDAP configurations are stored in the new `db.ldap` table. This table is journaled, so LDAP configurations are now included in checkpoints and are replicated.

Authentication is user-based:

- The LDAP authentication method is selected for each existing user with the `AuthMethod` field of the user specification. For more information, see the `p4 user` command.
- The authentication method applied to auto-created users (LDAP or Perforce) is determined by the `auth.userautocreate` configurable. For more information, see "Configurables" on page 715.

Here is a sample LDAP configuration:

```
Name:       olivia
Host:       openldap.example.com
```

```
Port:     389
Encryption:    tls
BindMethod:    search
Options: nodowncase nogetattrs norealminusername
SimplePattern: someuserid
SearchBaseDN:  ou=employees,dc=example,dc=com
SearchFilter:  (cn=%user%)
SearchScope:    subtree
GroupSearchScope:  subtree
```

Testing an LDAP Configuration

You can use a command like the following to test an LDAP configuration:

```
$ p4 ldap -t userX myConfig
```

The command prompts you for a password and returns successfully if `userX` can be found. If the AD/LDAP server specified by `myConfig` is down, if the user can't be found, or if the password you supply is incorrect, the command returns a detailed error message. For example:

```
c:\temp> p4 -p 1666 ldap -t userX olivia
Enter password:
Authentication as cn=userX,ou=employees,dc=example,dc=com
failed. Reason: Invalid Credentials
```

Form Fields

Field Name	Type	Description
`Name:`	Read only	The name of the LDAP configuration. Relevant to bind method: all
`Host:`	Writable	Fully qualified domain name of AD/LDAP server. The default is `localhost`. Relevant to bind method: all

Field Name	Type	Description
`Port:`	Writable	The port to connect on. The default is `389`. Relevant to bind method: all **Tip** Port 389 has historically been used for unencrypted connections into an LDAP server. Port 636 is used for legacy SSL connections. Port 389 is used for TLS connections; TLS establishes a non encrypted connection on port 389 that it 'upgrades' to an encrypted TLS connection as the initial connection proceeds. This allows unencrypted and encrypted connections to be setup and handled by this one port. The Perforce LDAP specification must therefore have the specified 'Port:' field corresponding to the relevant encryption method in the 'Encryption:' ('none', 'ssl' or 'tls') field.
`Encryption:`	Writable	One of `none`, `ssl`, and `tls`. The default is `tls`. Relevant to bind method: all
`BindMethod:`	Writable	One of `simple`, `search`, and `sasl`. See "Creating an LDAP Configuration" on page 321 above for more details. Relevant to bind method: all

Field Name	Type	Description
`Options`	Writable	Modifies the behavior of the LDAP integration that is specific to this configuration. Choose from the following: ■ `[no]downcase` specifies whether `p4 ldapsync -g` should downcase user names from the directory. For example, if user names in LDAP are ABrown, and SMITH, they are added to the group as `abrown` and `smith`. ■ `[no]getattrs` specifies whether the `Fullname` and `Email` fields for users auto created with `p4 login` should be populated from the directory. This requires that you set the `AttributeName` and/or `AttributeEmail` fields in the ldap spec. See below. ■ `[no]realminusername` specifies whether the realm should be taken from the SASL username if it is in UNC or UPN format. That is, if your user names look like this: `user@realm` or `realm\user`, the user and realm are separated and passed separately. By default, these options are not set.
`SimplePattern:`	Writable	The distinguished name used to bind against to validate the user's credentials. The `%user%` placeholder is replaced with the user's `userId`. Relevant to bind method: simple
`SearchBaseDN:`	Writable	The distinguished name from which to start the search for the user object. Relevant to bind method: search
`SearchFilter:`	Writable	The LDAP query filter that identifies the user object to bind against. The `%user%` placeholder is replaced with the user's `userId`. Relevant to bind method: search

Field Name	Type	Description
`SearchScope:`	Writable	One of the following: ■ `baseonly` - search just the `BaseDN` object. ■ `children` - search the `BaseDN` object and its direct children. ■ `subtree` - search the `BaseDN` object and all objects below it. Relevant to bind method: search
`SearchBindDN:`	Writable	The distinguished name to bind against to search the directory. For example, `CN=bruno, DC=foo, DC=com` Relevant to bind method: search
`SearchPasswd:`	Writable	The password for the `BindDN` record. You may quote this field; this allows special characters, like `#` to be used in the password. Relevant to bind method: search
`SaslRealm:`	Writable	The optional realm to use when authenticating the user using SASL. Relevant to bind method: sasl
`GroupSearchFilter:`	Writable	The filter to use for the group search. Relevant to bind method: all
`GroupBaseDN:`	Writable	The search base for performing a group search. The default is the value of `SearchBaseDN`. Relevant to bind method: all
`GroupSearchScope`	Writable	One of the following, to be used when performing a group search. ■ `baseonly` - search just the `BaseDN` object. ■ `children` - search the `BaseDN` object and its direct children. ■ `subtree` - search the `BaseDN` object and all objects below it. Relevant to bind method: all

Field Name	Type	Description
`AttributeUid`	Writable	The name of the attribute in the user object that contains the user's UID.
`AttributeName`	Writable	The name(s) of the attribute(s) in the user object that contains the user's full name. If multiple attributes are required to form the full name, specify each one surrounded by % symbols, so that expanding these forms the user's full name. The `getattrs` option must be enabled for this field to be populated using the value specified in the LDAP `AttributeName` field.
`AttributeEmail`	Writable	The name of the attribute in the directory's user object that contains the users' email addresses. The `getattrs` option must be enabled for this field to be populated using the value in the LDAP `AttributeEmail` field.

Options

`-d` *`config`*	Deletes the specified LDAP configuration.
`-i`	Read the LDAP specification from standard input.
`-o` *`config`*	Writes the specified LDAP configuration to standard output.
`-t` *`username`* *`config`*	Specifies a username to authenticate against the specified LDAP configuration. It is provided for testing purposes. The command returns a success message or a detailed error message. You do not have to enable the configuration to run this test.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

"Syntax" on page 19

Examples

`p4 ldap myLdap`	Create the `myLdap` configuration.
`p4 ldap -o myLdap`	Write the `myLdap` configuration to standard output.
`p4 ldap -t bruno myLdap`	Authenticate the user `bruno` against the server specified by the `myLdap` configuration.
`p4 ldap -d myLdap`	Delete the `myLdap` configuration.

Related Commands

To view a list of all LDAP configurations.	`p4 ldaps`
To define LDAP-related configurables.	`p4 configure`

p4 ldaps

Display a list of LDAP configurations or attempt to authenticate a user against active configurations.

"Syntax" on page 19

```
p4 [g-opts] ldaps [-A]
p4 [g-opts] ldaps -t username
```

Description

The **`p4 ldaps`** command includes two syntax variants:

- The first variant allows you to display existing LDAP configurations; the **`-A`** option lists active configurations according to the priority set for them with the **`auth.ldap.order.n`** configurable.
- The second variant allows you to attempt to authenticate the specified user against all active configurations. This command tests each configuration whether the authentication succeeds or fails. That is, testing does not stop with the first successful authentication.

Listing configurations

If you do not use the **`-A`** option, **`p4 ldaps`** returns information about all configurations. If a configuration has not been assigned a priority using the **`auth.ldap.order.n`** configurable, it is shown to be disabled. Output includes the configuration name, the host and port of the AD/LDAP server, the bind method used, and whether the server is enabled.

```
c: \temp>p4 -p 1666 ldaps
emma localhost:389 simple (disabled)
olivia localhost:389 sasl (enabled)
isabel localhost:389 search (enabled)
```

If you use the **`-A`** option, only enabled servers are shown, and they are listed in the order in which they will be searched. For example:

```
c: \temp>p4 -p 1666 ldaps -A
olivia localhost:389 search (enabled)
isabel localhost:389 sasl (enabled)
```

The order of the servers shown above are determined by the setting of the **`auth.ldap.order.n`** configurable; for example:

"Syntax" on page 19

```
c: \temp>p4 -p 1666 configure show
auth.ldap.order.1=olivia (configure)
auth.ldap.order.2=isabel (configure)
```

Testing active configurations

Using the `-t` option allows you to test all active configurations. A test might fail because a server is unavailable, because the user could not be found, or because the wrong credentials were submitted.

- Here is output from a successful authentication:

  ```
  c:\temp> p4 -p 1666 ldaps -t myUser
  Enter password:
  Testing authentication against LDAP configuration olivia
  Authentication successful
  Testing authentication against LDAP configuration isabel.
  Authentication successful
  ```

- Here is output from a test that failed because the AD/LDAP servers were unavailable:

  ```
  c:\temp> p4 -p 1666 ldaps -t myUser
  Enter password:
  Testing authentication against LDAP configuration olivia.
  Failed to initialize TLS: Server Down
  Testing authentication against LDAP configuration isabel.
  Failed to initialize TLS: Server Down
  ```

- Here is output when a bad password is given:

  ```
  c:\temp> p4 -p 1666 ldaps -t myUser
  Enter password:
  Testing authentication against LDAP configuration isabel
  Authentication as abrown failed. Reason: Invalid Credentials
  Testing authentication against LDAP configuration olivia
  Authentication as abrown failed. Reason: Invalid Credentials
  ```

Options

`-A` *`config`*	Display command output according to the priority set with the `auth.ldap.order.`*`n`* configurable. This limits the configurations displayed to those that have been assigned a priority. If you omit this option, all active configurations are listed in alphabetical order.
`-t` *`username`*	Specifies a user name to authenticate against all active LDAP configurations; this option is provided for testing purposes.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Examples

`p4 ldaps`	List all configurations.
`p4 ldaps -t bruno`	Authenticate the user `bruno` against all active configuration.

Related Commands

To create, edit, delete, or test an LDAP configuration.	`p4 ldap`
To define LDAP-related configurables.	`p4 configure`

p4 ldapsync

Synchronize Helix server users and group memberships with LDAP groups.

"Syntax" on page 19

```
p4 [gopts] ldapsync -g [-n] [-i N] [group ...]
p4 [gopts] ldapsync -u [ -c -U -d ] [ -n ] [ -i N] [ ldap ... ]
```

Description

When run with the `-g` option specified, this command updates the users lists in Helix server groups to match the lists of members in LDAP groups.

Tip
Any users that are not Active Directory members are removed.

If one or more group names are provided, only those groups are updated. If no groups are provided, all groups with LDAP configurations are updated.

When run with the `-u` option specified, this command updates the Helix server users to match those in the LDAP. This works by querying each LDAP server defined by the LDAP specifications passed in the arguments. The LDAP specification's `SearchFilter` is used to query the LDAP server with the `%user%` placeholder expanded to `*` in order to identify all LDAP users. The three `Attribute*` fields are used to map LDAP result to the Helix server user's username, full name and email address. All provided LDAP specifications are queried to build a full, combined list of LDAP users before any changes to the Helix server users are made.

Note
`p4 ldapsync` requires `super` access granted by `p4 protect`.

To keep users or groups with LDAP configurations in sync with their LDAP counterparts, `p4 ldapsync` can be set as a startup command that runs in the background. See the final example in the "Examples" on page 334 section.

The user synchronization has three actions that must be enabled separately by specifying the appropriate flags:

To create new users found in the LDAP servers that do not yet exist in Helix server	use the `-c` option

To update full name and email address of any existing Helix server users found in the LDAP servers	use the `-U` option
To delete Helix server users not found in any of the LDAP servers	use the `-d` option

Tip
You can track the activity of "p4 ldapsync" on the previous page. See `ldapsync.csv` at "p4 logparse" on page 356.

Options

`-u`	Allows users to be created, updated, or deleted based on users found in LDAP servers. This works by querying each LDAP server defined by the LDAP specifications passed in the arguments. The LDAP specification's SearchFilter is used to query the LDAP server with the `%user%` placeholder expanded to `*` to identify all LDAP users. The three `Attribute*` fields are used to map LDAP result to the Perforce: ■ user's username ■ full name ■ email address All provided LDAP specifications are queried to build a full, combined list of LDAP users before any changes to the Perforce users are made. Note: The usernames of members added to a Perforce group by `p4 ldapsync` can be normalised into lowercase by setting the `downcase` option in the LDAP spec.
`-c`	Creates any new users found in the LDAP servers that do not yet exist in Helix server. The `AuthMethod` will be set to `ldap` and `Type` set to `standard`.
`-d`	Deletes any Helix server users not found in the LDAP servers, provided that the user is of `Typestandard` and `AuthMethod` is `ldap`.
`-g`	Required to specify groups. Updates the users lists in Perforce groups to match the lists of members in LDAP groups. If one or more group names are provided, only those groups are updated. If no groups are provided, then all groups with LDAP configurations will be updated.
`-i` ***N***	Automatically repeats the command every `N` seconds. If this option is not specified, the command executes once and exits.
`-n`	Preview the operation and show the users or groups that would be affected without taking any action.

group	The name of a Helix server group that must be updated when changes to the corresponding LDAP group take place. If no group names are specified, all groups with LDAP configurations are updated.
`-U`	Updates the full name and email address of any existing Helix server users found in the LDAP servers, provided that: ■ the user is of `Type` standard ■ the `AuthMethod` is `ldap` ■ the values differ For a detailed walkthrough, see the Support Knowledgebase article, "Configuring ldapsync".

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Examples

To update the groups for which LDAP configurations have been defined:

```
p4 ldapsync -g
```

To configure a start up command that updates the groups every 30 minutes:

```
p4 configure set "myServer#startup.1=ldapsync -g -i 1800"
```

Note
This example assumes you have set `serverID` (see "p4 serverid" on page 505) to the server where you want to set startup.*n*, which is one of the Configurables.

Related Commands

To view a list of all LDAP configurations	`p4 ldaps`
To create or edit an LDAP configuration	`p4 ldap`
To define LDAP-related configurables	`p4 configure`
To define LDAP configurations for a Helix server group spec	`p4 group`

p4 license

Update or display the license file.

"Syntax" on page 19

```
p4 [g-opts] license -o
p4 [g-opts] license -i
p4 [g-opts] license -u
```

Description

The **`p4 license`** command allows Helix server superusers to update or display the Helix server license file. This command requires that there is already a valid license file in the Helix server root directory.

Use **`p4 license`** to add licensed users to a Perforce service without having to shut down the service and manually copy the license file into the server root.

Note
Most new license files obtained from Perforce can be installed with **`p4 license`**, or by copying over the existing license file. However, if the server IP address or port number has changed, you must explicitly "stop" the server:

For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

For Windows, see Starting and stopping the Helix Server.

Important
License expiration occurs at the START of the expiration date according to Coordinated Universal Time (UTC). For example, if **`p4 license -o`** shows the UNIX time for expiration as 1546300801, that means Tuesday, January 1, 2019 12:00:01 AM for UTC (GMT). If your server is in California, **the license expires the day before** on Monday, December 31, 2018 4:00:01 PM because California local time is 8 hours behind UTC. To convert UNIX time to a more readable format, use a converter, such as https://www.epochconverter.com/.

Alternatively, to get **`licenseTimeRemaining`** in seconds, use **`p4 license -u`** as in this example:

```
>p4 -u su-bruno -p p4prod.mycompany.com:1666 license -u
... isLicensed yes
... userCount 651
... userLimit 1000
... clientCount -
... clientLimit unlimited
... fileCount -
... fileLimit unlimited
... repoCount -
... repoLimit unlimited
... licenseExpires 1582934400
... licenseTimeRemaining 5172009
... supportExpires 1582934400
```

Tip
If you want a warning that the license will expire within a specified amount of time, consider the script at https://swarm.workshop.perforce.com/files/guest/nick_poole/scripts/nagios/readme.txt.

Limits for unlicensed use depend on the release

2016.1 and later	prior to 2016.1
■ unlimited number of files for 5 users and 20 client workspaces, or ■ unlimited number of users and workspaces for up to 1,000 files	■ unlimited number of files for 20 users and 20 client workspaces, or ■ unlimited number of users and workspaces for up to 1,000 files

Options

`-o`	Display the current license file on the standard output.
`-i`	Read in a new license file from the standard input. For example, see Adding or updating the license file.
`-u`	Report license limits, including the expiration of the license and of Support, and show how many entities (users or files) are in use with respect to these limits.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	**`super`** (**`admin`** for **`-u`** option)

Examples

`p4 license -o`	Display the current license file on the standard output.
`p4 license -i`	Read in a new license file from the standard input.

p4 list

Create a temporary list of files that can be used as a label.

"Syntax" on page 19

```
p4 [g-opts] list [-l labelname] [-C -M] FileSpec[revSpec]
p4 [g-opts] list -l labelname -d [-M]
```

Description

This command is intended for use by systems integrators and third-party developers.

p4 list builds an in-memory temporary list of files that can be used as a label for the duration of the single **p4** command session that created it. The list exists only as long as the connected session. The temporary list created by running **p4 list** from the command line is not available to subsequent **p4** commands.

By default, the head revision is listed. If the file argument specifies a revision, all files at that revision are listed. If the `file` argument specifies a revision range, the highest revision in the range is used for each file.

The `-d` option is handy for long-running processes that need to use and reuse lists within the scope of one session without exhausting the server's process memory.

Options

`-C`	Limits any depot paths to those that can be mapped through the client workspace.
`-d` ***labelname***	Delete the specified list.
`-l` ***labelname***	Specify the label to be applied to file revisions. If a label of that name already exists, the in-memory name has precedence over the stored one. If you do not use this option, the **p4 list** command assigns a unique name to the temporary list and returns the name as output.
`-M`	When run against a forwarding replica, forward the **p4 list** command to the master server.
g-opts	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`open`

p4 lock

Lock an opened file against other users submitting changes to the file.

"Syntax" on page 19

```
p4 [g-opts] lock [-c changelist] [file ...]
p4 [g-opts] lock -g -c changelist
```

Description

Locking files prevents other users from submitting changes to those files. If the files are already locked by another user, `p4 lock` fails. When the user who locked a particular file submits the file, the lock is released.

This command is normally called with a specific file argument; if no file argument is provided, all open files in the default changelist are locked. If the `-c` ***changelist*** option is used, all open files matching the given file pattern in changelist *changelist* are locked.

In an edge/commit architecture, use the `-g` flag to lock the files locally and globally. This syntax variant may only be used from an edge server, and it must be used with the `-c` ***changelist*** option. This lock is removed by the `p4 unlock -g` command or by any submit command for the specified changelist.

Options

`-c` ***changelist***	Lock only files included in changelist *changelist*
`-g`	Lock files in an edge/commit architecture. See "Description" above.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`write`

If the server configurable `server.locks.global` is set to `1`, when `p4 lock` is issued on an edge server it takes global locks on the commit server by default.

Related Commands

To unlock locked files	`p4 unlock`
To display all your open, locked files (UNIX)	`p4 opened	grep "*locked*"`

p4 lock (graph)

Lock an opened file to prevent it from being submitted to the repo.

"Syntax" on page 19

```
p4 lock [-c changelist] [file ...]
```

Description

The specified files are locked in the depot, preventing any user other than the current user on the current client from submitting changes to the files.

- If a file is already locked, the lock request is rejected.
- If no file names are specified and no ***changelist*** is specified, all open files are locked.
- If both the `-c` flag and a file specification are provided, only the matching files in the specified changelist are locked.

Options

`-c` ***changelist***	Lock only files included in changelist *changelist*
file ...	Limit the lock to the matching files

p4 lockstat

Report lock status of database tables.

"Syntax" on page 19

```
p4 [g-opts] lockstat [-c client | -C]
```

Description

By default, the `p4 lockstat` command reports any database tables that are currently locked for a read or write operation.

Options

`-c` *`client`*	Report whether or not the specified client workspace is locked for a read or write operation.
`-C`	Report all client workspaces that are locked for read/write operations.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

p4 logappend

Add a line to any user log files.

"Syntax" on page 19

```
p4 [g-opts] logappend -a args...
```

Description

The **p4 logappend** command appends a line to any structured log file that includes user log events. At least one argument is required, and up to 25 arguments may be supplied per line.

Use cases:

- To enable custom or third party tools or scripts report status or error conditions into logs associated with the Helix server. See the Example below.
- To insert a placeholder, observation, or question for later investigation. For example, "Slow server now. Is a large checkin being processed?"

Options

`-a` *args...*	At least one argument is required, and up to 25 arguments to be appended to the `user` log. See the Example below.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- The two log files that include user log events are `user.csv` and `all.csv`.

Example

Suppose you have tool named `mytool` that interacts with Helix Core server to log errors centrally on the Helix server:

```
$ p4 -zprog=mytool logappend -a error "too many links" 123
```

Search the log by using the "p4 logparse" on page 356 command:

```
$ p4 logparse -F 'f_prog=mytool f_args=links' user.csv
```

```
... f_eventtype 10
... f_timestamp 1511883486
... f_timestamp2 174298000
... f_date 2017/11/28 07:38:06 174298000
... f_pid 68580
... f_cmdno 1
... f_user bruno
... f_client bruno_ws
... f_func user-logappend
... f_host 127.0.0.1
... f_prog mytool
... f_version 2017.1.PREP-TEST_ONLY/DARWIN90X86_64/1505513
... f_args -a:error:too many links:123
... f_arg_1 error
... f_arg_2 too many links
... f_arg_3 123
... f_lognum 0
... f_logfile user.csv
... f_offset 1339
```

p4 logger

Report changed jobs and changelists.

"Syntax" on page 19

```
p4 [g-opts] logger [-c sequence#] [-t countername]
```

Description

External programs that call the Helix Core server can use this command.

Options

`-c` *`sequence#`*	List all events happening after this sequence number.
`-t` *`countername`*	List all events after this counter number.
`-c` *`sequence#`* `-t` *`countername`*	Update the supplied counter with the current sequence number and clear the log. This clears the log regardless of which counter name is specified, so only one user can make use of this option.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`review`

- The `p4 logger` command is not for end users. It supports propagation of information to an external defect tracking system.
- In multi-server environments, `p4 logger` commands should be issued to the Commit Server, not to an Edge Server. If you are using P4DTG or other third-party tools that make use of this command, ensure that your installation is properly configured.

"Syntax" on page 19

Related Commands

To list of users who have subscribed to review particular files	`p4 reviews`
To set or read the value of a Helix server counter	`p4 counter`
To see full information about a particular changelist	`p4 describe`
To see a list of all changelists, limited by particular criteria	`p4 changes`

p4 login

Log in to the Perforce service by obtaining a ticket.

"Syntax" on page 19

```
p4 [g-opts] login [-a -p] [-h host][user]
p4 [g-opts] login -s [-a | -h host][user]
p4 [g-opts] login [-a -p] -r remotespec [--remote-user=X]
p4 [g-opts] login [-s] -r remotespec [--remote-user=X]
```

Description

The **`p4 login`** command authenticates a user and creates a ticket that represents a session with Helix server. An authenticated user can access the shared versioning service until the ticket expires or the user issues the **`p4 logout`** command.

By default, tickets are valid for 12 hours. This value is defined on a per-group basis in the **`p4 group`** form.

To obtain a ticket valid for all IP addresses (for instance, to use Helix server simultaneously on more than one workstation), use **`p4 login -a`**. Users with tickets that are valid for all IP addresses still consume only one Helix server license.

Login attempts, whether successful or not, are logged to a structured log file. If the login fails, the reason for failure is included in the log. In the case of authentication triggers or LDAP authentication, the parts of the error message that contain user data are sanitized as needed.

If you use LDAP authentication, you can set the **`getattrs`** option in the ldap spec **`options`** field to specify whether the **`Fullname`** and **`Email`** fields for users created by **`p4 login`** are populated from the directory.

Options

`-a`	Obtain a ticket that is valid for all IP addresses.
`-h` ***`host`***	Request a ticket that is valid for the specified host IP address.
`-p`	Display the ticket, rather than storing it in the local ticket file.

`-s`	Display the status of the current ticket, if one exists. Use with `-a` to display status for all hosts, or with `-h` ***host*** to display status for a specfic host. Users with `super` access can provide a ***username*** argument to display the status of that ***username***'s ticket.
`-r`	Specify the remote server to which the login should be forwarded. If the remote spec passed in contains a `RemoteUser` entry, the login is performed for that user.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- To create tickets that do not expire, set the timeout value to `unlimited` in the `p4 group` form.
- By default, after 3 failed login attempts, a user must wait up to 10 seconds before logging in again. Helix server superusers can change that default of 3 by setting the "dm.user.loginattempts" on page 744 configurable.
- To extend a ticket's lifespan, use `p4 login` while already logged in. Your ticket's lifespan is extended by 1/3 of its initial timeout setting, subject to a maximum of your ticket's initial timeout setting.
- Helix server superusers can obtain login tickets for users other than themselves without entering passwords. Non-superusers who attempt to log in as other users must use the `p4 -u` ***username*** `login` form of the command and correctly supply the other user's password.
- Tickets are stored in the file specified by the `P4TICKETS` environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.
- The `-h` option causes the service to issue a ticket that is valid on the specified host IP address. This option is typically used with `-p` to display a ticket that can subsequently be used on another machine.
- In replicated environments, logging in to the master server does *not* log you in to any replica servers.
- To learn about "p4 login behavior with an auth-check-sso trigger", see Single sign-on and auth-check-sso triggers in the *Helix Core Server Administrator Guide*.

Examples

`p4 login`	Prompt the user for a password. If the password is entered correctly, issue a ticket valid on the user's machine.
`p4 -u builder login -a`	Attempt to log in as user `builder`. If the password is entered correctly, issue a ticket valid on all the client machines of this server.
`p4 login -s bruno`	Get the status of the specified user's ticket. The output might be: `User bruno ticket expires in 430808 hours 17 minutes.` `User bruno on host 10.0.101.22: Ticket: Set`

Related Commands

To end a login session	`p4 logout`
To display tickets	`p4 tickets`

p4 login2

Perform multi-factor authentication (MFA), formerly known as second factor authentication (2fa).

"Syntax" on page 19

```
p4 login2 [ -p -R ] [ -h host ] [ -S state ] [ -m method ] [
username ]
p4 login2 -s [ -a | -h host ] [ username ]
p4 login2 [-p] -r <remotespec> [--remote-user=X]
p4 login2 [-s -a] -r <remotespec> [--remote-user=X]
```

Description

Enables a user requiring multi-factor authentication to authorize access on a given host.

Note
The end-user will not need this command if auto-prompt is enabled.

See Triggering for multi-factor authentication (MFA) in the *Helix Core Server Administrator Guide*.

Options

`-p`	Causes the MFA to persist even after the user's ticket has expired.
`-s`	Display the status of the current ticket, if one exists.
`-R`	Causes the MFA to be restarted, which allows the user to re-request a one-time password.
`-r`	Causes the server to forward the MFA to the server referenced in the the specified remote specification. The authentication will be for the user specified by the `--remote-user` flag, or if `RemoteUser` is set in the remote specification, the login will be for that user. Specifying a host or a username is not allowed when logging into a remote server.
`-s`	Displays the MFA status for the user on the current host, or all hosts that the user has used if the `-a` flag is used. To show the status for a specific host, the IP address can be specified with the `-h` flag.
`username`	Specifying a username as an argument to this command requires `super` access, which is granted by "p4 protect" on page 401. In this case, `p4 login2` skips the MFA process and immediately marks the user as validated for the current host. The super user must already be logged in and verified, if necessary. A host (IP address) can be specified with the `-h` flag to validate the user on a different host.
`-S`	For non-interactive clients, executes each step of the MFA individually. This must begin with the `list-methods` state, which will report the list of available MFA methods for the given user. The next state must be `init-auth`, and must be accompanied by the chose method provided to the `-m` flag. This initiates the authentication with the MFA provider. The final step is `check-auth`, which will either prompt for a one-time password (OTP) or request the authorization status from the MFA provider, depending on the type of authentication method selected. The `-p` flag can be provided at the `init-auth` stage. If a host or user is being specified, the appropriate arguments must be provided at each stage.
`-a`	Shows the MFA status for the user on all hosts.
`-h`	Shows the status for a specific host if the IP address is specified.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Related Commands

To login	"p4 login" on page 349
To end a login session	`p4 logout`
To display tickets	`p4 tickets`

p4 logout

Log out of Helix server by removing or invalidating a ticket.

"Syntax" on page 19

```
p4 [g-opts] logout [-a] [username]
```

Description

Log a user out of Helix server by removing a ticket on the user’s workstation, or by invalidating the ticket on the service.

If you use `p4 logout -a`, the ticket remains in the ticket file, but is invalidated on the service: all users of the ticket are logged out simultaneously. You can also remove a single user’s ticket with the `-a` ***username*** option.

Options

`-a`	Log out all users of the ticket by invalidating the ticket on the service. If a username is specified, that user is logged out. You must have super user access to be able to log out a user other than yourself.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`, `super` to use `-a` ***username***

- Tickets are stored in the file specified by the `P4TICKETS` environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.
- In replicated environments, logging out of the master server with `p4 logout -a` also logs you out of any replica servers.

"Syntax" on page 19

Examples

`p4 logout`	Log out of Helix server by removing the local session ticket.
`p4 logout -a`	Log out of Helix server by removing the local session ticket and instructing the Perforce service to invalidate the ticket on all other workstations from which they were logged in.

Related Commands

To start a login session (to obtain a ticket)	`p4 login`
To display tickets	`p4 tickets`

p4 logparse

Parse a structured log file and return log data.

"Syntax" on page 19

```
p4 [g-opts] logparse [-e] [-T fields...] [-F filter] [-s offset]
[-m max] logfile
```

Description

The **`p4 logparse`** command parses the indicated structured *logfile* and returns the log data in tagged format.

Structured logs differ from the basic error log (**`P4LOG`**) and audit log (**`P4AUDIT`**). To read the basic error log, use the **`p4 logtail`** command.

Valid names for structured log files:

`all.csv`	All loggable events (commands, errors, audit, etc.)
`audit.csv`	Audit events (audit, purge)
`auth.csv`	Information about user login attempts.
`commands.csv`	Command events (command start, command compute, command end)
`errors.csv`	Error events (errors-failed, errors-fatal)
`events.csv`	Server events (startup, shutdown, checkpoint, journal rotation, etc.)
`integrity.csv`	Major events that occur during replica integrity checking.
`ldapsync.csv`	Activity of "p4 ldapsync" on page 332
`route.csv`	Log the full network route of authenticated client connections. Errors related to `"net.mimcheck" on page 779` are also logged against the related hop.
`track.csv`	Command tracking (track-usage, track-rpc, track-db)
`triggers.csv`	Trigger events.
`user.csv`	User events, with one record every time a user runs **`p4 logappend`**

To enable structured logging, set the **`serverlog.file.`*`n`*** configurable(s) to the name of the file. For example:

```
$ p4 configure set serverlog.file.2=commands.csv
$ p4 configure set serverlog.file.3=errors.csv
$ p4 configure set serverlog.file.5=audit.csv
```

Numbers provided for the configurables do not have to be consecutive. A given number cannot exceed 500, so the following assignment returns an error:

```
$ p4 configure set serverlog.file.666=commands.csv
```

Structured log files are automatically rotated on checkpoint, journal creation, overflow of associated `serverlog.maxmb.n` limit (if configured), and the `p4 logrotate` command.

Options

`-e`	Display special characters as hex-encodings.
`-F filter`	Limit output to records that match the filter pattern.
`-m max`	Limit the number of lines returned.
`-s f_offset`	Start parsing at the given file offset as returned in the `f_offset` field.
`-T fields...`	Limit displayed fields to those listed.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Examples

To match the contents of a particular field, use the ***field=word*** syntax. Logical operators `&` (AND), `|` (OR), ^ (not), and `()` (grouping) can

also be used. Spaces are treated as a low-precedence AND operator

OR (`|`) operator to get the event type and date for both user `bruno` and user `admin`:

```
p4 logparse -T 'f_user f_eventtype f_date' -F 'f_user=bruno |
f_user=admin' errors.csv
```

AND (`&`) operator, NOT (`^`) operator to get event type and date while excluding the user `admin`: `p4 logparse -T 'f_user f_eventtype f_date' -F 'f_eventtype=4 & ^f_user=admin ' errors.csv` The `^` operator can be used only in conjunction with the `&` or space operators.
wildcard (*) operator matches anything, so `mar*` will match mary, maria, mark, marcy, marcus: `p4 logparse -T 'f_user f_eventtype f_date' -F 'f_user=mar*' errors.csv`

Related Commands

To add entries to the log so that `p4 logparse` can find them,	`p4 logappend`

p4 logrotate

Rotate one or more structured log files.

"Syntax" on page 19

```
p4 [g-opts] logrotate [-l logname]
```

Description

The **`p4 logrotate`** command rotates the named logfile, or rotates all structured logs if the **`-l`** ***`logname`*** option is not supplied.

If the relevant configurables are set, structured log files automatically rotate when they grow to **`serverlog.maxmb.`*`n`*** megabytes in length, and the past **`serverlog.retain.`*`n`*** log files are preserved.

By default, structured logs have no maximum size limit, and automatically rotate only on checkpointing and journaling events.

Options

`-l` ***`logname`***	Rotate the log named *logname*.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	**`super`**

p4 logschema

Describe the schema of a log record type.

"Syntax" on page 19

```
p4 [g-opts] logschema -a
p4 [g-opts] logschema recordtype
```

Description

The first form of the **`p4 logschema`** command returns a description of all the log record types in tagged format.

The second form of the **`p4 logschema`** command returns a description of the specified log record type in tagged format.

See

- "Logging and structured log files" in Helix Core Server Administrator Guide.
- the Support Knowledgebase article, "Structured Server Logs".

Options

`-a`	Display the specification of every known log record type.
recordtype	Display the specification for the specified log record type.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Examples

`p4 logschema 0`	Display the tagged output for the CommandStart log.
`p4 logschema 2`	Display the tagged output for the CommandEnd log.
`p4 logschema 6`	Display the tagged output for the Audit log.

p4 logstat

Report size of journal, error log, and/or audit log files; or report size of the specified structured file.

"Syntax" on page 19

```
p4 [g-opts] logstat [-s | -l logname]
```

Description

If no options are specified the `p4 logstat` command reports the sizes of the journal, error log (if it exists), and audit log (if it exists).

- Use the `-l` ***`logname`*** option to display the file size of the specified log.
- Use the `-s` option to report the file size for each of the structured log files defined for this server.

The following two sample commands illustrate the different output due to the use of the `-s` option.

```
C:\temp\logs> p4 logstat
journal 2591 bytes
out 126 bytes
C:\temp\logs> p4 logstat -s
journal 2591 bytes
out 255 bytes
all.csv 13599 bytes
commands.csv 11321 bytes
path/to/log/files/all.csv 13599 bytes
```

See Logging and structured log files in *Helix Core Server Administrator Guide*.

Options

`-l` ***`logname`***	Display the file size of the named *logname*. Valid values for *logname* are `journal`, `errorLog`, and `auditLog`, or any of the `serverlog.file.`*`n`* filenames associated with structured logs.
`-s`	Report the file size for each of the structured log files defined for this server.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

p4 logtail

Display the last block(s) of the error log.

"Syntax" on page 19

```
p4 [g-opts] logtail [-b blocksize] [-s start_offset [-m
maxBlocks]] [-l log]
```

Description

The `p4 logtail` command displays the last block(s) of the error log, and the offset for the next block, when available.

Output consists of a series of lines in tagged format. The first line is "`... file LOG`", followed by multiple blocks of log data. By default, all blocks from the ***`start_offset`*** are output until the end of the file. The data is returned in blocks of size ***`blocksize`***, each of which is tagged with "`... data`". The last line is "`... offsetnext_offset`", where ***`next_offset`*** is the offset in the logfile from which the next block of data is to be retrieved.

If you specify the name of an error log that has an associated counter, the `p4 logtail` command returns the current value of that counter. It also returns the current size of the log, at the end of the output (along with the ending offset in the log). The size and offset are the same if the command reads to the end of the log.

Options

`-b` ***`blocksize`***	The block size, in bytes. The default is 8192 bytes.
`-l` ***`log`***	If specified, the name of the log to display.
`-m` ***`maxBlocks`***	The maximum number of blocks to output. Ignored unless `-s` is also specified.
`-s` ***`start`***	The offset (from the beginning of the file), in bytes.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

See Logging commands in the *Helix Core Server Administrator Guide*.

Examples

`p4 logtail -b 1024 -m 2`	Display the last two kilobytes of the log file, as two separate blocks of 1,024 bytes each.

p4 merge

Merge one set of files into another.

"Syntax" on page 19

```
p4 [g-opts] merge [-c change] [-m max] [-n -Ob -q -F] [--from
stream]
                  [toFileSpec[RevSpec]]
p4 [g-opts] merge [-c change] [-m max] [-n -Ob -q] fromFileSpec
[revSpec]
                  toFileSpec
```

Description

The `p4 merge` command is a simplified form of the `p4 integrate` command: it merges a set of changes from source to target files. The command outputs the scheduled resolves. This command is intended for use with streams and distributed version control, but is also usable for traditional Helix server branches.

- Use `p4 resolve` to resolve all changes. Then use `p4 submit` to commit merged files to the depot. Unresolved files may not be submitted.
- Use `p4 shelve` to shelve merged files or `p4 revert` to delete them.
- Use the `p4 integrated` and `p4 filelog` to display merge history.

In most cases, you can use the `p4 merge` and `p4 copy` commands to propagate changes between streams (or branches).

With no arguments, the target defaults to the stream associated with the current stream client, and the source defaults to the current stream parent.

You can specify a different source with `--from stream_name`, which is an alias for the `-P` option. You can specify the stream as a directory name relative to the current stream depot. For example, `--from main` instead of `--from //Ace/main`.

Using the client workspace as a staging area, `p4 merge` schedules all affected target files to be resolved per changes in the source. Target files outside of the current client view are not affected. Source files need not be within the client view.

Each file in the target is mapped to a file in the source. Mapping adjusts automatically for files that have been moved or renamed, as long as `p4 move` was used to move or rename the files. The scope of source and target files sets must include both old-named and new-named files for mappings to be adjusted. Moved source files can schedule moves to be resolved in target files. You can limit the revisions to be merged using the *revSpec* parameter. If the scope does not include both old and new files, for example, if you run the merge on a single file that is either the move/add or move/delete of the move pair of actions, an error message is shown.

With streams, use **p4 merge** to keep a child stream up to date with a more stable parent stream. This ensures that when you promote changes back to the stable parent, you do not inadvertently overwrite any other changes that were checked into the parent. Files are opened in a pending changelist and scheduled for resolve as required. To update the parent stream, resolve and submit. By default, **p4 merge** merges changes into the current stream from its parent, or from another stream specified by the `--from` option. The source and target can also be specified on the command line as a pair of file paths. More complex merge mappings can be specified using branch specifications as with `p4 integrate`. Use the `-F` option to force merging against a stream's expected flow. You can also use this option to force the generation of a branch view based on a virtual stream. The mapping itself refers to the underlying real stream.

The **p4 integrate** and **p4 merge** commands select (as the base) the revision with the most edits in common with the source and the target.

Options

`-c` ***change***	Specifies an existing pending changelist in which the files are to be opened.
`-F`	Force merge operation. Perform the operation when the target stream is not configured to accept a merge from the source. To determine a stream's expected flow of change, use `p4 istat`.
`--from` ***stream***	Specifies a stream other than the parent stream to merge from.
`-m` ***max***	Limits the number of files merged. This option is useful for scripts that integrate large number of files. It enables them to batch the integrations and minimize the locking-related impact to other users of the shared versioning service.
`-n`	Preview the merge.
`-Ob`	The `-Ob` option displays the base revision for the merge (if any) along with each scheduled resolve.
`-q`	Quiet mode, which suppresses normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`open`

Examples

`p4 merge`
`p4 resolve`
`p4 submit -d "Update latest changes from parent stream"`
	With no arguments, the target defaults to the stream associated with the current stream client, and the source defaults to the current stream parent.

`p4 merge --from //Ace/main`
`p4 resolve`
`p4 submit -d "Update latest changes from the specified stream"`
	Merge from the specified source stream, which is not necessarily the parent, to the default target stream.

`p4 merge //`*`mySourceStream`*`/... //`*`myTargetStream`*`/...`
`p4 resolve`
`p4 submit -d "Merge changes from source to target"`
	In the case of streams: merge from the specified source stream, which might or might not be the parent, to the specified target stream.
	If streams are not used: merge from the specified source branch to the specified target branch.

Related Commands

Promote changes to more stable neighbor stream	`p4 copy`
Propagate changes	`p4 integrate`
Resolve file conflicts	`p4 resolve`

p4 merge (graph)

Merge another branch into the current or target branch.

"Syntax" on page 19

`p4 merge [options] source-branch`

```
p4 merge [options] --repo=//repo/name --target=branch source-
branch0
             [source-branchn ...]
```

`p4 merge [options] --repo=//repo/name:target:source0[:sourcen ...]...`

Description

Merges commits from the source branch into this branch, typically creating a merge commit.

Interactive mode

`p4 merge [options] source-branch`

In this mode, a `graph` client is required that maps to a single repo and the target branch is the branch to which the client is currently synced.

Non-interactive mode

`p4 merge [options] --repo=//repo/name --target=branch source-branch0 [source-branchn ...]`

The `--repo` flag specifies that the merge is a non-interactive clientless merge to be performed on the server. This requires a `--target` branch to merge to.

Atomic non-interactive mode

`p4 merge [options] --repo=//repo/name:target:source0[:sourcen ...]...`

In the atomic non-interactive mode, a clientless merge is performed on the server and involves multiple repos.

Specify the `--repo` flag repeatedly with the single target branch and one or more source branches separated with a colon (`:`) character.

Options

`-d` ***description***	The default description is `Merge into <target> <source>` by default.

`--ff-only`	Refuse to merge unless the current branch head is already up-to-date or the merge can be resolved as a fast-forward.
`--no-ff`	Create a merge commit even if it resolves as a fast-forward.
`--`*`squash`*	Perform a merge without a merge commit (single parent).
`-n`	Preview the merge.

p4 monitor

Display Perforce process information and control long-running tasks.

"Syntax" on page 19

```
p4 [g-opts] monitor show [-a -l -e -L -s R | T | P | B | F | I ]
p4 [g-opts] monitor terminate id
p4 [g-opts] monitor clear [id | all]
p4 [g-opts] monitor pause id
p4 [g-opts] monitor resume id
```

Description

`p4 monitor` allows a system administrator to observe and control Helix server-related processes running on a Helix server machine. Processes are tracked using a dedicated table that is constantly updated. This has a minor impact on server performance.

To use **`p4 monitor`**, you must enable monitoring on the Perforce service by setting the `"monitor" on page 766` configurable with **`p4 configure`**.

Valid values for the monitor configurable are:

- `0`: Server process monitoring off. (Default)
- `1`: monitor active commands
- `2`: active commands and idle connections
- `3`: sames as `2`, but also includes connections that failed to initialize (stuck at the Init() phase)
- `5`: sames as `2`, but also includes a list of the files locked by the command for more than one second
- `10`: same as `5`, but also includes lock wait times
- `25`: sames as `10`, except that the list of files locked by the command includes files locked for *any* duration

Changes to the **`monitor`** configurable affect all new **`p4`** processes that connect to the server. Restarting the server is not required.

See Enabling process monitoring in Helix Core Server Administrator Guide.

Command syntax variants provide the following alternatives:

- (`list` level access):To list current process information, use `p4 monitor show`. By default, all processes are listed, but only the command (for example, `sync`, `edit`, `submit`) is shown, without arguments. Use the `-s` *`status`* option to restrict the display to processes in the specified state.
- (`super` level access):To show the list of arguments associated with each command, use the `-a` (arguments) option or `-l` (long) option. For additional information from the user environment, use the `-e` (environment) option. To show locked files, use the `-L` option.
- (operator or `super` level access): To mark a process for termination, use `p4 monitor terminate` *`id`*. This command requires that the user be an operator or have `super` level access.

  The `p4 monitor terminate` command does not mark a process for termination unless the process has been running for at least ten seconds. Some commands, such as `p4 obliterate`, cannot be terminated.

  To control how often the list of processes is refreshed, see the Configurable, `"db.monitor.interval" on page 734`, which is also mentioned in the Support Knowledgebase article, "Fixing a hung Helix server".
- (operator or `super` level access): If a command terminates prematurely on the server side, it might be erroneously listed as running. You can clear such processes with `p4 monitor clear`.

  With `super` level access:

  - To remove an entry from the monitor table, use `p4 monitor clear` *`id`*
  - To clear the entire table, use `p4 monitor clear all`

  Processes marked as running continue to run to completion even if removed from the monitor table with `p4 monitor clear`.
- (operator or `super` level access): To control the following tasks if they are running too long, use the `p4 monitor pause` and `p4 monitor resume`:
  - `p4 admin`
  - `p4 dbstat`
  - `p4 grep`
  - `p4 ldapsync` -g
  - `p4 ping`
  - `p4 pull`
  - `p4 verify`

Output format

Each line of `p4 monitor` output consists of the following fields:

`pid status owner hh:mm:ss command` `[`*`args`*`]`

pid	The process ID under Unix (or thread ID under Windows)
status	`R`, `T`, `P`, `B`, `F`, or `I`, depending on whether the process is: ■ **R**unning ■ marked for **T**ermination ■ **P**aused ■ **B**ackground ■ **F**inish ■ **I**dle **Note** **F**inish and **B**ackground occur only in replica servers.
owner	The Helix server user name of the user who invoked the command.
hh:mm:ss	The time elapsed since the command was called.
command [*args*]	The command and arguments as received by the Perforce service.

For example, consider the following output to the **`p4 monitor show -L`** command, which displays information about locked files:

```
8764 R user 00:00:00 edit
      [server.locks/clients/88,d/ws4(W),db.locks(R),db.rev(R)]
8766 R user 00:00:00 edit
      [server.locks/clients/89,d/ws5(W),db.locks(R),db.rev(R)]
8768 R user 00:00:00 monitor
```

Following pid, status, owner, and time information, this shows two edit commands that have various files locked, including the client workspace lock in exclusive mode for the workspaces **`ws4`** and **`ws5`**, and **`db.locks`** and **`db.rev`** tables in read-only mode.

If you have enabled idle process monitoring (by setting the **`monitor`** configurable to 2), idle processes appear with a ***`status`*** of **`R`**, but with a ***`command`*** of **`IDLE`**.

Some commands (for instance, **`p4 submit`**) invoke multiple processes. For example, **`dm_CommitSubmit`** or **`dm_SubmitChange`** might appear in the output of **`p4 monitor`** as two separate phases of the **`p4 submit`** command.

Getting pull thread information for replicas

If you are running a replica with monitoring enabled and you have not configured the monitor table to be disk-resident, you can run the following command to get more precise information about what pull threads are doing. (Remember to set **`monitor.lsof`**).

"Syntax" on page 19

```
$ p4 monitor show -sB -la -L
```

Command output would look like this:

```
31701 B uservice-edge3 00:07:24 pull sleeping 1000 ms
    [server.locks/replica/49,d/pull(W)]
```

Options

You must be an operator or have `super` access to use the following options:

`-a`	Show all arguments associated with the process (for example, `edit file.c`, or `sync -f //depot/src/...`). Helix server user names are truncated to 10 characters, and each line of output is limited to 80 characters.
`-e`	Show environment information, including invoking Helix server application (if known), host IP address, and workspace name.
`-l`	Show all arguments in long form; that is, without truncating user names or the list of command line arguments.

`-L`	Show information about locked files. The information is collected only for the duration of the `p4 monitor` command, and is not persisted. Pre-requisites for using this option vary with the platform on which the server is running. ■ On Unix platforms, you must set the `monitor.lsof` configurable to the following value: `$ path/lsof -F pln` The value for *`path`* varies with the version of Unix you are using. For example, `/usr/bin/lsof`. There are circumstances in which `monitor.lsof` might not work for you: your Linux machine does not support `lsof`, the version of `lsof` might not work with the Helix server, or the administrator might not be willing to run the `lsof` command for security reasons. If this is the case, you can still get information about locked files by setting the `monitor` configurable, described next. **Note** Microsoft Windows does not have the `lsof` utility to list open files, and we therefore recommend that Windows systems use values other than `5`, `10`, or `25`. You can use the `-z tag` option with this option. In that case, the `p4 monitor show` command will return one lockinfo tag for each file that the process has locked.
`-s` *`status`*	Restrict the display to processes in the `R`unning, `T`erminated, `P`aused, <u>B</u>ackground, <u>F</u>inish, or `I`dle states.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`, `super`

Examples

`p4 monitor show`	Show Helix server process information (commands only). Requires `list` access only.

`p4 monitor show -l`	Show arguments and commands, without limits on line length. Requires **`super`** access.
`p4 monitor show -a`	Show arguments and commands, limited to 80 characters per line of output. Requires **`super`** access.
`p4 monitor terminate 123`	Instruct the Perforce service to mark process 123 for termination. Requires **`super`** access.
`p4 monitor clear all`	Clears the monitor table of all entries. Requires **`super`** access.

Related Commands

To turn on monitoring	**`p4 configure`** **set monitor=1**
To turn off monitoring	**`p4 configure`** **set monitor=0**

p4 move

Move (rename) a file from one location to another within the branch.

"Syntax" on page 19

```
p4 [g-opts] move [-c change] [-f -n -k] [-t filetype]
fromFileSpec toFileSpec
```

```
p4 [g-opts] move -r [-c change] [-n -k] fromFileSpec toFileSpec
```

Description

The `p4 move` command takes a file already opened for edit or add and moves it to the destination provided.

An open file can be moved many times before it is submitted. Moving a file back to its original location undoes the pending move, leaving it open for edit. Using `p4 revert` on a moved file both undoes the move and reverts the unsubmitted content.

Note
The `p4 move` command should only be used for intra-branch file rename or move, that is, renaming a file within the same directory, or between folders within the same directory tree. To learn how to restructure the depot by moving an entire branch, see the Knowledge Base article, "Renaming Depot Directories".

Options

`-c` *change*	If a changelist number is provided, the files are opened in the numbered pending changelist.
`-t` *filetype*	If a filetype is specified, the file is reopened as the new filetype.
`-f`	Force a move to an existing target file. The file must be synced, but not opened. The originating source file will no longer be synced to the workspace. If you use `p4 move -f`, you will need to resolve the move before submitting the changelist.

`-k`	Keep existing workspace files by bypassing the renaming in the client workspace. Use `p4 move -k` only in the context of reconciling work performed while disconnected from the Perforce service.
`-r`	Rename existing files without altering content or type to a new target. This option allows the user to move files from the depot to a different location without opening the files first. This option performs a strict rename only. Therefore the `-t` and `-f` flags are not allowed, source files must not be already opened, and the targets must not already exist.
`-n`	Preview the move that would be performed, without actually moving files.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required

`fromFile`: Yes
`toFile`: No
	No	`read` access for *`fromFile`*
		`write` access for *`toFile`*

- Files must be open for `add` or open for `edit` before they can be moved.
- To move and resolve a file that is open for edit but has been renamed at the head revision, you can use the `-f` option to force the move.

Examples

`p4 move file1.c file2.c`	Assuming that `file1.c` is open for add or edit, move `file1.c` to `file2.c`.
`p4 move //depot/main/directory1/... //depot/main/directory2/...`	Move open files from one directory to another within the same depot tree.

p4 obliterate

Removes files and their history from the depot.

"Syntax" on page 19

```
p4 [g-opts] obliterate [-y -A -b -a -h -p] FileSpec[revSpec]
```

Description

`p4 obliterate` can be used by Helix server administrators to permanently remove files from the depot.

Warning
The `p4 obliterate -y` command deletes the server's copy of a file's data, precluding any possibility of recovery. The file is removed from the Perforce service, along with all associated metadata, including references to the file in labels, the have list, and so on. After `p4 obliterate` completes, the service has no record that those files ever existed. Although any copies of those files in user workspaces are left untouched, they are no longer recognized as being under Helix server control.

Tip
Consider using the `p4 delete` command, which marks the latest revision as deleted, but leaves earlier revisions in the depot.

Tip
Consider using the `-p` option (new in 2019.2) to purge file content. For example, to permanently remove file content that was submitted in the years 2015 through 2018, while preserving all file metadata, use the following command:

```
p4 obliterate -p -y file@2015/01/01,2019/01/01
```

and for a directory:

```
p4 obliterate -p -y //depot/some/path/...@2015/01/01,2019/01/01
```

`p4 obliterate` requires at least one file pattern as an argument. To actually perform the obliteration, the `-y` option is required. Without `-y`, the `p4 obliterate` command merely reports what it would do without actually performing the obliteration.

If you specify a single revision (for instance, `p4 obliterate file#3`), only that revision of the file is obliterated. If you specify a revision range (for instance, `p4 obliterate file#3,5`), only the revisions in that range are obliterated.

"Syntax" on page 19

Note
`p4 obliterate` ***`myfile`*** does not obliterate a shelve of the file (archive or metadata). When you attempt to unshelve a file that has been obliterated, you will get an error. To recover the content of that file, print the file. To get rid of the shelve, delete the shelf.

`p4 obliterate` with a file spec does not obliterate a shelve of the file.

Options

`-y` *`filespec`*	Perform the obliterate operation. Without this option, `p4 obliterate` merely reports what it would do.
`-A`	Obliterate a revision marked for archive. By default, archived revisions are skipped.
`-b`	Restrict files in the argument range to those that are branched, and to files that are both the first revision and the head revision. This option is useful for removing old branches (where only one revision exists) while preserving files that have been modified post-branch. You can greatly improve the performance of `obliterate -b` by using the `-a` option with `-b`.
`-a`	Skip the (potentially resource-intensive) search of `db.archmap` and do not remove the file from the server. Only the file's metadata is removed. Although the file is not removed from disk, you can use `-a` in conjunction with `-b` to speed up obliteration of branched files known to exist only as lazy copies.
`-h`	Skip the search of `db.have` when looking for matching metadata to delete. The next time a client workspace that refers to these files is synced, any such files in the workspace will also be removed from the workspace. (This is often the desired behavior, for example, in client workspaces on build machines.)
`-p`	Mark the revison as purged and leave the integration history intact rather than removing the records.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`admin`

- `p4 obliterate` is one way to reclaim disk space from files that are no longer required, or to clean up mistakes made by users who create file hierarchies in the wrong place. Do not use operating system commands (**`erase`**, **`rm`**, and their equivalents) to remove files from the Helix server root by hand.

- A better way to save disk space is to relocate infrequently-accessed files onto lower-performance (or detachable) storage. Instead of obliterating files outright, consider using **`p4 archive`** and **`p4 restore`** in conjunction with an archive depot (see archive depots). With archive depots, file history is preserved and available to all users, and file contents may be moved to offline or near-line storage.
- Obliterating files can alter the behavior of user commands. Syncing to an obliterated file does not remove the file from your client workspace, because the file is no longer part of any client workspace. Syncing to an obliterated revision of a file will either report that the file does not exist (if all revisions were obliterated), or provide you with the most recent non-obliterated revision of the file.
- Obliterating files in revision ranges can change the behavior of scripts that rely on filelog output. This is because revision numbers of files "skip" or omit obliterated revisions. For example, the output of **`p4 filelog`** after obliterating revisions **`#2`** and **`#3`** might only have **`#4`** and **`#1`**:

  ```
  ... #4 change 1276 edit on 2011/04/18 by user@dev1 (binary) 'Fixed'
  ... #1 change 1231 add on 2011/04/12 by user@dev1 (binary) 'First
  try'
  ```

  In this case, if a script uses the **`#4`** in the first line of the output to assume the existence of four change descriptions in the output of **`p4 filelog`**, the existence of only two change descriptions might affect the result of running the script.
- The output of **`p4 filelog`** after purging revisions **`#1`** **and** **`#2`** (or a range including those revisions) might change from:

  ```
  ... #3 change 2123 edit on 2019/12/04 by user1@dev1 (binary) 'Third
  version'
  ... #2 change 2115 edit on 2019/12/03 by user1@dev1 (binary) 'Second
  version'
  ... #1 change 2101 add on 2019/12/01 by user1@dev1 (binary) 'First
  version'
  ```

  to the following because actions, such as **`edit`** and **`add`**, for affected revisions are changed to **`purge`**:

```
... #3 change 2123 edit on 2019/12/04 by user1@dev1 (binary) 'Third
version'
... #2 change 2115 purge on 2019/12/03 by user1@dev1 (binary) 'Second
version'
... #1 change 2101 purge on 2019/12/01 by user1@dev1 (binary) 'First
version'
```

Examples

`p4 obliterate` ***`dir/...`***	Do not obliterate any files; list the files that would be obliterated with the `-y` option. In this case, all files in directory ***`dir`*** and below would be subject to deletion with the `-y` option.
`p4 obliterate -y` ***`file`***	Obliterate ***`file`*** from the depot. All history and metadata for every revision of ***`file`*** are erased.
`p4 obliterate -y` ***`file`*****`#3`**	Obliterate only the third revision of ***`file`***. If `#3` *was* the head revision, the new head revision is now `#2` and the next revision will be revision `#3`. If `#3` was *not* the head revision, the head revision remains unchanged.
`p4 obliterate -y` ***`file`*****`#3,5`**	Obliterate revisions 3, 4, and 5 of ***`file`***. If `#5` *was* the head revision, the new head revision is now `#2`, and the next revision will be `#3`. If `#5` was *not* the head revision, the head revision remains unchanged.

Related Commands

To mark a file deleted at its head revision but leave it in the depot. This is the normal way of deleting files.	**`p4 delete`**
Instead of obliterating files, you can save disk space on a local depot by archiving some of its revisions to an archive depot. History of changes to these files is preserved.	**`p4 archive`**
To restore archived revisions from an archive depot. (You cannot restore obliterated files, but you can restore archived files.)	**`p4 restore`**

p4 opened

List files that are open in pending changelists.

"Syntax" on page 19

```
p4 [g-opts] opened [-a -c change] [-C workspace] [-u user -m max
-s -g] [file ...]
p4 [g-opts] opened [-a -x] [-m max] [file ...]
```

Description

Use `p4 opened` to list files that are currently open via `p4 add`, `p4 edit`, `p4 delete`, or `p4 integrate`. By default, all open files in the current client workspace are listed. You can use command line arguments to list only those files in a particular pending changelist, to show open files in all pending changelists, to limit the number of files displayed, or to limit the files opened for a particular user.

If file specifications are provided as arguments to `p4 opened`, only those files that match the file specifications are included in the report.

The information displayed for each opened file includes the file's name, its location in the depot, the revision number that the file was last synced to, the number of the changelist under which the file was opened, the operation it is opened for (`add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`), and the type of the file. The output for each file looks like this:

```
depot-file#rev - actionchnum change (type) [lock-status]
```

where:

- ***depot-file*** is the path in depot syntax
- ***rev*** is the revision number
- ***action*** is the operation the file was open for: `add`, `edit`, `delete`, `branch`, or `integrate`
- ***chnum*** is the number of the submitting changelist
- ***type*** is the type of the file at the given revision.
- If the file is locked (see `p4 lock`), a warning that it is `*locked*` appears at the line's end
- Files with filetypes that use the `+l` modifier are exclusively-locked (see the example for `p4 typemap`) and are displayed with a lock status of `*exclusive*`.

You can use the `-s` option to provide shortened output that omits the `#rev` number and the (***type***) of the file. This form of the command typically runs faster than the default.

The `-u` option limits output to files opened by a particular user. Otherwise, the command applies to all users.

The `-g` option lists files that are opened on a commit server in a multi-server installation.

The following examples show how user details are included in command output. In the following, information is about the current workspace and the current user is not identified. However, other users with files opened are identified.

```
$ p4 opened
//depot/file-1.txt#1 - edit default change (text) by adam@scratch
//depot/file-2.txt#1 - edit default change (text)
```

In this example, which asks for all workspaces, user names are always displayed:

```
$ p4 opened -a
//depot/file-1.txt#1 - edit default change (text) by adam@scratch
//depot/file-2.txt#1 - edit default change (text) by normal@scratch
```

Options

`-a`	List opened files in all client workspaces. In multi-server environments, this option lists files opened on all workspaces on your edge server.
`-c` ***change***	List the files in pending changelist ***change***. To list files in the default changelist, use `p4 opened -c default`.
`-C` ***workspace***	List only files that are open in the specified client ***workspace***.
`-g`	List files that are opened on the commit server in a multi-server installation. This allows you to track files that might be locked globally on the commit server from an edge server. Because the command query runs on the commit server, if you provide a file spec argument, it must be in depot syntax. The `-g` option implies the `-a` option.
`-m` ***max***	List only the first ***max*** open files.
`-s`	Short output; do not output the revision number or file type. This option is more efficient, particularly when using the `-a` (all-workspaces) option at large sites.
`-u` ***user***	List only those files that were opened by ***user***. Note that this option lists files opened in any workspace for the specified user name, not just the current workspace.

`-x`	In multi-server environments, list all open files that have the `+l` filetype (exclusive open) over all servers. This option implies the `-a` option.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`

- Helix server does not prevent users from opening already open files. Instead, Helix server, by default, allows multiple users to edit the file simultaneously, and then resolve file conflicts with `p4 resolve`. To determine whether or not another user already has a particular file opened, use `p4 opened -a` *`file`*.
- Locked files appear in the output of `p4 opened` with an indication of `*locked*`. On UNIX, you can find all locked files you have open with the following command:

  ```
  $ p4 opened | grep "*locked*"
  ```

  This lists all open files you have locked with `p4 lock`.
- `p4 opened -a` can have a performance impact on large sites. Unless you need the exact revision number or file type of an opened file, the best practice is to use `p4 opened -as`.
- `p4 opened` does not show files in shelved changelists. To display shelved changelists, use `p4 changes -s shelved`, and then use `p4 describe -s -S` *`changelist`* to display the files in the selected changelist(s).

Examples

`p4 opened -c 35 //depot/main/...`	List all files in pending changelist 35 that lie under the depot's `main` subdirectory.
`p4 opened -a -c default`	List all opened files in the default changelists for all client workspaces.

p4 opened (graph)

Display the list of files opened in pending changelists.

"Syntax" on page 19

```
p4 opened [-a -c change] [-C workspace] [-u user -m max -s] [file
...]
p4 opened [-a] [-m max] [file ...]
```

Description

Lists files currently opened in pending changelists, or, for specified files, show whether they are currently opened or locked.

If the file specification is omitted, lists the files that are open in the current client workspace.

By default, the files opened by the current user in the current client workspace are listed.

Options

`-a`	List opened files in all client workspaces. This option is ignored if `-C` or `-u` is used.
`-c` ***`change`***	List the files in pending changelist ***`change`***.
`-C` ***`workspace`***	List only files that are open in the specified client ***`workspace`***.
`-m` ***`max`***	List only the first ***`max`*** open files.
`-s`	Produces short and optimized output when used with the `-a` (all clients) option. For large repositories, `-a` can take a long time compared to `-as`.
`-u` ***`user`***	List only those files that were opened by ***`user`*** in any workspace for the specified user name, but can be combined with `-C`.

p4 passwd

Change a user’s Helix server password.

"Syntax" on page 19

```
p4 [g-opts] passwd [-O oldpassword] [-P newpassword] [user]
```

Description

By default, user records are created without passwords, and any Helix server user can impersonate another by setting `P4USER` or by using `-u`, which is one of the Global Options. To prevent another user from impersonating you, use `p4 passwd` to set your password.

Important
We recommend that you improve security by using ticket-based authentication. This requires security level 3 or higher. See Server security levels in *Helix Core Server Administrator Guide*.

To authenticate with tickets, first set a password with `p4 passwd`, and then use the `p4 login` and `p4 logout` commands to manage your authentication.

You can further improve security by assigning users to groups and setting the `PasswordTimeout:` field in the `p4 group` form. If a user belongs to more than one group, the largest `PasswordTimeout` value applies.

For Helix server applications on Windows and OS X that connect to Helix server services at security levels 0 and 1, `p4 passwd` stores the password by using `p4 set` to store the MD5 hash of the password in the registry or system settings. When connecting to Helix server services at security levels 2, 3, or 4, password hashes are neither stored in, nor read from, these locations.

Helix server superusers can reset the passwords of individual users (or all users site-wide) with the `p4 admin resetpassword` command. You can also set the `dm.user.resetpassword` configurable (set with `p4 configure`) to require that any newly-created users reset the password you assigned them when you created their account.

Tip
To avoid possible character set mismatches with LDAP servers and clients, we recommend that passwords contain only the printable characters of the ASCII table, which are characters 32 - 126 at http://www.asciitable.com/

"Syntax" on page 19

strong password

Certain combinations of security level and Helix server applications releases require users to set "strong" passwords. Helix Core server defines a strong password as:

- at least `dm.password.minlength` long, which, by default, is `8` characters
- contains at least two of the following :
  - Uppercase letter(s)
  - Lowercase letter(s)
  - Non-alphabetic character(s)

Although `abcd1234` is by default, considered a strong password in an environment with the security configurable set to `2`, it is too easy to guess.

Tip
To create secure password that is easy-to-remember:

1. Start with a phrase, such as
   `Perforce Enterprise-class Version Control.`
2. Make the phrase resemble a single word, such as
   `PEnterprise-classVC.`
3. Represent some letters with non-alphabetical characters:
   `PN2prI$-k|@zV(.`

See also Server security levels in *Helix Core Server Administrator Guide*.

If your security needs are minimal

We recommend using ticket-based authentication. However, if your security needs are minimal, you can use one of these methods:

Method 1	Set the environment variable `P4PASSWD` to the password value
Method 2 (overrides Method 1)	Create a setting for `P4PASSWD` within the `P4CONFIG` file.
Method 3 (overrides Methods 1 and 2)	Use the `-P` password option on the command line. For example, `p4 -u bruno -P PN2prI$-k|@zV(. sync` allows the administrator to invoke the `p4 sync` command as the user named `bruno`.

Depending on the security level of your installation, one or more of these methods might not be permitted. See the "Server security levels" topic in *Helix Core Server Administrator Guide*.

Options

`-O` *`oldpassword`*	Avoid prompting by specifying the old password on the command line. This option is not supported if your site is configured to use security level 2, 3, or 4. If you use the `-O` option, you must use the `-P` option.
`-P` *`newpassword`*	Avoid prompting by specifying the new password on the command line. This option is not supported if your site is configured to use security level 2, 3, or 4.
`user`	Superusers can provide this argument to change the password of another user.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- Passwords can be up to 1,024 characters in length. As of Release 2013.1, password length is configurable by setting the `dm.password.minlength` configurable. To require passwords to be at least 16 characters in length, a superuser can run:

  ```
  $ p4 configure set dm.password.minlength=16
  ```

  The default minimum password length is eight characters.
- The `p4 passwd` command never sends plaintext passwords over the network. A challenge/response mechanism is used to send the encrypted password to the service.
- A password can contain spaces, but command line use of such a password requires quotes to enclose it in a single string:

  `p4 -P "my password"` *`command`*
- If a user forgets her password, a Helix server superuser can reset it by specifying the username on the command line:

  `p4 passwd` *`username`*
- To delete a password, set the password value to an empty string. Depending on your site's security level, your Perforce service might not permit you to set a null password.
- If you are using ticket-based authentication, changing your password invalidates all of your tickets and logs you out. This is equivalent to `p4 logout` **-a**.

Examples

The superuser wants to create a new user named `joecoder` and assign a password to that user:

```
p4 -u -f joecoder passwd
```

The server displays a user spec with default values, which the superuser accepts.

The server responds:

```
Enter new password:
```

The superuser types a password for `joecoder`, and the server responds:

```
Re-enter new password:
```

The superuser repeats the password, and the server responds:

```
Password updated.
```

Related Commands

To change other user options	`p4 user`
To change users' access levels	`p4 protect`
To log in using tickets instead of passwords	`p4 login`
To force password reset	`p4 admin resetpassword`

p4 ping

Test network performance.

"Syntax" on page 19

```
p4 [g-opts] ping [-f] [-p pausetime] [-c count] [-t transmittime]
                 [-i iterations] [-s sendsize] [-r receivesize]
```

Description

`p4 ping` simulates Helix server network traffic by sending messages from the versioning service to the Helix server application and back, and times the round trips. Round-trip times are reported in milliseconds. Because the round-trip time is typically too fast to measure for a single message, you can specify a message ***`count`*** per test.

Options

`-c` *`count`*	Number of messages per test.
`-f`	Flood mode: the service transmits continuously, sending the next message without waiting for the Helix server application to confirm receipt of the prior message.
`-i` *`iterations`*	Repeat the test for the specified number of ***`iterations`***.
`-p` *`pausetime`*	Pause for ***`pausetime`*** seconds between tests, up to 120 seconds. To disable pausing, specify a pausetime of 0.
`-r` *`receivesize`*	Size of the user-to-service message, up to a maximum value of 100,000 bytes.
`-s` *`sendsize`*	Size of the service-to-user message, up to a maximum value of 10,000,000 bytes.
`-t` *`transmittime`*	Transmit data for ***`transmittime`*** (maximum 6,000) seconds.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`admin`

- Like the operating system's counterpart, **`p4 ping`** can flood the network with traffic.

p4 populate

Branch a set of files as a one-step operation.

"Syntax" on page 19

```
p4 [g-opts] populate [-d description] [-f -n -o] [-m max]
fromFile[rev] toFile
```

```
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] -b
branch [-r]
                         [toFile[rev]]
```

```
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] -b
branch -s fromFile[rev]
                         [toFile]
```

```
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] -S
stream [-P parent] [-r]
                         [toFile[rev]]
```

Description

The **`p4 populate`** command branches a set of files (the source) into another depot location (the target) in a single step. The new files are created immediately, without requiring a **`p4 submit`** or a client workspace.

The execution of the **`p4 populate`** command fires a **`change-submit`** trigger to allow interested parties to perform a check or validation before submission. See "Change-submit triggers" in the *Helix Core Server Administrator Guide*.

Similar to the pending changelist of a change-content trigger, a **`change-submit`** temporary pending change record is created so that the description can be accessed. However, no files are returned to **`change-submit`** triggers from "p4 describe" on page 157 and "p4 opened" on page 383 because files branched with **`p4 populate`** are not opened.

If no **`-d`** ***`description`*** is given, the command line arguments are used for a description.

If the **`p4 populate`** command fails after the **`change-content`** stage succeeds, a **`change-fail`** trigger is enabled. See "change-failed" in "Triggering on submits".

The value of the `"rpl.checksum.change" on page 794` configurable determines the level of verification performed for this command.

"Syntax" on page 19

Tip
To get a list of the files involved in a populate trigger script, use the `p4 files @=`***`change`*** command.

Options

`-b` ***`branch`***	Use the view in a user-defined ***`branch`*** specification; the source is the left side of the branch view and the target is the right side of the branch view.
`-d` ***`description`***	Provide a description for the automatically-submitted changelist. If no description is provided, the command line arguments are used for a description.
`-f`	Force deleted files to be branched into the target. (By default, deleted files are treated as nonexistent and are skipped.)
`-m` ***`max`***	Limit the action to the first ***`max`*** files.
`-n`	Preview the operation without actually doing anything.
`-o`	Display a list of files created by the `p4 populate` command.
`-P` ***`parent`***	When used with `-S` ***`stream`***, specify a parent stream other than the stream's actual parent.
`-r`	Reverse direction of integration (from target to source, rather than from source to target).
`-s`	If used with `-b` ***`branch`***, treat ***`fromFile`*** as the source, and use both sides of the user-defined branch view as the target. (Optional ***`toFile`*** arguments may be given to further restrict the scope of the target file set.) The `-r` option is ignored when `-s` is used.
`-S` ***`stream`***	Use a stream's view; the source is the stream itself, and the target is the stream's parent.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	No	`write`

p4 print

Print the contents of a depot file revision.

"Syntax" on page 19

```
p4 [g-opts] print [-a -A -k -q] [-m max] [-o outfile] FileSpec
[revSpec]
p4 [g-opts] print -U unload FileSpec
```

Description

The **p4 print** command writes the contents of a depot file to standard output.

A revision range can be included. In this case, only the files with revisions in the specified range are printed. By default, only the highest revision in that range is printed.

Tip
To output a file at every revision within a specified revision range, such as versions `2`, `3`, and `4` of `readme.txt`:

```
p4 print -a readme.txt#2,4
```

The output is the contents of `readme.txt#4`, followed by the contents of `readme.txt#3`, and concluding with the contents of `readme.txt#2`.

To get solely version `3`:

```
p4 print -a readme.txt#3,3
```

To get all the versions up to version `3`:

```
p4 print -a readme.txt#3
```

To get all versions:

```
p4 print -a readme.txt#head
```

Multiple file patterns can be included. All files matching any of the patterns are printed.

Any file in the depot can be printed, subject to permission limitations as granted by **p4 protect**.

If the file argument does not map through the client view, you must provide it in depot syntax.

By default, the file is written with a header that describes the location of the file in the depot, the revision number of the printed file, and the number of the changelist that the revision was submitted under. To suppress the header, use the **-q** (quiet) option.

By default, RCS keywords are expanded. To suppress keyword expansion, use the **-k** (keyword) option.

By default, the local depot is searched for the specified file. If you specify the `-U` option, the unload depot is searched instead.

Options

`-a`	For each file, print all revisions within a specified revision range, rather than only the highest revision in the range.
`-A`	Attempt to print a file stored in an archive depot.
`-k`	Suppress RCS keyword expansion.
`-m` ***`max`***	Print only the first ***`max`*** files.

`-o` *`outfile`*	Redirect output to the specified output file (*outfile*) on the local disk.
	This preserves the same file type, attributes, and/or permission bits as the original file (*FileSpec*) in the depot.
	Multiple files can be written by using wildcards in the localFile argument that match wildcards in the depot (*FileSpec*) argument.
	For example:
	To print the contents of a directory and directories under that directory, use the `...` wildcard:
	`$ p4 print -o c:/tmp/main-copy/... //depot/main/...`
	To print all files that match `readme.txt` or `readme.pdf`, you might specify
	`$ p4 print -o readme.* //depot/readme.*`

`-q`	Suppress the one-line file header normally added by Helix server.
`-U`	Look for the specified file or files in the unload depot. Data about an unloaded client, label, or task stream can be printed.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`read`

- Because most terminals are unable to display UTF16 content, the default behavior of the `p4 print` command is to return UTF8 content. You can override this behavior by bypassing terminal output entirely and specifying an output file, for example:

  ```
  $ p4 print -q -o outputfile //depot/file
  ```

  If your terminal supports UTF16 output, specify standard output as the output file:

  ```
  $ p4 print -q -o - //depot/file
  ```

- `p4 print`'s file arguments can take a revision range. By default, only the highest revision matched by any particular file is printed (that is, when no range is specified, the implied range is `#1,#head`, and the highest revision is `#head`). To print all files in a specified (or implied) range, use the `-a` option.
- Because `p4 print`'s output can be quite large when called with highly non-restrictive file arguments (for instance, `p4 print //depot/...` prints the contents of all files in the depot), it may be subject to a `maxresults` limitation as set in `p4 group`.
- In many cases, redirecting `p4 print`'s output to a file via your OS shell will suffice.

  The `-o` option is intended for users who require the automatic setting of file type and/or permission bits. This is useful for files such as symbolic links (stored as type `symlink`), files of type `apple`, automatically setting the execute bit on UNIX shell scripts stored as type `text+x`, and so on.

Related Commands

To compare the contents of two depot file revisions	`p4 diff2`
To compare the contents of an opened file in the client workspace to a depot file revision	`p4 diff`

p4 print (graph)

Write a repo file to standard output.

"Syntax" on page 19

```
p4 print [-o localFile -q -m max] file ...
```

Description

Retrieve the contents of a repo file to the client's standard output. This command does not sync the workspace with the graph depot. If the file is specified using client syntax, Perforce uses the client view to determine the corresponding repo file.

Options

`-m` ***max***	Print only the first ***max*** files.

`-o` ***outfile***	Redirect output to the specified output file (*outfile*) on the local disk.
	Multiple files can be written by using wildcards in the localFile argument that match wildcards in the depot (*FileSpec*) argument.
	For example:
	To print the contents of a directory and directories under that directory, use the `...` wildcard:
	`$ p4 print -o c:/tmp/dev-copy/... //repo/dev/...`
	To print all files that match **`readme.txt`** or **`readme.pdf`**, you might specify:
	`$ p4 print -o readme.* //repo/readme.*`

`-q`	Suppress the one-line file header normally added by Helix server.

p4 property

Add, delete, or list property values.

"Syntax" on page 19

```
p4 [g-opts] property -a -n name -v value [-s sequence] [-u user |
-g group]
```

```
p4 [g-opts] property -d -n name [-s sequence] [-u user | -g
group]
```

```
p4 [g-opts] property -l [-A] [-n name [-s sequence] [-u user | -g
group]]
                      [-F filter] [-T taglist] [-m max]
```

Description

The **p4 property** command can be used by administrators to view and update property definitions stored in the Perforce service. The service does not use the property definitions; it provides this capability for other Helix server applications, such as P4V.

The Perforce service offers three ways of storing metadata: counters/keys, attributes, and properties.

If your application requires only the flat storage of simple key/value pairs, and attempts to implement no security model, use the **p4 counters** and **p4 keys** commands.

If your application's metadata is associated with particular files, use **p4 attribute**.

If your application's metadata is not associated with files, and if you have a requirement to restrict its visibility to users, groups, and/or to control the precedence of multiple values using sequence numbers, use **p4 property**.

When specifying multiple property values for the same property, use distinct sequence numbers to specify the precedence order. A value with a higher sequence number is ordered before a value with a lower sequence number. Values with the same sequence number have an undefined ordering relationship.

Options

`-a`	Update a property value, or add a property value if it is not yet present. Requires `admin` access.

`-A`	List properties for all users and groups, as well as the property sequence number of each property value. Requires `admin` access.
`-d`	Delete a property value. Requires `admin` access.
`-F` *`filter`*	Limit the properties displayed to those that match the *`filter`* pattern. Syntax is that used by `p4 fstat`.
`-g` *`group`*	Specify the user group to which this property applies.
`-l`	List one or more property values. Performance is substantially improved when you supply a `-n` *`name`* argument.
`-m` *`max`*	Limit output to the first max matching properties.
`-n` *`name`*	Specify the name of the property.
`-s` *`sequence`*	Specify the sequence number of the property. If not specified, the default value is `1`.
`-T` *`taglist`*	Limit the fields that are displayed to the fields listed in *`taglist`*. Syntax is that used by `p4 fstat`.
`-u` *`user`*	Specify the user to whom this property applies.
`-v` *`value`*	Specify the value of the property.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`, `admin`

- Helix server administrators can use `p4 property` to centrally manage P4V's performance settings and selectively enable/disable features. See Staging P4V help files locally in the *Helix Core Server Administrator Guide*.

p4 protect

Control user access to files, directories, and commands.

Note
These access rights are different from the graph depot permissions described at "p4 grant-permission (graph)" on page 227.

"Syntax" on page 19

```
p4 [g-opts] protect
p4 [g-opts] protect -o
p4 [g-opts] protect -i
p4 [g-opts] protect --convert-p4admin-comments -o | -i
```

Description

Important
We recommend that you run `p4 protect` immediately after installation because a new Helix server installation:

- allows anyone who wants to use Helix server to connect to the service
- considers all Helix server users as superusers

The first time a user runs `p4 protect`, that user is made the superuser, and all other users are given `write` privilege on all files. We recommend that you use `p4 protect` to create additional controls.

`p4 [g-opts] protect` allows a user with `super` access to edit the protections table in a text form.

Use `p4 protect` to set Helix server privileges in any of the following ways:

- Control which commands particular users can access
- Control which commands particular users can run
- Grant access to groups of users, as defined with `p4 group`
- Grant access to the `p4 protect` command **for a particular path** to a user or group
- Use `=read`, `=open`, `=write`, and `=branch` in conjunction with an exclusionary mapping
- Limit access to particular IP addresses

It is common to specify any of the following access levels: `list`, `read`, `write`, `owner`, and `super`. The `open` and `review` access levels are used less often.

In general,

1. You grant an access level to a:
   - group (best practice) or
   - a user (if you cannot combine multiple users into a suitable group)
2. If finer-grained control is required, you can selectively deny one or more specific rights.

Tip
To understand the effect of "exclusionary mappings", see How protections are implemented in Helix Core Server Administrator Guide.

Permission levels and access rights

If the name of a permission level has the `=` prefix, it means the denial of a right.

Permission Level / Right	What the User Can Do
`list`	The user can access all Helix server metadata, but has no access to file contents. The user can run all the commands that describe Helix server objects, such as `p4 files`, `p4 client`, `p4 job`, `p4 describe`, `p4 branch`, etc. Those commands that list files, such as `p4 describe`, will only list those files to which the user has at least `list` access.
`read`	The user can do everything permitted with `list` access, and also run any command that involves reading file data, including `p4 print`, `p4 diff`, `p4 sync`, and so on.
`=read`	If this right is denied, users cannot use `p4 print`, `p4 diff`, or `p4 sync` on files. For example, `=read group Dev1 * -//depot/dev/prodA/...`

Permission Level / Right	What the User Can Do
`open`	This gives the user permission to do everything she can do with `read` access, and gives her permission to `p4 add`, `p4 edit`, `p4 delete`, and `p4 integrate` files. However, the user is not allowed to lock files or submit files to the depot. The `open` access level gives the user permission to change files but not submit them to the depot. Assign this level when you're temporarily freezing a codeline, but don't want to stop your developers from working, or when you employ testers who are allowed to change code for their own use but are not allowed to make permanent changes to the codeline.
`=open`	If this right is denied, users cannot open files with `p4 add`, `p4 edit`, `p4 delete`, or `p4 integrate`. For example, `=open group Dev1 * -//depot/dev/prodA/...`
`write`	The user can do all of the above, and can also write files with `p4 submit` and lock them with `p4 lock`.
`=write`	If this right is denied, users can open and edit files, but cannot submit them. For example, `=write group Admins * -//depot/dev/prodA/...` to ensure that Admins do not accidentally alter the files that developers have approved for a product release.
`=branch`	If this right is denied, users cannot use files as a source for `p4 integrate`. For example, `=branch group Dev1 * -//depot/dev/prodA/...`
`review`	This permission is meant for external programs that access Helix server. It gives the external programs permission to do anything that `list` and `read` can do, and grants permission to run `p4 review` and `p4 counter`. It does not include `open` or `write` access.
`admin`	Includes all of the above, including administrative commands that override changes to metadata, but do not affect service operation. These include `p4 branch` -f, `p4 change` -f, `p4 client` -f, `p4 job` -f, `p4 jobspec`, `p4 label` -f, `p4 obliterate`, `p4 shelve` -f -d, `p4 typemap`, `p4 unlock` -f, and `p4 verify`.
`super`	Includes all of the above, plus access to the superuser commands such as `p4 admin`, `p4 counter`, `p4 triggers`, `p4 protect`, the ability to create users with `p4 user` -f, and so on.

Permission Level / Right	What the User Can Do
`owner`	Use this permission to assign permission to a specific user or group to run `p4 protect` for a particular path. For details, see the section Authorizing access in *Helix Core Server Administrator Guide*.

Stream spec permissions

Important
Prior to release 2020.1, there were no protections modes specific to stream specs. The permissions for listing, reading, and editing stream specs were inferred from the corresponding filespec protections as follows:

- `list` permission in a depot granted the ability to run p4 streams to list streams in that depot
- `open` or `write` permission on a file spec granted the ability to edit and save any stream spec whose stream root matches that file spec

The 2020.1 release added protections modes that are specific to stream specs. By default, these permissions can exist in the protection table, but will not be used until the dm.protects.streamspec configurable has been set to `1`. If the `dm.protects.streamspec` configurable is set to `1` and any stream spec permissions exist in the protection table, the pre-2020.1 permissions no longer apply and all users who are not admin or super require explicit stream spec permissions.

If you want to implement the legacy permissions with the stream spec permissions:

- Create a user group containing all users who require `readstreamspec`
  `readstreamspec group readgroup * //...`
- Create another user group containing all users who require `writestreamspec`
  `writestreamspec group writegroup * //...`

`readstreamspec`	The user can display a stream spec with `p4 stream -o`
`=readstreamspec`	If this right is denied, users cannot execute `p4 stream -o`
`openstreamspec`	This gives the user permission to revert, resolve, shelve, or open for edit a stream spec.
`=openstreamspec`	If this right is denied, users cannot revert, resolve, shelve, or open for edit a stream spec.
`writestreamspec`	The user can submit or modify a stream spec.
`=writestreamspec`	If this right is denied, users cannot submit or modify a stream spec.

"Syntax" on page 19

Form Fields

When you run `p4 protect`, Helix server displays a form with a single field, `Protections:`. Each permission is specified in its own indented line under the `Protections:` header, and uses the following five values to define protections:

Column	Description
Access Level or Mode	One of the access levels `list`, `read`, `open`, `write`, `admin`, `super`, `owner`, `review`; or one of the rights `=read`, `=open`, `=write`, and `=branch`, as defined above.
User or Group	Specifies whether this protection applies to a `user` or a `group`.
Group Name or User Name	The name of the user or the name of the group, as defined by `p4 group`. To grant this permission to all users, use the `*` wildcard.
Host	The IP address, or a comma-separated list of IP addresses, of the client host. IPv6 addresses and IPv4 addresses are also supported. You can use the `*` wildcard to refer to all IP addresses, but only when you are not using CIDR notation. If you use the `*` wildcard with an IPv6 address, you must enclose the entire IPv6 address in square brackets. For example, `[2001:db8:1:2:*]` is equivalent to `[2001:db8:1:2::]/64`. Best practice is to use CIDR notation, surround IPv6 addresses with brackets, and to avoid the `*` wildcard. How the system forms host addresses depends on the setting of the `dm.proxy.protects` variable. By default, this variable is set to 1. This means that if the client host uses some intermediary (proxy, broker, replica) to access the server, the `proxy-` prefix is prepended to the client host address to indicate that the connection is not direct. If you specify `proxy-*` for the `Host` field, that will affect all connections made via proxies, brokers, and replicas. A value like `proxy-10.0.0.5` identifies a client machine with an IP address of `10.0.0.5` that is connected to the server through an intermediary. Setting the `dm.proxy.protects` variable to 0, removes the `proxy-` prefix and allows you to write a single set of protection entries that apply both to directly-connected clients as well as to those that connect via an intermediary. This is more convenient but less secure if it matters that a connection is made using an intermediary. If you use this setting, all intermediaries must be at release 2012.1 or higher.

Column	Description
Depot File Path	The depot file path on which this permission is granted, in Helix server depot syntax. The file specification can contain Helix server wildcards. To exclude this mapping from the permission set, use a dash (`-`) as the first character of this value. If a depot is excluded, the user denied access will no longer see the depot in the output of **`p4 depots`**. Nor will the depot show up for this user in the default branch, client, and label views.
SubPath	The root path of the sub-protections table: ■ only used when editing a sub-protections table ■ to learn about the "sub-protections table", see "Delegate management of parts of the protections table" in *Helix Core Server Administrator Guide* This field is not part of the protections table.
Update	The date this specification was last modified (read-only). ■ If this date is modified, the protections table will not be saved ■ Omitting this field will avoid the modification check. This field is not part of the protections table.

Comments

Protection tables can be difficult to interpret and debug. Including comments can make this work much easier.

- You can append comments at the end of a line using the ## symbols:

```
write user *   10.1.1.1   //depot/test/...  ## my comment
```

- Or you can write a comment line by prefixing the line with the ## symbols:

```
## my comment
write user *   10.1.1.1   //depot/test/...
```

Warning
Comments you have created using the P4Admin tool are not compatible with comments created using **`p4 protect`** (2016.1 or later). You can use the following command to convert a file containing comments created with P4Admin into a file containing **`p4 protect`** type comments:

```
$ p4 protect --convert-p4admin-comments -o
```

Then save the resulting file.

After you have converted the comments, you must continue to define and manage protections using `p4 protect` and can no longer use P4Admin to do so because this tool is unable to parse `p4 protect` comments.

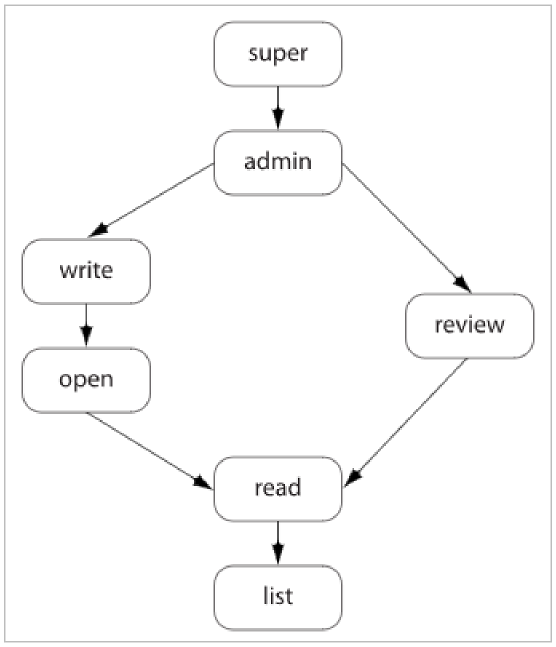
Options

`-i`	Read the form from standard input without invoking an editor.
`-o`	Write the form to standard output without invoking an editor.
`--convert-p4admin-comments`	Converts an existing protections form (and comments) created using P4Admin tool, to a form that can be used by `p4 protect`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`super`

Each permission level includes all the access levels below it:



The specific rights of `=read`, `=open`, `=write`, and `=branch` can be used to override the automatic inclusion of lower access levels. This makes it possible **to deny individual rights without having to then re-grant lesser rights**.

For example, if you want administrators to have the ability to run administrative commands, but to deny them the ability to make changes in certain parts of the depot, you could set up a permissions table as follows:

```
Protections:
    admin       user      joe       *       //...
    =write      user      joe       *       -//depot/build/...
    =open       user      joe       *       -//depot/build/...
```

In this example, user `joe` can perform administrative functions, and this permission applies to all depots in the system. Because the `admin` permission level also implies the granting of all lower access levels, `joe` can also write, open, read and list files anywhere in the system, including `//depot/build/`. To protect the build area, the `=write` and `=open` exclusionary lines are added to the table:

- The `=open` exclusion prevents `joe` from opening any files for edit under `//depot/build/`
- The `=write` exclusion prevents `joe` from submitting any changes he might already have open under `//depot/build/`

To limit or eliminate the use of the files on a particular server as a remote depot from another server (as defined by `p4 depot`), create protections for user `remote` (or for the service user by which the other server authenticates itself). Remote depots are accessed either by the service user associated with the user's Perforce service, or by a virtual user named `remote`.

Access levels determine which commands you can use. For example, because `p4 add` requires at least `open` access, you can run `p4 add` if you have `open`, `write`, `admin`, or `super` access.

Some commands (for instance, "p4 change" on page 83, when editing a previously submitted changelist) take a `-f` option that requires admin or super access.

For command-by-command details, see "Access levels required by Helix server commands" in *Helix Core Server Administrator Guide*.

Related Commands

To create or edit groups of users	`p4 group`
To list all user groups	`p4 groups`
To grant permission for depots and repos of type graph	"p4 grant-permission (graph)" on page 227

p4 protects

Display protections in place for a given user, group, or path.

"Syntax" on page 19

```
p4 [g-opts] protects [-s spec][-a | -g  group | -u user]
            [-h host] | -H] [-m] [-S | -A] [file ...]
p4 [g-opts] protects -M [-g  group | -u user] [-h host | -H]
[file ...]
```

Description

Use the `p4 protects` command to display the lines from the protections table that apply to a user, group, or set of files.

- With no options, `p4 protects` displays the lines in the protections table that apply to the current user. If a *`file`* argument is provided, only those lines in the protection table that apply to the named files are displayed.
- Use the `-a` option to display lines for all users, or `-u` *`user`*, `-g` *`group`*, or `-h` *`host`* options to display lines for a specific user, group, or host IP address.
- Use the `-h` *`host`* option to display lines that apply to the specified host (IP address).
- Use the `-H` *`host`* option to display lines that apply to the current client's host (IP address).
- Use the `-m` option to display a one-word summary of the maximum applicable access level, ignoring any provided file path.
- Use the `-M` option to display a one-word summary of the maximum access level. Unlike the `-m` option, it takes into account the specified file path.
- Use the `-s` *`spec`* option to dispay the contents of the file in the spec depot rather than the current protections table. This allows the `super` user back-in-time access to how permission would have behaved in a previous version of the protections spec.

  For example, the following command returns information about the user `sam` in the third revision of the protections table:

  ```
  $ p4 -u super protects -s //spec/protect.p4s#3 -u sam
  write user * * //...
  ```

  This is useful when users lose access privileges at a given point in time and you want to check what changes were made to the protection table just before that date.

To use this option, you must define a spec depot for protect forms. This automatically saves revisions to the protect specification every time you edit the protection table. For information on how to create a spec depot, see `p4 depot`

- Use the `-S` option to report only stream spec protection lines.
- Use the `-A` option to report both stream spec and stream file lines.

Options

`-a`	Display protection lines for all users. This option requires `super` access.
`-g` *`group`*	Display protection lines that apply to the named *`group`*. This option requires `super` access.
`-h` *`host`*	Display protection lines that apply to the specified *`host`* IP address. This option requires `super` access.
`-H` *`host`*	Display protection lines that apply to the current client's *`host`* IP address. This option requires `super` access.
`-m`	Display a one-word summary of the maximum applicable access level. (Note: this does not take into account exclusionary mappings or the specified file path into account.)
`-M`	Differs from `-m` because `-M` does take into account exclusions and the specified file path.
`-s` *`spec`*	Display information from the specified earlier version of the protect file. The *`spec`* parameter specifies the path of the file version you're interested in. To automatically save protect revisions every time you edit a form, define a spec depot for protect forms.
`-u` *`user`*	Display protection lines that apply to the named *`user`*. This option requires `super` access.
`-S`	Display only stream spec protection lines.
`-A`	Display file and stream spec protection lines.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`, `super` for `-a`, `-h`, `-H`, `-g`, `-u`

- If the `dm.protects.allow.admin` configurable is set to `1`, Helix server administrators, in addition to Helix server superusers, can also use `p4 protects -a`, `-g`, and `-u`.

Related Commands

To edit the protections table	`p4 protect`

p4 proxy

Display Proxy connection information.

"Syntax" on page 19

```
p4 [g-opts] proxy
```

Description

If connected through a Helix Proxy, the **`p4 proxy`** command displays information about the proxy connection.

For complete information on the use and configuration of proxies, see Helix Proxy in *Helix Core Server Administrator Guide*.

Options

`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`none`

- This command only works when the user is connected to a Helix Proxy.

Related Commands

To display information about a connection to Perforce	`p4 info`

p4 prune

Removes unmodified files from a stream.

The `p4 prune` command is equivalent to the `p4 obliterate` command, except that it can be done by the stream owner rather than an administrator.

"Syntax" on page 19

```
p4 [g-opts] prune [-y] -S stream
```

Description

The `p4 prune` command permanently removes unmodified files (files with one revision) from a stream that is no longer being actively used. Only the owner of a stream may prune it.

By default, `p4 prune` displays a preview of the results. To execute the operation, issue the command again, this time using the `-y` option.

After a stream has been pruned, files with more than one revision remain in the stream so that their edit history is preserved. Unmodified files are gone as if obliterated by an administrator with the `p4 obliterate` command.

Pruned files remain in client workspaces until the next `p4 sync` command runs, which removes them. If pruned files have been branched to a child stream, new integration records are generated to directly link the branched files in the child stream to the files in the parent stream that they were previously related to indirectly.

Mainline, task, and virtual streams may not be pruned. To remove unmodified files from a task stream, delete or unload the stream using the `p4 stream` or `p4 unload` command. The unmodified files automatically go away when the stream spec does.

The stream owner who executes this command must have write access, as granted by the `p4 protect` command.

Options

`-y`	Execute the command. Without this option, the command previews the results but takes no action.
`-S` *`stream`*	The name of the stream you want to prune.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`write` for stream owner

Related Commands

An equivalent for **`p4 obliterate -ahbi`**	`p4 obliterate`

p4 pubkey (graph)

Add, update, or delete an SSH public key on the Helix server.

> **Note**
> For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] pubkey -i [-u user] [-s scope] [-f]
p4 [g-opts] pubkey -d -u user | -s scope
```

Description

The user or the administrator uses this command to:

- upload a user’s public SSH key onto the server
- force an update of the user’s key that overwrites the old key
- delete a user’s key

Scope

When you upload a user’s public SSH key onto the server, if you do not specify an explicit scope, the key gets `default` as its implicit scope name.

If a user has more than one computer, and thus more than one SSH key, we recommend that you assign an explicit `scope` name for each key. A typical scope name might be a physical location, such as `home` or `office`, or a computer name, such as `server1`, `macintosh_laptop`, or `Windows_desktop`.

When you delete a key associated with the specified user, if you do not specify an explicit scope, this command deletes the key with the implicit `default` scope.

You cannot update the scope, but you can delete the key and generate a new key with whatever scope you want.

> **Note**
> To get a listing of all the keys and scopes for a specific user, see the `p4 pubkeys` topic, "Examples" on page 419.

"Syntax" on page 19

Prerequisite for a user to upload a key

The prerequisite to being able to upload a SSH public key on the Helix server is to have, at a minimum, the `list` access to a filename in the protections table. Let us assume that the Helix server administrator wants the protections table to provide only this minimum access.

As super user, the administrator issues the `p4 protect` command, which opens the `Protections` form.

The administrator fills out the `Protections` form to specify that members of a group named `repoUsers` have `list` access to a specific file:

```
list group repoUsers * //depot/no-file.txt
```

Note that `no-file.txt` does not need to be an actual file.

The administrator issues the `p4 group` command with the `repoUsers` group name:

```
$ p4 group repoUsers
```

In the Group Specification form, the administrator adds the names of the users who belong to the `repoUsers` group.

Options

`-d`	Deletes the key for the specified user. If you do not specify a scope, deletes the specified user's key that has the `default` scope. If you specify a scope, deletes the key associated with that scope.
`-f`	The `-f` option allows a user or an administrator to force an overwrite of the user's existing key.
`-i` *`entry-from-standard-input`*	Uploads the public SSH key from the location specified in a standard input redirect.
`-s`*scope*	Assigns a scope to this public key, which is useful if a given user has more than one key. If you do not use this option to specify a scope, the scope is set to `default` automatically.
-u *username*	Administrators can associate the key with a specific user. **Note** There is no check to validate whether the specified user actually exists.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Examples

Upload with scope

The user indicates the location of the key with a standard input redirect. A best practice is to provide an explicit scope that helps the user identify which computer the key belongs to. The explicit scope also ensures the user does not inadvertently overwrite the implicit scope, which has `default` as its implicit name.

```
p4 pubkey -i < path/to/your/.ssh/id_rsa.pub -s mac-laptop
```

The admin can upload the key for the user by including the `-u`*username* option.

```
p4 pubkey -i < path/to/your/.ssh/id_rsa.pub -u bruno -s mac-
laptop
```

Force an update to an existing key

If a message appears that is similar to the following:

```
Public Key for 'bruno/mac-laptop' already exists, use '-f' to
replace.
```

To force the overwrite of an existing key, include the `-f` force option.

```
p4 pubkey -f -u bruno -s mac-laptop -i < path/to/your/.ssh/id_
rsa.pub
```

Delete a key

To delete the SSH public key for the specified user and the `default` scope, which is implicit.

```
p4 pubkey -d -u bruno
```

To delete the SSH public key for the specified user and scope.

```
p4 pubkey -d -u bruno -s windows-laptop
```

Related Commands

To list the SSH public keys	`p4 pubkeys`

p4 pubkeys (graph)

Display a list of the SSH public keys.

> **Note**
> For depots of type `graph` only.

"Syntax" on page 19

```
p4 [-ztag] pubkeys [-u user] [-s scope]

p4 [g-opts] pubkeys [-u user] [-s scope]
```

Description

> **Note**
> An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

To get a listing of the public SSH keys that have been posted on the server, the administrator issues the `p4 pubkeys` command. In addition to the administrator, any user with `list` access can run this command. See `p4 protect`.

To learn about the `scope` of a key, see the section "Scope" on page 415 in the topic `p4 pubkey`.

> **Note**
> To get additional information, such as the `scope` of the SSH key. Use the `scope` to differentiate the keys for a user who has more than one SSH key.

Options

`-u` *`username`*	Limit the output to the specified user.
`-s` *`scopename`*	Limit the output to the specified scope. To learn about the `scope` associated with a key, see `p4 pubkey`.
`-ztag`	Shows additional data, such as the `scope` of that SSH key. A user might have multiple keys, each of which has a different `scope`.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Examples

To get additional information, such as the `scope` of the SSH key. Use the `scope` to differentiate the keys for a user who has more than one SSH key.

```
p4 -ztag pubkeys -u bruno
```

This output shows that the user has one key for a laptop computer and another key for a desktop computer:

user	scope	digest	lastUpdate	key
`bruno`	`desktop`	`9DF…C5`	`2017/03/22 10:03:01`	`ssh-rsa AN…C1==` bruno@p4.com
`bruno`	`laptop`	`B3…BF`	`2017/04/08 14:08:29`	`ssh-rsa Aj…6Q==` bruno@p4.com

Note: The `digest` is explained at `p4 fstat` under Form Fields.

To limit the list to a specific scope, use the `-s` option:

```
p4 pubkeys -s mac-laptop
```

Related Commands

To create an SSH public key	`p4 pubkey`

p4 pull

Retrieve metadata or versioned files from a Helix server master server to a replica, or display status information about pending transfers, although there is a special syntax for a commit server to pull from a replica. In most situations, server replication with `p4 pull` is preferable to `p4 replicate`.

Syntax for a replica to pull from a commit server

```
p4 [g-opts] pull [-J prefix] [-i interval] [-b interval] [-T
excluded_tables] [-P serverid]
p4 [g-opts] pull -u [-i interval -b interval --batch=number
                    --min-size=number  --max-size=number --
trigger]
p4 [g-opts] pull -l [-s | -j [-J prefix]]
p4 [g-opts] pull -d -f file -r revision
p4 [g-opts] pull -L [-i interval]
p4 [g-opts] pull -R [file]
p4 pull -T
```

Syntax for a commit server to pull from a replica

`p4 pull -u -t target [-i interval -b interval]`

Description

The `p4 pull` command provides syntax variants, such as:

Description
p4 [*g-opts*] pull [-J *prefix*] [-i *interval*] [-b *interval*] [-T *excluded_tables*] [-P *serverid*] retrieves **journal** records from a target server specified by `P4TARGET`
p4 [*g-opts*] pull -u [-i *interval* -b *interval* --batch=*number* --min-size=*number* --max-size=*number* --trigger] retrieves **file contents** from a target server specified by `P4TARGET`

Description
p4 [*g-opts*] pull -l [-s | -j [-J prefix]] displays information about scheduled file transfers
p4 [*g-opts*] pull -d -f *file* -r *revision* cancels a scheduled file transfer
p4 [*g-opts*] pull -L [-i *interval*] specifies that journal records be retrieved from a **local** journal file (produced by the `p4 journalcopy` command) rather than from the journal file of the target server. These records are then written to the replica's database. **For a standby replica for failover**.

Except for testing purposes, `p4 pull` is rarely run from the command line. Instead, set the `"startup.N" on page 816` configurable to start the `p4 pull` processes every time the replica server starts.

Note
When you stop either the master server or a replica server, the replica server tracks the most recent journal position in a small text file called the state file. By default, the state file is named `state` and resides in the replica server's root directory. You can specify a different file name by setting the `statefile` configurable with `p4 configure`.

Retrieving journal and file content

The `p4 pull` command instructs the current replica server to retrieve either journal records or file contents from a target server specified by `P4TARGET`. Some replica servers do not need both journal records and file contents: for example, if you are creating a replica to help with offline checkpointing, you do not need to transfer file contents.

To replicate both metadata and file contents, you must run at least two `p4 pull` commands: one `p4 pull` (without the `-u` option) to replicate the master server's metadata, and at least one `p4 pull` (with the `-u` option) to replicate the server's versioned files.

- The `-i` option specifies a polling interval (in seconds) between updates. If `-i` is not specified, `p4 pull` runs for one polling interval and then exits.
- The `-b` option specifies a wait time after a failed pull attempt. If `-b` is not specified, `p4 pull` retries after 60 seconds.
- The `-u` option specifies that file content should be retrieved. If this option is not specified only journal records are fetched.
- The `--batch` option specifies the number of files a pull thread should process in a single request. The default value of 1 is usually adequate. For high-latency configurations, a larger value might improve archive transfer speed for large numbers of small files. (Use of this option requires that both master and replica be at version 15.2 or higher.)

Use the `-T` option to exclude tables you do not want to replicate. For example a build farm server does not need to replicate the db.have, `db.working`, or db.resolve tables.

To delete a pending file transfer operation, use **`p4 pull -d -f`** ***`file`*** **`-r`** ***`rev`***. This can be useful if a pending file transfer is failing repeatedly due to unrecoverable errors on the master.

> **Note**
> Setting the `"rpl.compress" on page 795` configurable allows you to compress journal record data that is transmitted using **`p4 pull`**.

Getting status information

Use the `-l` option to display a list of files that are scheduled for transfer. If `-s` is specified along with `-l`, a summary of scheduled file transfers is displayed. An additional line specifies the oldest changelist number that has at least one pending transfer. This provides a clue about how far the replica is lagging in its transfer of archive content.

An operator may run the **`p4 journalcopy -l`**, **`p4 pull -l -j`**, and **`p4 pull -l -s`** commands. This makes it possible for an operator to confirm the state of a replica.

```
File transfers: n active/m total, bytes: nnn active/mmmmm total.
Oldest change with at least one pending file transfer: n
```

For example, the following output:

```
File transfers: 1 active/63 total, bytes: 745 active/23684 total.
```

Tells us that there are 63 pending archive file transfers, one of which is currently active; and there are 23,684 bytes needed to be transferred of which 745 are currently actively being transferred.

If `-j` is specified with `-l`, report the current journal state at the current replica and its master, the last time the state file was modified, and the server’s local time and time zone. For example:

```
Current replica journal state is: Journal jjj, Sequence: sssss.
Current master journal state is: Journal jjj, Sequence: sssss.
The statefile was last modified at: 2012/01/10 14:23:23.
The Server time is currently: 2012/01/10 14:23:23 -0800 PST
```

The value of ***`jjj`*** specifies a journal number; ***`sssss`*** specifies an offset in that journal.

Options

`-b` ***`interval`***	Specify a polling interval in seconds for retries after failed retrieval attempts. If you do not specify this option, the pull is retried after 60 seconds.

`-u`	Transfer archive files instead of journal records. If you omit this option, the command retrieves journal records. Multiple `p4 pull -u` commands can be active on a single replica server.
`--batch` ***`number`***	Use this option to specify the number of files a pull thread should process in a single request. For high-latency configurations, providing a larger value than the default might improve archive transfer speed for large numbers of small files. Default: `1`
`--min-size` ***`number`***	The `--min-size` and `--max-size` options: ■ must be used with the `-u` and `--batch` ■ can be used with pull commands that create different pull threads for files with different sizes Pull threads called with these options pull files within the data size range specified with these options. The default size unit is bytes, but K, M, G, and T modifiers can also be used, such as 2K. See Example for min and max sizes.
`--max-size` ***`number`***	
`-d-f` ***`file`***`-r` ***`rev`***	Cancel a pending file content transfer, where ***`file`*** and ***`rev`*** refer to a depot file and a specific revision. **Note** This is not the normal Helix server file and revision data, but rather the archive file and revision. Use the `p4 pull -l` command to get the correct file name and revision.
`-i` ***`interval`***	Specify a polling interval in seconds for content retrieval. The smallest interval is one second. If you omit this option, the command runs once and exits. If you set the interval to be `0`, the master server advises the replicate as soon as new data becomes available. This way the replicated server can pull new data with no delay.
`-J` ***`prefix`***	Specify a prefix for the rotated journal file; overrides `journalPrefix` configurable. If your master server uses a non-default rotated journal location, this allows you to specify the rotated journal file location on the master server.

`-l`	List files that are scheduled for transfer. If you use this option on an edge server or build server that has `"lbr.replication" on page 758`**`=cache`** set, you might see several entries because of parallel file transfers. **`p4 pull -l`** (and **`p4 pull -ls`**) can be done on a commit server to monitor reverse replication as a result of submits to edge servers.
`-l -j`	Display the current journal state on the replica and the master. During the process of journal rotation on the master, the output of **`p4 pull -l -j`** can have three lines of output: one for the replica journal's current state, one for the state of the corresponding journal on the master, and a third line for the new journal on the master, data from which has not yet arrived at the replica.
`-l -s`	Display a summary of scheduled file content transfers. If this list is unexpectedly long or is growing, you might consider running additional **`p4 pull -u`** commands.
`-L`	Retrieve journal records from a local journal file, normally produced by the `"p4 journalcopy" on page 294` command.
`-P` ***`serverid`***	Filter data from ***`serverid`*** according to the **`ArchiveDataFilter:`** and **`ClientDataFilter:`** and **`RevisionDataFilter:`** fields in the specified server's **`p4 server`** form. In older releases, this option confirmed filters defined in the filter spec. This confirmation is no longer required. The option is retained for continued support of earlier releases. It can also be useful if you want to share filter configuration among multiple servers. In this case, the ***`serverid`*** refers to the server whose filter definitions you want shared. **Note** For compatibility with earlier releases of Helix server, you can also supply filter patterns directly within this field by using the same syntax used by the **`p4 export`**, but specifying a server and using fields in the **`p4 server`** form is strongly encouraged, because the behavior of a replica that makes use of multiple **`p4 pull`** commands with inconsistent or conflicting **`-P`** ***`filterpattern`*** arguments is undefined.
`-R`	Retry failed archive file transfers. While the **`-d`** option may be used to remove pending or failed file transfers individually, this option will instead reset the retry count of failed transfers so that subsequent pull operations can retry the transfers.

`-T` *`excluded_tables`*	Supply a list of database tables (for example, `db.have` and `db.working`) to exclude from the replica's journal records. The table names must begin with `db.`, following the naming convention used for database files in the server root directory. To specify multiple tables, double-quote the list and separate the table names with spaces. Table names can also be separated by commas. For example, `-T db.have,db.working` or `-T "db.have db.working"`.
`--trigger`	The `--trigger` option is used with a pull-archive trigger to transfer files using an alternative file transfer mechanism from within the trigger. This option is only used in a multi-server environment and is not supported for RCS storage. The configurable "pull.trigger.dir" on page 792 must be set to a location to write temporary files. It is also recommended to set the configurable "lbr.replica.notransfer" on page 758to `=1` to suppress "on demand" file transfer.
`-t` *`target`*	On the commit server, if `p4 pull -l` indicates that the commit server is not able to pull the archives from the edge, issue the following command manually: `p4 pull -u -t` *`target`* where: ■ `-u` enables archive transfer ■ *`target`* specifies the *`ExternalAddress`* field of the server spec of the edge. (See "p4 server" on page 493)
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

For more about configuring Helix server to run in a replicated environment, see Replication in the *Helix Core Server Administrator Guide*.

Example for min and max sizes

```
startup.2=pull -u -i 1  --batch=1000 --min-size=1 --max-size=2047
startup.3=pull -u -i 1  --batch=5 --min-size=2048 --max-size=4096
startup.4=pull -u -i 1  --batch=5 --min-size=4097
```

Related Commands

To configure a Helix server to run a set of **`p4 pull`** commands upon startup.	**`p4 configure`**
To replicate metadata from one server to another	**`p4 replicate`**
To display journal or checkpoint records in raw form	**`p4 export`**
To copy journal data to a replica's local file system.	**`p4 journalcopy`**

p4 push

Copy submitted files in your local server to a remote server.

Note
For distributed version control. See *Using Helix Core Server for Distributed Versioning* (DVCS).

"Syntax" on page 19

p4 [*g-opts*] push [-n -v] [-r *remotespec*] [-Ox] [-S *stream* | *filespec*]

p4 [*g-opts*] push [-n -v] [-r *remotespec*] [-Ox] -s *shelf*

Description

The **p4 push** copies the following items from the specified local server to the remote server:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes associated with the changelists, but only if the job that is associated with the fix is already present in the remote server. If it is not, then the fix is not copied.
- all integration records that describe integrations to the files being pushed

A push is only allowed if the files being pushed fit cleanly into the remote server, building precisely on a shared common history. If there are any conflicts or gaps, the push is rejected. Otherwise, the changelists become new submitted changelists in the remote server.

The second form of the command pushes a shelved changelist, rather than one or more submitted changelists, which avoids conflicts; the result is a new shelved change in the remote server.

When the changelists are added to the remote server, they are given newly assigned change numbers but they retain the same description, user, date, type, workspace, and set of files.

When the files are added to the remote server, they are kept in their same changelists, as new revisions starting after the current head. The new revisions retain the same revision number, file type, action, date, timestamp, digest, and file size.

Although the changelists are new submitted changelists in the remote server, none of the submit triggers are run in the remote server.

"Syntax" on page 19

Note that, once a particular revision of a file has been copied to another server, using **`p4 attribute -f`** to change the attributes on that revision will only affect the revision on that server, not on any other server to which it may have been copied.

Typically, the **`p4 push`** command specifies a remote spec, and the **`DepotMap`** field in the remote spec specifies which files are to be pushed. The **`p4 push`** command may also specify a filespec argument to further restrict the files to be pushed. If the remote spec uses differing patterns for the local and remote sides of the **`DepotMap`**, the filespec argument, if provided, must specify the files using the local filename syntax.

If a particular changelist includes some files that match the filespec, and other files that do not, then only the matching files are included in the push. To ensure that a partial changelist is not pushed, an appropriate filespec should be specified (for example, **`//...@change,#head`**).

The integration records describing integrations to the files being pushed are adjusted in the remote server to reflect the resulting changelist numbers and revision numbers of the remote server.

To push a set of files, you must have read access to those files in the local server, and you must have write access to those same files in the remote server (according to the remote server's protections table). Your local userid is also used as the userid at the remote server and you must already be logged in to both servers prior to running the **`p4 push`** command.

The **`p4 push`** command is atomic: either *all* the specified files are pushed, or *none* of them are pushed.

The value of the `"rpl.checksum.change" on page 794` configurable determines the level of verification performed for the **`p4 push`** command. See "Configurables" on page 715.

Triggering on pushes

The following push trigger types may be invoked during the execution of the **`p4 push`** command:

- The **`push-submit`** trigger can customize processing during the phase of the **`p4 push`** command when metadata has been transferred but files have not yet been transferred.
- The **`push-content`** trigger can customize processing during that phase of the **`p4 push`** command when files have been transferred but their contents have not yet been committed.
- The **`push-commit`** trigger can do any clean up work or other post processing after changes have been committed by the **`p4 push`** command.

For more information, see the section Triggering on pushes and fetches in the scripting chapter of *Helix Core Server Administrator Guide*.

Options

If the **`-r`** option is not specified, **`p4 push`** pushes files to the remote server named origin.

By default, changes cannot be pushed from server to server; in order to push changes between servers, an administrator of each server must enable pushing. Set `server.allowpush` to `1` on the server which initiates the push; set `server.allowpush` to `2` on the destination server. Files with the filetype modifiers `+k`, `+l`, or `+S` have some special considerations. Files of type `+k` have their digests cleared when pushed. This means certain cross-server merge conflicts are not detected. To re-generate the digests in the remote server after the push, use `p4 verify`.

When pushing files of type `+l`, the new files are added to the remote server even if the files are currently open by a pending changelist in that server. When pushing files of type `+S`, old archives which exceed the specified limit are not purged by the `p4 push` command.

`-n`	Performs all the correctness checks, but does not push any files or changelists to the remote server.
`-Oc`	When set, the `p4 push` command outputs information about every changelist. The `-v` option must be set for this to take effect.
`-Of`	When set, the `p4 push` command outputs information about every file in every changelist. The `-v` option must be set for this to take effect.
`-Oi`	When set, the `p4 push` command outputs information about every integration in every file in every changelist. The `-v` option must be set for this to take effect.
`-r` *`remotespec`*	Specifies a remote spec which contains the address of the remote server, and a file mapping which is to be used to re-map the files when they are pushed to the remote server.
`-s`	Specifies a shelved changelist to be pushed, instead of one or more submitted changelists. For more information, see the section "Fetch and push a shelved changelist" in the "Fetching and Pushing" chapter of *Using Helix Core Server for Distributed Versioning*.
`-S` *`stream`*	Specifies a particular stream to push. If you specify a stream you cannot also specify a file or files.

`-v`	Specifies verbose mode, which provides diagnostics for debugging. You must opt in to verbose mode. With verbose mode turned on, you can refine and control the precise level of verbosity. Specifically, you can indicate whether you want information about: ▪ every changelist pushed (with the `-Oc` option) ▪ every file in every changelist pushed (with the `-Of` option) ▪ every integration of every file in every changelist pushed (with the `-Oi` option) You can specify any combination of these options. The default is to display information about every changelist.
`filespec`	Specifies which file or files to push. If you specify a file or files you cannot specify a stream with the `-S` option.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`read` on the local server, `write` on the remote server.

Examples

`p4 push -r bruno-remote`	Push a file or files that are specified in the remote spec.

Related Commands

Copy files from a remote server into your local server	`p4 fetch`

p4 reconcile

Open files for add, delete, or edit to reconcile a workspace with changes made outside of Helix server. You might need to use this command after working offline from Helix server.

`p4 rec` is a synonym for `p4 reconcile`.

"Syntax" on page 19

```
p4 [g-opts] reconcile [-c changelist] [-a -d -e -f -I -k -l -m -n
-t -w] [file ...]
```

Description

If the `p4 reconcile` command finds unopened files in a user's workspace and detects inconsistencies between the workspace and the depot, it takes the following actions:

Inconsistency	Action
Files present in the depot, present in your `have` list, but missing from your workspace	Open for `delete`.
Files present in your workspace, but missing in the depot	Open for `add`.
Files modified in your workspace that are not open for edit	Open for `edit`.
Files opened for delete and present in your workspace without pending resolve records	Reopen for `edit`.
Files that are opened for edit but missing from the client	Reopen for `delete`.
The list of files to be opened includes both adds and deletes	If the file sizes and contents are similar, the missing and added files are compared and converted to pairs of `move/delete` and `move/add` operations.

The following table gives a use case for certain options:

To ...	Use ...
limit the scope of `p4 reconcile` to add, edit, or delete	`-a`, `-e`, or `-d` options
update the `have` list if files are mapped in a client's workspace to files in the depot that are not on the `have` list	`-k` option
preview the set of proposed workspace reconciliation actions	`-n` option
improve performance when reconciling changes to large files	`-m` option
override the default behavior and ignore the `P4IGNORE` file	`-I` option
consider whether the file type has changed (see "File types" on page 707	`-t` option

Reconcile and implicit p4 move affects p4 status

If you do a `p4 reconcile` and some of the files show up as potential "adds" while others show up as "deletes", Helix server compares the new files and the missing files to see if any of them appear to be the same file. If so, Helix server links them as if you had used "p4 move" on page 377 to open them. The output of `p4 reconcile` and "p4 status" on page 530 will include a "`... moved from`" line so that you can preview how Helix server matched the files before you submit them. You can also use "p4 resolved" on page 472 after opening the files.

Options

`-a`	Add files: Find files in the workspace that are not under Helix server control and open them for add.
`-c` *`changelist`*	Open the files to be reconciled in the specified pending changelist. If you omit this argument, the files are opened in the default changelist.
`-d`	Delete files: Find files missing from the workspace but present in the depot. Open these files for delete, but only if these files are in the user's `have` list. (See "p4 have" on page 247.)
`-e`	Edit files: Find files in the workspace that were been modified outside of Helix server, and open them for edit.
`-f`	Add filenames that contain special (wildcard) characters. Files containing the special characters @, #, %, or * are reformatted to encode the characters using hex notation. After these files are added, you must refer to them using their reformatted filenames.
`-I`	Do not perform any ignore checking, which means to ignore any settings specified by "P4IGNORE" on page 660.

`-k`	Update the `have` list if files are mapped in a client's workspace to files in the depot that are not on the `have` list.
`-l`	Display output in local file syntax with relative paths, similar to the workspace-centric view of "p4 status" on page 530.
`-m`	Compare the file sync or submit time (in the depot) with the file **modification time** (in the workspace) to help determine whether the file has changed. Normally Helix server uses file digests to determine whether files in the workspace differ from the head revisions of these files in the depot. This can be time consuming for large files. But when the timestamps are the same, the costly digest comparisons can be skipped. This option is only relevant if you are using **reconcile** to find changed files rather than files that were deleted or added.
`-n`	Preview the results of the operation without performing any action.
`-t`	Allows the user to reconcile files that are in the user's directory that had their file type changed. The reconcile command opens these files for edit even if the content is unchanged. See "File types" on page 707.
`-w`	Forces the workspace files to be updated to match their corresponding latest synced versions from the depot. Workspace files that are not in the depot are deleted. Files that are modified or deleted in the workspace will be replaced with their corresponding versions in the depot. This operation will result in the loss of any changes made to unopened files. The use of **p4 reconcile** with this option is the same as using the "p4 clean" on page 97 command. For other options when using **p4 reconcile** with the `-w` option, see "p4 clean" on page 97. This option requires read permission.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

- The `p4 reconcile` command produces output in depot syntax. To see file names and paths in local syntax, use the `-l` option with `p4 reconcile` or use `p4 status`.

- When called without arguments, `p4 reconcile` opens the files in a changelist. To preview an operation, use the `-n` option with `p4 reconcile` or use `p4 status`.

Related Commands

A shortcut for `p4 reconcile -n`	`p4 status`
A shortcut for `p4 reconcile -ead`	`p4 status -A`

p4 reconcile (graph)

Open files for add, delete, or edit to reconcile a workspace with changes made outside of Helix server.

"Syntax" on page 19

```
p4  reconcile [-a -e -d -n] [file ...]
```

Description

If the `p4 reconcile` command finds unopened files in a user's workspace and detects inconsistencies between the workspace and the depot, it takes the following actions:

Inconsistency	Action
Files present in the depot, present in your `have` list, but missing from your workspace	Open for `delete`.
Files present in your workspace, but missing in the depot	Open for for `add`.
Files modified in your workspace that are not open for edit	Open for `edit`.

Options

`-a`	Add files: Find files in the workspace that are not under Helix server control and open them for add.
`-d`	Delete files: Find files missing from the workspace but present in the depot. Open these files for delete, but only if these files are in the user's `have` list. (See "p4 have" on page 247 and "p4 have (graph)" on page 249)

`-e`	Edit files: Find files in the workspace that were been modified outside of Helix server, and open them for edit.
`-n`	Preview the results of the operation without performing any action.

p4 reload

Reloads the specified workspace, label, or task stream from the unload depot.

"Syntax" on page 19

```
p4 [g-opts] reload [-f] [-c client | -l label | -s stream] [-p
address]
```

Description

The **`p4 reload`** command reloads the state of an unloaded workspace (or the files tagged by an unloaded label, or stored in an unloaded task stream) from the unload depot into the versioning service's **`db.have`** (or **`db.label`**) tables.

Use **`-c`** ***`workspace`*** to reload an unloaded workspace, **`-l`** ***`label`*** to reload an unloaded label, or **`-s`** ***`stream`*** to reload an unloaded task stream. Helix server administrators can use the **`-f`** option to reload workspaces and/or labels owned by other users.

You can use the **`-c`** and **`-p`** options to migrate your unlocked workspace from one edge server to another without unloading the client first. The **`p4 reload`** command automatically issues the **`p4 unload`** command and waits for it to complete before reloading your workspace in the new edge server.

Options

`-c` ***`client`***	Reload the specified client workspace from the unload depot.
`-f`	Administrator force option; allows reloading of labels and workspaces owned by other users. Requires **`admin`** access.
`-l` ***`label`***	Reload the specified label from the unload depot.
`-p` ***`address`***	In multi-server environments, the **`-p`** option can be used to reload an unloaded client workspace from the remote edge server specified by ***`address`***. This migrates that workspace from the remote edge server to this one. To perform this operation, each edge server's service user must be properly authenticated to the other edge server. The ***`address`*** parameter can be specified either as the **`P4PORT`** or as the server id of the remote server. If you specify a server id, the server spec must contain the correct **`P4PORT`** value in its **`Address`** field.

`-s` *`stream`*	Reload the specified task stream from the unload depot.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`write`

- To reload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough (in the `p4 group` form) that users do not need to ask for assistance with `p4 unload` or `p4 reload` operations.

Related Commands

To unload a client workspace or label name	`p4 unload`

p4 remote

Create, modify or delete a remote specification.

Note
For distributed version control. See *Using Helix Core Server for Distributed Versioning* (DVCS).

"Syntax" on page 19

```
p4 [g-opts] remote [-f] remoteID
p4 [g-opts] remote -d [-f] remoteID
p4 [g-opts] remote -o remoteID
p4 [g-opts] remote -i [-f]
```

Description

A *remote* describes the shared server that your server cooperates with. The **`p4 remote`** command lets you configure your system to use the "p4 fetch" on page 194 and "p4 pull" on page 420 commands to copy work between your server and the shared server. A remote specification describes the high level configuration and usage of a remote. The **`p4 remote`** command allows you to create, modify, or delete a remote specification.

Note
These remotes have nothing to do with the Helix server construct of remote depots.

The **`p4 remote`** command puts the remote specification (*spec*) into a temporary file and invokes the editor configured by the `"P4EDITOR" on page 656` environment variable. Saving the file creates or modifies the remote spec.

A remote spec contains the following fields:

- **`RemoteID`**: The identifier of the remote.
- **`Address`**: The **`P4PORT`** that is used by the server.
- **`Owner`**: The user who created this remote spec. Can be changed.

  The specified owner does not have to be a Helix server user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
- **`RemoteUser`**: Specifies the identity (user) Helix server uses to authenticate against this remote server when pushing and fetching.

"Syntax" on page 19

- `Update`: The date this remote spec was last modified.
- `Access`: The last time this remote was used to fetch or push.
- `Description`: A description of the remote spec (optional).
- `Options`: Flags to change the remote spec behavior. The defaults are marked with `*`.
  - locked/*unlocked Permits only the owner to change the remote, and prevents the remote spec from being deleted.
  - compress/*nocompress Compresses data sent between the local and remote server to speed up slow connections.
  - *copyrcs/nocopyrcs During a **`p4 fetch`** or **`p4 push`**, transfers entire Revision Control System (RCS) archive files when possible, or never transfers entire RCS archive files. By default, **`p4 fetch`** and **`p4 push`** copy RCS archive files from and to the shared server as a unit, retrieving multiple revisions with a single file transfer. Sometimes, this optimization is undesirable, because when entire RCS archive files are copies, the archive change numbers are copied as well. This can cause your personal server to experience a "jump" in changelist numbers when changes are fetched from or pushed to the shared server.
- `LastFetch`: The last changelist that was fetched.
- `LastPush`: The last changelist that was pushed.
- `DepotMap`: Mapping between the local and remote files.
- `ArchiveLimits`: One or more entries specifying how many revisions of file archives to store locally when the files are fetched. See the *Using Helix Core Server for Distributed Versioning* topics on:
  - ArchiveLimits: entries, which shows an example of ArchiveLimits referencing the local (receiving) server path
  - Configure server to limit storage of archive revisions

Options

With no options specified, **`p4 remote`** invokes your editor for the specified remote spec.

`-d` ***`remote`***	Deletes the named remote.
`-f`	Enables a user with *admin* privileges to delete the spec or set the last modified date. By default, specs can be deleted only by their owner.
`-i`	Causes a remote spec to be read from the standard input. The user's editor is not invoked.
`-o` ***`remote`***	Writes the remote spec for the named remote to standard output. The user's editor is not invoked.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`, or `list` to use the `-o` option or `admin` to use the `-f` option

Examples

`p4 remote -i`	Read in a remote spec from standard input.

Related Commands

To display a list of remote specifications	`p4 remotes`

p4 remotes

Display a list of remote specifications.

> **Note**
> For distributed version control. See *Using Helix Core Server for Distributed Versioning* (DVCS).

"Syntax" on page 19

```
p4 [g-opts] remotes [[-e|-E] namefilter] [-m count]
```

Description

Use this command to display a list of remote specifications.

> **Note**
> These remotes have nothing to do with the Helix server construct of remote depots.

Options

With no options specified **`p4 remotes`** lists all remote specifications defined on this server.

`-e` ***`namefilter`***	Lists remote specs with a name that matches the ***`namefilter`*** pattern. For example: `$ p4 remotes -e svr-dev-rel*` The **`-e`** option uses the server's normal case-sensitivity rules.
`-E` ***`namefilter`***	Performs the same operation as the **`-e`** option, but makes the matching case-insensitive even on a case-sensitive server.
`-m` ***`count`***	Limits output to the specified number of remote specs.
`-u` ***`user`***	Lists remote specs owned by the specified user.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Examples

`p4 remotes -m 5`	List up to 5 remote specs.

Related Commands

To create, modify or delete a remote specification	`p4 remote`

p4 rename

Renaming files under Helix server control.

"Syntax" on page 19

```
p4 [g-opts] rename [-c change] [-f -n -k] [-t filetype]
fromFiletoFile
```

Description

The command `p4 rename` is an alias for `p4 move`.

p4 renameuser

Rename a user and modify all database records that mention the user.

"Syntax" on page 19

```
p4 [g-opts] renameuser [-f] --from=old --to=new
```

Description

The **p4 renameuser** command renames a user and modifies the following elements to reflect this change:

- the user record
- groups that include the user
- properties that apply to the user
- objects owned by the user: workspaces, labels, branches, streams, and so forth
- objects created by the user: all pending, shelved, and committed changes
- files the user has opened or shelved
- fixes the user made to jobs

The user name is not changed in descriptive text fields (such as job descriptions or change descriptions). It is only changed where the name appears as the owner or user field of the database record.

Protection table entries that apply to the user are updated only if the **Name** field exactly matches the user name. If the **Name** field contains wildcards, it is not modified.

The only job field that is processed is attribute code 103. If you have included the user name in other job fields, they will need to be changed manually.

The **p4 renameuser** command does not modify anything in the spec depot.

Warning

- If you are renaming a user who is being authorized by means of a **P4AUTH** configuration, you must issue the **p4 renameuser** command for every server that the user is authorized to use.
- For Perforce Swarm, any reference to a renamed user will be broken. For a potential workaround, contact Perforce Technical Support.

Usage and Limitations

For best results, follow these guidelines:

- Before you use this command, check to see that the user name you want to specify for `new` does not already exist. Using an existing name might result in the merging of data for the existing and the renamed user despite the best efforts of the system to prevent such merges.
- The user issuing this command should not be the user being renamed.
- The user being renamed should not be using the server when this command executes. After the command completes, the user should log out and then log back in.
- The `p4 renameuser` command does not process unloaded workspaces: all the user's workspaces should be reloaded (or deleted) first.

  A multi-serverinstallation might contain local workspaces or local labels owned by the user. These workspaces and labels, which are bound to Edge Servers, should be deleted or moved to the Commit Server first.

  If a central authentication server has been configured using `P4AUTH`, the user must be renamed in both servers, using separate invocations of `p4 renameuser`. The commands may be run in either order.
- If the user submitted files of type `+k` that contain the `$Author$` tag, those files will have incorrect digests following this command. Use `p4 verify -v` to recompute the digest value after the rename.

Options

`--from=`*`old`*	The name of the old user.
`--to=`*`new`*	The name of the new user.
`-f`	Forces the command to execute without checking for accidental merge checks that might have happened if the new user had already been used in this server. If the new user name has never been used before, you can improve performance using this option.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Related Commands

To recompute digest values after the rename for certain files.	`p4 verify`
To create a user or manage user preferences.	`p4 user`
To list existing users.	`p4 users`

p4 reopen

Move opened files between changelists or change the files' type, or move the client's stream spec to a different changelist.

"Syntax" on page 19

```
p4 [g-opts] reopen [-c changelist] [-t filetype] file ...
```

```
p4 [g-opts] reopen [-c changelist] -So
```

Description

for files

`p4 reopen` has the following uses for files:

- To move an open file from its current pending changelist to pending changelist ***`changelist`***, use **`p4 reopen -c`** ***`changelist`*** **`file`**
- To move a file to the default changelist, use **`p4 reopen -c default`**
- To change the type of a file, use **`p4 reopen -t`** ***`filetype`***
  - If file patterns are provided, all open files matching the patterns are moved or retyped. The two options can be combined to move a file and change its type in the same operation.

for streams

`p4 reopen` has the following uses for a stream:

- To move an open stream from its current pending changelist to pending changelist ***`changelist`***, use **`p4 reopen -So -c`** ***`changelist`***
- To move a stream to the default changelist, se **`p4 reopen -So -c default`**

Options for files

`-c` *`changelist`*	Move all open files matching file pattern ***`file`*** to pending changelist ***`changelist`***. To move a file to the default changelist, use `default` as the changelist number. (See Examples below.)

`-t` ***`filetype`***	When submitted, store file as type ***`filetype`***. All subsequent revisions will be of that file type until the type is changed again. See "File types" on page 707 for a list of file types.
`g-opts`	See "Global options" on page 690.

Options for streams

`-c` ***`changelist`***	Move the open stream spec to pending changelist ***`changelist`***, where ***`changelist`*** can specify a changelist number, or **`default`** for the default changelist.
`-So`	Move the the stream spec only.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

Examples for files

`p4 reopen -t text+k //...`	Reopen all open files as text files with keyword expansion.
`p4 reopen -c 410 //depot/proj1/... //.../README`	Move all open files under directory **`//depot/proj1`** or that are named **`README`** to pending changelist **`410`**.
`p4 reopen -c default -t binary+S //....exe`	Move all open **`.exe`** files to the default changelist, overwriting older revisions of those files in the depot.

Examples for streams

`p4 reopen -So -c 411`	Move the stream spec of the client to changelist 411.
`p4 reopen -So -c default`	Move the stream spec of the client to the default changelist.

p4 replicate

Poll for journal changes on one Helix server for forwarding to another Helix server.

"Syntax" on page 19

```
p4 replicate [-j token] [-s statefile] [-i interval] [-k -x -R]
[-J prefix]
              [-T tables] [-o output] [command]
```

Description

This command polls for new journal entries from a Helix server, and either outputs them to standard output, or, if a ***command*** is specified, pipe the journal records to the ***command***, which is spawned as a subprocess.

Options

`-i` ***interval***	Specify a polling interval, in seconds. The default is two seconds. To disable polling (that is, to check once for updated journal entries and then exit), specify an ***interval*** of `0`.
`-j` ***token***	Specify a journal number or position token of the form ***journalnum/byteoffset*** from which to start replicating metadata. If this option is specified, it overrides any state file specification.
`-J` ***prefix***	Specifies a filename prefix for the journal, such as that used with **`p4d -jc`** ***prefix***.
`-k`	Keep the pipe to the ***command*** subprocess open between polling intervals.
`-o` ***savefile***	Specify a file for output. If a ***command*** subprocess is specified, both the subprocess and the specified savefile are provided with the output.
`-R`	The **`-R`** option causes **`p4 replicate`** to attempt reconnection to the server in the event of connection loss or serious error. A polling interval must be specified with **`-i`** ***interval***.
`-s` ***statefile***	Specify a state file which tracks the most recent journal position. You can also use the **`statefile`** configurable to specify the state file.
`-T` ***tables***	Supply a list of database tables (for example, **`db.have`**) to exclude from export.

`-x`	Exit the `p4 replicate` command when journal rotation is detected.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

- Use `p4 replicate` in situations where you need to replicate metadata (but not archived files), or when you need to perform filtering operations on metadata. In most situations, replication with `p4 pull` is preferable to `p4 replicate`.
- See Replication in the *Helix Core Server Administrator Guide*.

Related Commands

To update file content as well as journal records	`p4 pull`
To display journal or checkpoint records in raw form	`p4 export`

p4 repo (graph)

Create, edit, or delete a repo specification.

> **Note**
> For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] repo [-d | -i] [-f] reponame
```

```
p4 repo -o reponame
```

Description

> **Note**
> An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

Such an administrator uses this command to create, edit, or delete a repo that stores files from Git users.

By default, the editor configured by the `$P4EDITOR` environment variable displays the repo specification, which you can save immediately, or after making edits.

To create a repo, you must be an admin, be the owner of the graph depot, or have the `create-repo` permission. For details about permissions, see `p4 grant-permission`.

> **Note**
> It is a best practice when you create a repo to set the default branch. See "Specify a default branch" in the *Helix4Git Administrator Guide*.

Options

`-d` ***`reponame`***	Delete the repo, including the files it contains. The name of a repo cannot be the same as the name of a branch, client workspace, or label. **Warning** The Helix server cannot retrieve a deleted repo.

`-f`	Enable any user with the `admin` privilege to force a change, such as to delete the repo or change its owner. By default, a repo can be deleted only by the owner.
`-i`	Read a repo specification from standard input.
`-o` *`reponame`*	Write a repo specification to standard output.
`g-opts`	See "Global options" on page 690.

Form Fields

Field Name	Description
`Name:`	The name of this repo.
`Owner:`	The user who created this repo.
`Created:`	The date this specification was created.
`Pushed:`	The date of the last `push` to this repo.
`ForkedFrom:`	The name of the repo from which this repo was forked.
`Description:`	A short description of the repo (optional).
`MirroredFrom:`	Upstream URL this read-only repo is mirrored from.
`DefaultBranch:`	The default branch to clone, such as `refs/heads/master`.
`gconn-serverId:`	The id of the gconn server that is configured to do mirroring for this repo.

Examples

To create a repo, `//graphDepot1/repo8`, within the depot of type `graph`, `graphDepot1`, that already exists:

```
p4 repo //graphDepot1/repo8
```

To create a repo directly from the command line, without having to use the spec editor:

```
p4 repo -o //graphDepot1/repo8 | p4 repo -i
```

To delete a repo named `//graphDepot1/repo8`:

```
p4 repo -d //graphDepot1/repo8
```

You will be prompted to use the `-f` option to force deletion:

```
p4 repo -f -d //graphDepot1/repo8
```

"Syntax" on page 19

Related Commands

To list the repos known to Helix server	`p4 repos`
To list all depots known to Helix server	`p4 depots`

p4 repos (graph)

Display a list of repos.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] repos [ [-e|-E] nameFilter] [-m maximum] [-u
userName] [-O ownerName] ]
```

Description

This command allows a user to get a list of the repos to which that user has at least the `read` permission. See `p4 grant-permission`.

The user can search the list by using an optional name filter.

The listing contains the name, owner, creation timestamp, and creator:

Name	Owner	Creation Timestamp	Creator
`//gamesGD1/Game1Repo1`	`bruno`	`2017/01/30 14:58:01`	`Created by bruno.`

Options

`-e` *`nameFilter`*	Lists repo specs with a name that matches the *nameFilter* pattern. `p4 repos -e //graphDepotName/repoName` See Examples. This option follows the case-sensitivity of the server.
`-E` *`nameFilter`*	Makes the matching case-insensitive, even on a case-sensitive server.
`-m`	Maximum: Limits output to the specified number of repo specs. For example, `p4 repos -m 5` displays the first five repos.

`-O`	Owner: Limits output to repos owned by the specified user or group. For example, **`p4 repos -O bruno -m 5`** displays the first five repos for which **`bruno`** is the owner.
`-u`	Limits output to those repos that can be read by the specified user or group.
`g-opts`	See "Global options" on page 690.

Examples

To determine which repos belong to depots where the depot name begins with "gra" and the repo name begins with "rep":

```
p4 repos -e //gra.../rep...
```

To determine which repos belong to depots where the depot name contains "d", and the repo name begins with "g" and contains "m", use the asterisk wildcard:

```
p4 repos -e "//*d*/g*m*"
```

To determine which repos have a name that ends with **`gameRepo1`**:

```
p4 repos -e "*gameRepo1"
```

To determine which repos have Bruno as owner and Gale as user with **`read`** permission:

```
p4 repos -O bruno -u gale
```

Related Commands

To create a repo	**`p4 repo`**

p4 reshelve

Copies shelved files from an existing shelf into either a new shelf or one that has already been created. This command does not require a client workspace.

"Syntax" on page 19

```
p4 [g-opts] reshelve [-p] -s changelist [file ...]
p4 [g-opts] reshelve [-f] [-p] -s changelist -c changelist [file
...]
```

Description

The **`p4 reshelve`** command copies shelved files from an existing shelf into either a new shelf or one that has already been created. This command leaves the source shelf intact.

If a file pattern is specified, **`p4 reshelve`** shelves the files that match the pattern.

Options

`-s` *`changelist`*	Specify the shelved changelist that contains the shelved files to be copied.
`-c` *`changelist`*	Specify the pending changelist that will be the target for the shelved files rather than creating a new one. To update a target shelf you must be the owner of the changelist.
`-f`	When the same file already exists in the target changelist, force the overwriting of it.
`-p`	Promote the new or target changelist where it can be accessed by other edge servers participating in the multi-server configuration. Once a shelved change has been promoted, all subsequent local modifications to the shelf are also pushed to the commit server and remain until the shelf is deleted.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`open`

For more information on promoted shelves, see "p4 shelve" on page 517.

Related Commands

To restore shelved files into a workspace	`p4 unshelve`
To shelve files	`p4 shelve`

p4 resolve

Resolve conflicts between file revisions, or resolve a stream spec conflict.

"Syntax" on page 19

```
p4 [g-opts] resolve [-a options] [-A options] [-d options] [-f  -
n  -N  -o  -t  -v]
                    [-c change] [file ...]
p4 resolve -So [ -af -am -as -at -ay -n -o ]
```

Description for files

Use `p4 resolve` to combine the contents of two files or file revisions into a single file revision in your workspace.

When `p4 resolve` is run with no file arguments, it operates on all files in the client workspace that have been scheduled for resolve.

Two situations require the use of `p4 resolve` before a file can be submitted:

- When a simple conflict exists: the revision of a file last synced to the client workspace is not the head revision at the time of the submit.

  For example, Alice does a `p4 sync` followed by a `p4 edit` of file `file.c`, and Bob does the same thing. Alice `p4 submit`s `file.c`, and then Bob tries to submit `file.c`. Bob's submit fails because if his version of `file.c` were to be accepted into the depot, Alice's changes to `file.c` would no longer be visible. Bob must resolve the conflict before he can submit the file.
- When `p4 integrate` has been used to schedule the integration of changes from one file (or branch) to another.

The primary difference between these two cases is that resolving a simple file conflict involves multiple revisions of a single file, but resolving for integration involves combining two separate files. In either case:

- If the file is of type `text`, `p4 resolve` allows the user to use the file in the client workspace instead of the file in the depot, overwrite the file in the client workspace with the file in the depot, or merge changes from both the depot revision and the client workspace revision into a single file.
- If the file is of type `binary`, only the first two options (use the file in the workspace, or overwrite the file in the workspace with the file in the depot) are normally available, because merges generally do not work with binary files.

"Syntax" on page 19

The output of **`p4 resolve`** is primarily diagnostic in nature; files are either resolved against ("vs") another file, copied, merged, edited, branched, added, deleted, moved, or ignored with respect to other files. The actual work performed by **`p4 resolve`** is reflected by the changes it makes to files in the client workspace.

Revisions Used to Detect Conflicts

The **`p4 resolve`** dialog refers to four file revisions whose meaning depends on whether or not the resolution fixes a simple file conflict or is resolving for integration:

Term	Meaning when Resolving Conflicts	Meaning when Resolving for Integration
yours	The revision of the file in the client workspace	The file to which changes are being propagated (in integration terminology, this is the *target* file). Changes are made to the version of this file in the client workspace, and this file is later submitted to the depot.
theirs	The head revision of the file in the depot.	The file revision in the depot from which changes are being propagated (in integration terminology, this is the *source* file). This file is not changed in the depot or the client workspace.
base	The file revision synced to the client workspace before it was opened for edit.	The previously-integrated revision of *theirs*. The latest common ancestor of both *yours* and *theirs*.
merge	A file version generated by Helix server from *yours*, *theirs*, and *base*. The user can edit this revision during the resolve process if the file is a text file.	Same as the meaning at left.

Resolve Options and Details

The interactive **`p4 resolve`** dialog presents the following options. Note that the dialog options are not the same as the command line options.

"Syntax" on page 19

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`e`	edit merged	Edit the preliminary merge file generated by Helix server.	no
`ey`	edit yours	Edit the revision of the file currently in the workspace.	yes
`et`	edit theirs	Edit the revision in the depot with which the workspace revision conflicts (usually the head revision). This edit is read-only.	yes
`dy`	diff yours	Show diffs between *yours* and *base*.	no
`dt`	diff theirs	Show diffs between *theirs* and *base*.	no
`dm`	diff merge	Show diffs between *merge* and *base*.	no
`d`	diff	Show diffs between *merge* and *yours*.	yes
`m`	merge	Invoke the command: `P4MERGE`*basetheirsyoursmerge* To use this option, you must set the environment variable `P4MERGE` to the name of a third-party program that merges the first three files and writes the fourth as a result. This command has no effect if `P4MERGE` is not set.	no
`?`	help	Display help for `p4 resolve`.	yes
`s`	skip	Don’t perform the resolve right now.	yes
`ay`	accept yours	Accept *yours*, ignoring changes that may have been made in *theirs*.	yes
`at`	accept theirs	Accept *theirs* into the client workspace as the resolved revision. The revision (*yours*) that was in the client workspace is overwritten. When resolving simple conflicts, this option is identical to performing `p4 revert` on the client workspace file. When resolving for integrate, this copies the source file to the target file.	yes

"Syntax" on page 19

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`am`	accept merge	Accept the *merged* file into the client workspace as the resolved revision without any modification. The revision (*yours*) originally in the client workspace is overwritten. **Note** Manually accepting a merge resolve (`p4 resolve` and selecting `am`) is different from automatic merge resolve (`p4 resolve -am`). When manually accepting a merge resolve, the user makes a conscious choice. `p4 resolve -am` automates the choice to accept *yours*, accept *theirs*, or accept *merge*, depending on the three-way merge differences. If additions and deletions are in: ▪ only the target file, the target differences are kept (the same as a manual accept *yours*) ▪ only the source file, the source differences are kept (the same as a manual accept *theirs*) ▪ both the source and target files with no conflicts, the merged result is kept (the same as a manual accept *merged*) ▪ both the source and the target files with conflicts, the merged result is skipped (the same as a manual *skip*)	no
`ae`	accept edit	If you edited the file (that is, by selecting “e” from the `p4 resolve` dialog), accept the edited version into the client workspace. The revision (*yours*) originally in the client workspace is overwritten.	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`a`	accept	Keep Helix server's recommended result: ▪ if *theirs* is identical to *base*, accept *yours*; ▪ if *yours* is identical to *base*, accept *theirs*; ▪ if *yours* and *theirs* are different from *base*, and there are no conflicts between *yours* and *theirs*; accept *merge*; ▪ otherwise, there are conflicts between *yours* and *theirs*, so skip this file	no

Resolution of a file is completed when any of the **`accept`** dialog options are chosen. To resolve the file later or to revert the change, **`skip`** the file.

To help decide which option to choose, counts of four types of changes that have been made to the file revisions are displayed by **`p4 resolve`**:

```
Diff Chunks: 2 yours + 3 theirs + 5 both + 7 conflicting
```

The meanings of these values are:

Count	Meaning
n **`yours`**	***n*** non-conflicting segments of *yours* are different than *base*.
n **`theirs`**	***n*** non-conflicting segments of *theirs* are different than *base*.
n **`both`**	***n*** non-conflicting segments appear identically in both *theirs* and *yours*, but are different from *base*.
n **`conflicting`**	***n*** segments of *theirs* and *yours* are different from *base* and different from each other.

If there are no conflicting chunks, it is often safe to accept Helix server's generated merge file, because Helix server will substitute all the changes from *yours* and *theirs* into *base*.

If there are conflicting chunks, the *merge* file must be edited. In this case, Helix server will include the conflicting *yours*, *theirs*, and *base* text in the *merge* file; it's up to you to choose which version of the chunk you want to keep.

The different text is clearly delineated with file markers:

```
>>>> ORIGINAL VERSION file
#n
```

```
<text>==== THEIR VERSION file
#m
<text>==== YOUR VERSION file
<text><<<<
```

Choose the text you want to keep; delete the conflicting chunks and all the difference markers.

Non-Content-Related Resolves

You can also resolve other types of conflicts between related files: filetype, deletion, branching, as well as moves and renames. See the "Resolve conflicts" chapter of the *Helix Core Server User Guide*.

To constrain the process to one type of resolve, use the `-A` option.

Option	What is Resolved
`-Aa`	Resolve attributes set by `p4 attribute`.
`-Ab`	Integrations where the source is edited and the target is deleted.
`-Ac`	Resolve file content changes as well as actions.
`-Ad`	Integrations where the source is deleted and target is edited.
`-Am`	Renames and moves.
`-At`	Filetype changes.
`-AQ`	Charset changes.

Each type of resolve is handled separately. For example, if a file has both a filetype conflict and a content conflict, you are prompted separately to specify how each is handled. To avoid file-by-file prompting when the desired outcome is the same for all resolves, include the `-at` or `-ay` option following the `-A` option. The following example illustrates how prompting is handled for different resolves.

```
Merging //depot/rel/fileb#1
Diff chunks: 1 yours + 0 theirs + 0 both + 0 conflicting
Accept(a) Edit(e) Diff(d) Merge (m) Skip(s) Help(?) ay: m
//depot/main/filez - resolve skipped.
Resolving move to //depot/main/fileb
Filename resolve:
at: //depot/main/fileb
ay: //depot/main/filez
```

"Syntax" on page 19

Description for a stream spec conflict

Suppose that two clients, A and B, both edit the same stream spec. Client A opens the stream spec for editing. Client B either opens the stream spec for editing and submits, or edits globally and saves changes to the stream spec. Client A's open stream spec is now in a conflict condition.

`p4 submit [-Si|-So|-Sx]` detects the conflict condition and prevents the opened stream from being submitted.

`p4 resolve –So [ -af -am -as -at -ay -n -o ]` allows for a resolve preview, or resolves by merging the changes. The resolve can ignore either the changes that are `theirs` or `yours`.

Options for files

`-a` *`options`* `-am` `-af` `-as` `-at` `-ay`	Skip the resolution dialog, and resolve the files automatically as follows: ■ `-am`: Automatic Mode. Automatically accept the recommended file revision: if *theirs* is identical to *base*, accept *yours*; if *yours* is identical to *base*, accept *theirs*; if *yours* and *theirs* are different from *base*, and there are no conflicts between *yours* and *theirs*; accept *merge*; otherwise, there are conflicts between *yours* and *theirs*, so skip this file. ■ `-ay`: Accept *Yours*, ignore *theirs*. ■ `-at`: Accept *Theirs*. Use this option with caution, as the file in the client workspace will be overwritten! ■ `-as`: Safe Accept. If either *yours* or *theirs* is different from base, accept that revision. If changes are identical, accept *theirs*. If both are different from base, skip this file. ■ `-af`: Force Accept. Accept the *merge* file no matter what. If the *merge* file has conflict markers, they will be left in, and you'll need to remove them by editing the file.

`-A options` `-Aa` `-Ab` `-Ac` `-Ad` `-At` `-Am`	Action (non-content) resolves: Constrain the type of resolve to branching, deletion, file type change, or move/rename. ■ `-Aa`: Resolve attributes set by `p4 attribute` ■ `-Ab`: Resolve file branching; that is, integrations where the source is edited and the target is deleted ■ `-Ac`: Resolve file content changes ■ `-Ad`: Integrations where the source is deleted and target is deleted ■ `-At`: Filetype changes ■ `-Am`: Move and renames For details, see the *Helix Core Server User Guide* and "Non-Content-Related Resolves" on page 463.
`-doption`	When merging files, ignore specified differences in whitespace or line-ending convention. (If you use these options, and the files differ by whitespace only, `p4 resolve` uses the text in the workspace file.) ■ `-db`: Ignore whitespace-only changes (for instance, a tab replaced by eight spaces) ■ `-dw`: Ignore whitespace altogether (for instance, deletion of tabs or other whitespace) ■ `-dl`: Ignore differences in line-ending convention
`-f`	Allow already resolved, but not yet submitted, files to be resolved again. **Tip** The content of the target (yours) file being re-resolved is the result of the previous resolve, not the content of the original file. To preserve the option of using the original file, revert the resolved file. See the "Examples for files" on page 467.
`-n`	List the files that need resolving without actually performing the resolve.
`-N`	Preview the operation with additional information about any non-content resolve actions that are scheduled.
`-o`	Output the base file name and revision to be used during the resolve.
`-t`	Force a three-way merge, even on binary (non-text) files. This allows you to inspect diffs between files of any type, and lets you merge non-text files if `P4MERGE` is set to a utility that can do such a thing.

`-v`	Include conflict markers in the file for all changes between yours and base, and between theirs and base. Normally, conflict markers are included only when yours and theirs conflict.
`-c` *`change`*	Limit the scope of the resolve operation to the files opened in the specified changelist number.
`g-opts`	See "Global options" on page 690.

Options for streams

The `-So` option resolves only opened stream spec conflicts. For example,

```
p4 resolve -So [ -af -am -as -at -ay -n -o ]
```

`-af`	Force the combination of text fields with conflicts.
`-am`	Resolve by merging; skip fields with conflicts.
`-as`	Safe resolve; skip fields that needs merging.
`-at`	Force acceptance of theirs; overwrite yours.
`-ay`	Force acceptance of yours; ignore theirs.
`-n`	Preview which fields require resolve.
`-o`	Output the base change (the `have` change) used for the merge.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

`p4 resolve` works only with files that have been scheduled for resolve.

Three operations schedule files for resolution:

Option	Description
Integrating the file with `p4 integrate` or `p4 merge`.	■ When scheduling files for resolve, `p4 integrate` selects the closest common ancestor as the base. ■ The `p4 merge` command selects the revision with the most edits in common with the source and target.

Option	Description
Submitting an open file that was synced from a revision other then the current head revision.	The submit fails, and the file is scheduled for resolve.
Running `p4 sync` instead of running `p4 submit` on the open file.	Nothing is copied into the client workspace. Instead, the file is scheduled for resolve. The benefit of scheduling files for resolve with `p4 sync` instead of a failed submit is that the submit will not fail.

Note
If translation errors occur during integrations between `text` and `unicode` files, the most likely cause is the presence of non-ASCII characters in the `text` file. Either remove the non-ASCII characters from the file before integration, or set `P4CHARSET` to `utf8` and attempt the merge again.

Examples for files

Re-resolving a file using the `-f` flag is not necessarily equivalent to reverting the resolved file and performing the resolve again. Suppose that in the initial resolve, you used the accept theirs (`-at`) option:

```
$ p4 resolve -at
$ /Users/bruno/dir8/dir2/fileA.txt - vs //depot/dir6/dir2/fileA.txt#2
//bruno/dir8/dir2/fileA.txt - copy from //depot/dir6/dir2/fileA.txt
```

But for re-resolving, you instead use the accept yours (`-ay`) option:

```
$ p4 resolve -f -ay
/Users/bruno/dir8/dir2/fileA.txt - vs //depot/dir6/dir2/fileA.txt#2
//bruno/dir8/dir2/fileA.txt - copy from //depot/dir6/dir2/fileA.txt
```

In this case, the depot version is copied into the client workspace instead of the depot version being ignored.

Example for a stream conflict

A conflict prevents an open stream edit from being submitted:

```
p4 -c ws0 submit -d "Conflict detected?" -So
Submitting change 5.
Stream //root/main is out of date; run 'p4 stream resolve'.
```

Stream resolve preview:

```
p4 -c ws0 resolve -So -n
//root/main Paths - resolving //root/main@4
```

Stream resolve with a conflict merge scenario, the attempt with `-am` fails but the attempt with `-af` succeeds:

```
p4 -c ws0 resolve -So -am
```

```
//root/main Paths - skipped //root/main@4
p4 -c ws0 resolve -So -af
//root/main Paths - combined with //root/main@4
```

The `Paths:` field now looks like:

```
Paths:
share a # open
share c # open
share b # open
```

Related Commands

To view a list of resolved but unsubmitted files	`p4 resolved`
To schedule the propagation of changes between two separate files	`p4 integrate`
To submit a set of changed files to the depot	`p4 submit`
To copy a file to the client workspace, or schedule an open file for resolve	`p4 sync`

p4 resolve (graph)

Resolve integrations and updates to repo workspace files.

"Syntax" on page 19

```
p4 resolve [options] [file ...]
```

Description

`p4 resolve` works only on files that have been scheduled to be resolved.

The commands that can schedule resolves are:

- " p4 sync (graph)" on page 581
- "p4 update" on page 615
- "p4 submit (graph)" on page 569

- "p4 merge (graph)" on page 368
- "p4 integrate" on page 265

Files must be resolved before they can be submitted. Resolving involves two sets of files, a source and a target. The target is a set of depot files that maps to opened files in the client workspace.

- When resolving an integration, the source is a different set of depot files than the target.
- When resolving an update, the source is the same set of depot files as the target, at a different revision.

The file argument specifies the target. If the file argument is omitted, all unresolved files are resolved.

Resolving can modify workspace files. The resolve process is a classic three-way merge. The participating files are referred to as follows:

- `yours` - The target file open in the client workspace
- `theirs` - The source file in the depot
- `base` - The common ancestor, which is the highest revision of the source file already accounted for in the target
- `merged` - The merged result

Filenames, filetypes, and text file content can be resolved by accepting `yours`, `theirs`, or `merged`.

Branching, deletion, and binary file content can be resolved by accepting either `yours` or `theirs`.

When resolving integrated changes, `p4 resolve` distinguishes among four results:

- entirely yours
- entirely theirs
- a pure merge
- an edited merge

The distinction is recorded when resolved files are submitted, and will be used by future commands to determine whether integration is needed.

In all cases, accepting `yours` leaves the target file in its current state.

The result of accepting `theirs` is as follows:

- Content: The target file content is overwritten.
- Attribute: The target's attributes are replaced.
- Deletion: The target file is deleted.
- Filename: The target file is moved or renamed.
- Filetype: The target file's type is changed.

For each unresolved change, the user is prompted to accept a result.

Content and non-content changes are resolved separately.

For content, `p4 resolve` places the merged result into a temporary file in the client workspace. If there are any conflicts, the merged file contains conflict markers that must be removed by the user.

`p4 resolve` is not supported for files with propagating attributes from an edge server in a multi-server environment.

Prompts during the diff operation

`p4 resolve` displays:

- a count of text diffs and conflicts.
- the following prompts, where the options marked (*) appear only for text files, and the suggested action is displayed in brackets:

Accept	
`at`	Keep only changes to their file
`ay`	Keep only changes to your file
`* am`	Keep merged file
`* ae`	Keep merged and edited file
`* a`	Keep autoselected file

Diff	
`* dt`	See their changes alone
`* dy`	See your changes alone
`* dm`	See merged changes
`d`	Diff your file against merged file

Edit	
`et`	Edit their file (read only)
`ey`	Edit your file (read/write)
`* e`	Edit merged file (read/write)

Misc	
`* m`	Run `$P4MERGE base theirs yours merged` (Runs `$P4MERGEUNICODE charset base theirs yours merged` if set and the file is a unicode file.)

`s`	Keep only changes to your file
`h`	Print this help message
`^C`	Quit the resolve operation

The `merge (m)` option enables you to invoke your own merge program if one is configured using the "P4MERGE" on page 667 environment variable. Four files are passed to the program:

- the base
- yours
- theirs
- the temporary file (The program is expected to write merge results to the temporary file)

Options

The `-a` flag puts `p4 resolve` into automatic mode. The user is not prompted, and files that can't be resolved are skipped:

`-a`	Automatic mode: the user is not prompted, and files that can't be resolved are skipped.
`-as`	Safe resolve skips any files that need merging. Causes the workspace file to be replaced with their file only if theirs has changed and yours has not.
`-am`	Resolve by merging skips any files with conflicts Causes the workspace file to be replaced with the result of merging theirs with yours. If the merge detected conflicts, the file is left untouched and unresolved.
`-af`	Causes the workspace file to be replaced with the result of merging `theirs` with `yours`, even if there were conflicts. This can leave conflict markers in workspace files.
`-at`	Force acceptance of `theirs` and overwrites `yours`, that is, overwrites any changes made to the file in the client workspace.
`-ay`	Resolves all files by accepting `yours` and ignoring `theirs`. This preserves the content of workspace files.
-n	Previews the operation without altering files

p4 resolved

Display a list of files that have been resolved but not yet submitted.

"Syntax" on page 19

```
p4 [g-opts] resolved [-o] [file ...]
```

Description

`p4 resolved` lists files that have been resolved, but have not yet been submitted. The files are displayed one per line in the following format:

```
localFilePath - action from depotFilePath#revisionRange
```

where ***`localFilePath`*** is the full path name of the resolved file on the local host, ***`depotFilePath`*** is the path of the depot file relative to the top of the depot, ***`revisionRange`*** is the revision range that was integrated, and ***`action`*** is one of `merge`, `branch`, or `delete`.

If file pattern arguments are provided, only resolved, unsubmitted files that match the file patterns are included.

Although the name `p4 resolved` seems to imply that only files that have gone through the `p4 resolve` process are listed, this is not the case. A file is also considered to be resolved if it has been opened by `p4 integrate` for `branch`, opened by `p4 integrate` for `delete`, or has been resolved with `p4 resolve`.

Options

`-o`	Output the base file name and revision that was used during the resolve.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open`

"Syntax" on page 19

Related Commands

To see a list of integrations that have been submitted	`p4 integrated`
To view a list of integrations that have not yet been resolved	`p4 resolve` **-n**
To schedule the propagation of changes from one file to another	`p4 integrate`
To resolve file conflicts, or to propagate changes as scheduled by `p4 integrate`	`p4 resolve`

p4 restore

Restore old archived revisions from an archive depot.

"Syntax" on page 19

```
p4 [g-opts] restore [-n] -D archiveDepot[revRange] ...
```

Description

The `p4 restore` command transfers archives from a named ***depot*** of type `archive` back to their original locations in a local depot. After being restored, the revisions' action is restored to whatever it was before it was archived.

Set the `"server.locks.archive" on page 809` configurable to disable server locks when running the `p4 restore` command.

Options

`-D` *depot*	Specify an archive depot from which files are to be restored.
`-n`	Do not restore files. Report on revisions that would be restored.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`admin`

- Storage for the archive depot must be mounted unless you are using the `-n` option.

Related Commands

To create a depot	`p4 depot`
To archive files into an archive depot	`p4 archive`
To obliterate files without archiving them	`p4 obliterate`

"Syntax" on page 19

p4 resubmit

Resolve and resubmit some or all unsubmitted changes.

Note
For distributed version control only:

- You **can** issue this command directly to a **commit** server.
- You **CANNOT** issue this command directly to an **edge** server.
- See *Using Helix Core Server for Distributed Versioning* (DVCS).

"Syntax" on page 19

```
p4 [g-opts] resubmit -l
p4 [g-opts] [-R] resubmit -m
p4 [g-opts] [-R] resubmit -e
p4 [g-opts] [-R] resubmit -i [[-r remote] filespec ...]
```

Description

The **`p4 resubmit`** command resubmits changes that have been unsubmitted. Use this command to revise a set of changelists that you have:

- submitted locally
- not pushed to any other server
- have unsubmitted

This command has three modes:

automatic	`-m`	Syncs your workspace to `#head` and processes each conflicting change. For details, see `-m` under Options.
partially interactive	`-e`	Prepares the first (oldest) conflicting change, then exits. The files for the change are open in your workspace. 1. Resolve or make any other necessary changes to the workspace files 2. Run `p4 resubmit -R -e` 3. This prepares the next conflicting change.
fully interactive	`-i`	If a file path is also specified, this option unsubmits each change that modified a file in that path. If the `-r` flag is also specified, it names a remote spec. The mapping in the remote spec limits the files affected by the unsubmit operation. Thus a command such as `p4 resubmit -r rmt @>=17` affects only the files specified by the remote spec. `resubmit -i` then processes each conflicting change.

Options

`-e`	Runs `p4 resubmit` in partially-interactive mode, allowing you to inspect each change prior to submitting it.

`-i`	Runs `p4 resubmit` as a fully interactive resubmission tool. You can reorder, combine, or discard changes. For each change, resubmit displays summary information about the change and prompts you to indicate which action to take:

`c`	Modify the change description for this change.
`m`	Merge this change, then submit if no conflicts.
`e`	Merge this change, then exit for further editing. After choosing either `e` or `q` you can continue interactive resubmit by running `resubmit -R -i`
`r`	Interactively resolve this change, then submit if no conflicts.
`a`	Add (squash) this change into the next conflicting change.
`s`	Skip this change and move on to the next.
`d`	Delete this change without submitting it.
`b`	Begin again from the earliest remaining change.
`l`	List the changes remaining to be processed.
`v`	View the current change in short form.
`V`	View the current change with full diffs.
`R`	Display the status of resolved and unresolved merges.
`q`	Quit the resubmit operation. After choosing either `e` or `q` you can continue interactive resubmit by running `resubmit -R -i`
`?`	Display (short) help during the resubmit command.

`-l`	Lists all the unsubmitted changes but takes no action. This is useful as a way to preview the work that must be resubmitted.

`-m`	`p4 resubmit -m` runs in automatic mode: 1. Syncs your workspace to `#head`. 2. Processes each unsubmitted change: • If the change is a tangent of unsubmitted work in the tanget depot created by `p4 fetch -t`: `sync` `integrate tangent/...@=change dest/...` `resolve -am` `submit` • If the change is a shelf of unsubmitted work created by `p4 unsubmit`: `sync` `unshelve -s change -c change` `resolve -am` `submit` If, for any change in the list, the `p4 resolve` -am processing detects merge conflicts in any file in that change, the `p4 resubmit` command terminates. All the files in that change which had merge conflicts are left unresolved until you do the following: 1. Run the `p4 resolve` command to resolve the conflicts. 2. Re-run `p4 resubmit -Rm` to resume the resubmit process (the first thing it does is submit the resolved files from this change, then it proceeds to the next change).
`-r` *`remote`*	When `p4 resubmit` is run with the `-i` option, the `-r` option specifies the remote spec whose mapping is used to limit the files affected by the unsubmit operation. For example: `$ p4 resubmit -r rmt @>=17` This example affects only the files specific by the remote spec, not all files in the depot.
`-R`	Resume the resubmit process once conflicts have been resolved. With this flag, resubmit begins by submitting the fully-resolved change and then proceeds to the next unsubmitted change.
`filespec`	When a filespec is provided with the `-i` option, the interactive resubmit first unsubmits each change that modified a file in that path.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`write`, or `admin` to use the `-i` option

Examples

`p4 resubmit -m`	Merges and resubmits your unsubmitted changes.

Related Commands

To unsubmit submitted changelists	`p4 unsubmit`

p4 revert

Discard changes made to open files or revert an open stream spec.

"Syntax" on page 19

```
p4 [g-opts] revert [-a -n -k -w] [-c change] [-C client] [--
remote=remote] file ...
```

```
p4 [g-opts] revert [-a -n -k -w] [-Si] [-c change] [-C client] [-
-remote=remote] file ...
```

```
p4 [g-opts] revert -So [-c change]
```

Description

Use **`p4 revert`** to discard changes made to open files, reverting them to the revisions last synced from the depot (with **`p4 sync`**). This command also removes the reverted files from the pending changelists with which they're associated. An administrator can use the **`-C`** option to revert another user's open files.

- When you revert files you opened with **`p4 delete`**, the files are reinstated in the client workspace.
- When you revert files that have been opened by **`p4 add`**, Helix server leaves the client workspace files intact.
- When you revert files you've opened with **`p4 integrate`**, Helix server removes the files from the client workspace.
- When you revert files you've opened with **`p4 move`**, only the file open for **`move/add`** can be reverted.

The host name is implied and the username is not needed.

streams and p4 revert

2019.1 introduced the **`-Si`** and **`-So`** options to revert an open stream spec.

By default, an open stream spec is not reverted.

Options for files

`-a`	Revert only those files that haven't changed (in terms of content or filetype) since they were opened. The only files reverted are those whose client revisions are the following: ■ Open for edit but have unchanged content and unchanged filetype. ■ Open for integrate via `p4 integrate` and have not yet been resolved with `p4 resolve`. ■ Open for add, but are missing from the workspace. Files that are open for add that are missing but which also have pending integrations will not be reverted.
`-c` *`change`*	Reverts only those files in the specified changelist. Revert all files in a specific change in a client workspace `$ p4 revert -c 345627 "//..."`

`-C` *`client`*	Revert another user's open files. This option implies the `-k`, `-H`, and `-u` options. `-H` and `-u` are global options.
	Revert a single file in a client workspace:
	`$ p4 revert -C bruno_ws //depot/www/dev/Jam.html`
	Revert all files in a client workspace:
	`$ p4 revert -C bruno_ws //...`
	This option is useful in freeing up exclusive locks held on an edge server. In such a case, the command needs to be directed to the edge server where the client resides. For example:
	`$ p4 revert -C SusanClient //depot/src/x.cc`
	The option is also very useful in cleaning up after old users' workspaces: you need to revert any open files in a workspace before you delete it.

`-k`	Keep workspace files; the file(s) are removed from any changelists and Helix server records that the files as being no longer open, but the file(s) are unchanged in the client workspace.
`-n`	List the files that would be reverted without actually performing the revert. This lets you make sure the revert does what you think it does before actually reverting the files.

`--remote=`*`remote`*	Reverts the file in your personal server, and additionally — if the file is of type `+l` — releases the global exclusive lock on the file in the shared server from which you cloned the file. For more information, see the section Support for exclusive locking in the Fetching and Pushing chapter of *Using Helix Core Server for Distributed Versioning*. For more information, see the section "Support for exclusive locking in personal servers" in the "Fetching and Pushing" chapter of *Using Helix Core Server for Distributed Versioning*.
`-w`	Files that are open for `add` are to be deleted (wiped) from the workspace when reverted.
`g-opts`	See "Global options" on page 690.

Options for streams

`-Si`	Include the open stream spec when reverting the specified list of files
`-So`	Only the open stream spec is reverted. No list of files allowed.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`

- `p4 revert` differs from most Helix server commands in that it usually *requires* a file argument. The files that are reverted are those that lie in the intersection of the command line file arguments and the client view.

  You don't need to specify a file argument when using the `-a` option.
- Reverting a file that has been opened for `edit` will overwrite any changes you have made to the file since the file was opened. It may be prudent to use `p4 revert -n` to preview the results before running `p4 revert`.

Examples for reverting streams

`p4 revert -Si ...`	Revert everything in the default changelist, including the current open stream spec.

`p4 revert -So`	Revert only the current open stream spec in the default changelist
`p4 revert -Si -c 987654321 ...`	Revert everything in the numbered changelist.
`p4 revert -So -c 987654321`	Revert only the open stream spec in the numbered changelist.
`p4 revert -Si foo`	Revert the current open stream spec and the file in the default changelist
`p4 revert -Si -c 987654321 foo`	Revert the current open stream spec and the file in the numbered changelist.
`p4 revert foo`	Revert only the file, not the stream, from a default change list.
`p4 revert -c 987654321 foo`	Revert only file, not the stream, from a numbered change list.

Related Commands

To open a file for add	`p4 add`
To open a file for deletion	`p4 delete`
To copy all open files to the depot	`p4 submit`
To read files from the depot into the client workspace	`p4 sync`
To list all opened files	`p4 opened`
To forcibly bring the client workspace in sync with the files that Helix server thinks you have, overwriting any unopened, writable files in the process.	`p4 sync -f`

p4 revert (graph)

Discard changes from an opened file.

"Syntax" on page 19

```
p4 [g-opts] revert [-a -n -w -c changelistNumber] file ...
```

Description

Revert an open file to the revision that was synced from the depot, discarding any edits or integrations that have been made. You must explicitly specify the files to be reverted. Files are removed from the changelist in which they are open. Locked files are unlocked.

Suppose the user invokes `p4 revert` to cancel the editing of a file prior to resolving that file,

If the file belongs to a ...	the file reverts to ...	because ...
classic depot	the depot revision you had in your workspace just before invoking "p4 edit" on page 180	In a classic depot, each file is tracked **individually**
graph depot	the file revision associated with the commit SHA of your workspace	In a graph depot, the" p4 sync" on page 574 operation represents a snapshot of a **collection** of files with a single commit SHA

Options

`-a`	The -a flag reverts only files that are open for edit, add, or integrate and are unchanged or missing. Files with pending integration records are left open. The file arguments are optional when -a is specified.
`-c` *change*	Reverts only those files in the specified changelist. `$ p4 revert -c 345627 "//..."`
`-n`	List the files that would be reverted without actually performing the revert. This lets you make sure the revert does what you think it does before actually reverting the files.
`-w`	Files that are open for `add` are to be deleted (wiped) from the workspace when reverted.

p4 review

List all submitted changelists above a provided changelist number.

"Syntax" on page 19

```
p4 [g-opts] review [-c changelist] [-t countername]
```

Description

`p4 review -c` ***`changelist`*** provides a list of all submitted changelists between ***`changelist`*** and the highest-numbered submitted changelist. Each line in the list has this format:

```
Change changelistusername <email-addr> (realname)
```

The ***`username`***, ***`email-addr`***, and ***`realname`*** are taken from the **`p4 user`** form for ***`username`*** whenever **`p4 review`** is executed.

When used as **`p4 review -t`** ***`countername`***, all submitted changelists above the value of the Helix server counter variable *countername* are listed. (Counters are set by **`p4 counter`**). When used with no arguments, **`p4 review`** lists all submitted changelists.

The **`p4 review`** command is meant for use in external programs that call Helix server, such as the change review daemon. The change review daemon is available from the Helix server Public Depot:

http://wiki.workshop.perforce.com/wiki/P4Review

and is documented in the *Helix Core Server Administrator Guide*.

Options

`-c` ***`changelist`***	List all submitted changelists above and including ***`changelist`***.
`-t` ***`countername`***	List all submitted changelists above the value of the Perforce counter ***`countername`***.
`-c` ***`changelist`*** **`-t`** ***`countername`***	Set the value of counter ***`countername`*** to ***`changelist`***. This command has been replaced by **`p4 counter`**, but has been maintained for backwards compatibility.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`review`

- The commands `p4 review`, `p4 reviews`, and `p4 counter` are all intended for use by external programs that call Helix server.
- The warnings applicable to `p4 counter` apply here as well.

Related Commands

To list users who have subscribed to review particular files	`p4 reviews`
To set or read the value of a Helix server counter	`p4 counter`
To see full information about a particular changelist	`p4 describe`
To see a list of all changelists, limited by particular criteria	`p4 changes`

p4 reviews

List all the users who have subscribed to review particular files.

"Syntax" on page 19

```
p4 [g-opts] review [-c changelist] [-t countername]
```

Description

The `p4 reviews` command is intended for use in external programs that call Helix server.

Users subscribe to review files by providing file patterns in the `Reviews:` field in their `p4 user` form.

`p4 reviews -c` ***change*** lists each user who has subscribed to review any files included in the submitted changelist ***change***. The alternate form, (`p4 reviews` ***file*** `...`), lists the users who have subscribed to review any files that match the file patterns provided as arguments. If you provide no arguments to `p4 reviews`, all users who have subscribed to review any files are listed.

Options

`-C` ***client***	List all users who have subscribed to review any files opened in the specified workspace ***client***.
`-c` ***change***	List all users who have subscribed to review any files included in submitted changelist ***changelist***.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`

- The syntax `p4 reviews -c` ***changelistfile***`...` ignores the file arguments entirely.
- `p4 reviews` is an unusual command. It was created to support external daemons, but it does nothing without the `Reviews:` field of the `p4 user` form, which has a very specific meaning.

It is possible to enter values in the `Reviews:` field that mean something originally unintended by Helix server to create more generalized daemons. At Perforce, for example, we run a jobs daemon that sends email to any users who have subscribed to review jobs when a new job is submitted. Because there is nothing built into Helix server that allows users to subscribe to review jobs, we co-opt a single line of the `Reviews:` field: Helix server sends a job email to any users who have subscribed to review the non-existent path `//depot/jobs/`.

Related Commands

To subscribe to review files	`p4 user`
List all submitted changelists above a provided changelist number	`p4 review`
To set or read the value of a Helix server counter	`p4 counter`
To read full information about a particular changelist	`p4 describe`

p4 revoke-permission (graph)

Remove from a user or group access to a depot of type `graph` or to a repo.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] revoke-permission -d graphDepot1 -g group [-r ref] -p
permission
p4 [g-opts] revoke-permission -d graphDepot1 -u user [-r ref] -p
permission
p4 [g-opts] revoke-permission -n //graphDepot1/reponame -g group
[-r ref] -p permission
p4 [g-opts] revoke-permission -n //graphDepot1/reponame -u user
[-r ref] -p permission
p4 [g-opts] revoke-permission -n //graphDepot1/reponame -g group
-r ref -p restricted-ref
p4 [g-opts] revoke-permission -n //graphDepot1/reponame -u user -
r ref -p restricted-ref
```

Description

Note
An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

After the administrator has granted a permission to a user or group for a depot of type `graph` or a repo, the administrator can remove that permission with the `revoke-permission` command. The administrator specifies the group or user that will no longer have the specified permission to the specified depot of type `graph` or repo.

An administrator is the `owner`, or a user that has been granted the `admin` permission for that specific graph depot or repo.

Note
Certain permissions imply (implicitly include) other permissions. An implied permission cannot be revoked directly. See the section for "Permissions" on page 228 in the `p4 grant-permission` topic.

Options

`-d`	Applies at the level of the depot, and therefore includes all of its repos.
`-n`	Applies to the repo with the specified name.
`-g`	Applies to the specified group.
`-u`	Applies to the specified user.
`-r`	Applies to the specified branch or tag. Required for the `restricted-ref` permission, but otherwise optional.
`-p`	Applies to the specified permission.
g-opts	See "Global options" on page 690.

Examples

To remove from user bruno the ability to `read` the files in the specified depot of type graph, which also prevents that user from making a Git clone:

```
$ p4 revoke-permission -n //graphDepot1/repo8 -u bruno -p read
```

To remove from user `bruno` the ability to create a Git reference to a release tag in the specified depot of type `graph`.

```
$ p4 revoke-permission -d graphDepot1 -u bruno -r refs/head/rel-*
-p create-ref
```

p4 revoke-permission can use the `-r` option with additional permissions: `delete-ref`, `write-ref`, `force-push`, `write-all`, and `restrict-ref`. To remove from the group `devops` the ability to update the master branch:

```
$ p4 revoke-permission -n //repo/test -g devops -p restricted-ref
-r refs/heads/master
```

"Syntax" on page 19

Related Commands

To assign a permission	`p4 grant-permission`
To list the permissions currently granted	`p4 show-permission`

p4 server

Create, modify, or delete a Helix server specification.

"Syntax" on page 19

```
p4 [g-opts] server serverID
p4 [g-opts] server -g
p4 [g-opts] server -d serverID
p4 [g-opts] server -o [-l] serverID
p4 [g-opts] server -i [-c edge-server|commit-server]
p4 [g-opts] server -c edge-server|commit-server serverID
```

Description

A server specification describes the high-level configuration and intended usage of a Helix server. For installations with only one Helix server, the server specification is optional.

The **`p4 server`** command puts the server spec into a temporary file and invokes the editor configured by the **`P4EDITOR`** variable. Saving the file creates or saves changes to the server specification.

An *operator* type user cannot execute this command. (The three user types are explained in the description of p4 user.)

Filtering

The **`ClientDataFilter:`**, **`RevisionDataFilter:`**, and **`ArchiveDataFilter:`** fields are for replicated environments where you filter out unnecessary data. For instance, a build server does not need to replicate the have list for every open client workspace on the master server. See "Filtering metadata during replication" in *Helix Core Server Administrator Guide*.

Warning
It is best if **`ArchiveDataFilter:`** is kept static. You must reseed the server if you change this filter.

Tip
(2019.1 and later) For a convenient way to adjust the configuration, see the "DistributedConfig:" on page 502 field.

"Syntax" on page 19

Form Fields

Field Name	Type	Description
`ServerID:`	Read-only	A unique identifier for this server. This must match the contents of the server's `server.id` file as defined by the `p4 serverid` command. **Important** To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting `serverID`, but support `P4NAME` for backward compatibility.
`Type:`	Writable	Server executable type. One of the following: Type / Notes: `server` `proxy` – provides a `p4p` caching proxy `broker` – provides a `p4broker` proces `connector` – provides a `git-connector` - p4gconn caching proxy Each type can offer one or more services.

Type	Notes
`server`	
`proxy`	provides a `p4p` caching proxy
`broker`	provides a `p4broker` proces
`connector`	provides a `git-connector` - p4gconn caching proxy

Field Name	Type	Description
`Services:`	Writabl e	The `server` type server provides the following services: ▪ `standard` - a standard Helix server ▪ `replica` - a read-only replica server ▪ `commit-server` - central server in a multi-server installation ▪ `edge-server` - node in multi-server installation ▪ `forwarding-replica` - a replica configured to forward commands that involve database writes to a master server ▪ `build-server` - a replica that supports build automation and builder server (or build farm) integration ▪ `P4AUTH` - a server that provides authentication ▪ `P4CHANGE` - a server that provides change numbering ▪ `standby` - read-only replica server that uses "p4 journalcopy" on page 294 ▪ `forwarding-standby` - forwarding replica server that uses "p4 journalcopy" on page 294 ▪ `local` - personal DVCS server created by p4 init For more information on these services, see "Deployment architecture" in the *Helix Core Server Administrator Guide*. To assign a service to a server, the administrator uses the `Services:` field that appears with the p4 server command:

Field Name	Type	Description

Service	Description	Notes
standard	standard Perforce server	The server that supports directly-connected clients in a single-server environment. (This is the default and does not require explicit assignment in p4 server form.)

"Syntax" on page 19

Field Name	Type	Description

Field Name	Type	Description
`Options:`	Writable	<ul><li>`mandatory`: A standby or forwarding-standby server that persists journalcopy'ed metadata before that metadata is replicated to other replicas. A standby or forwarding-standby server with this option set can be used for failover whether or not the server from which it is journalcopy'ing is available at the time of the failover.</li><li>`nomandatory`: (default) Replication to other replicas can occur before the metadata has been persisted by this standby or forwarding-standby server. Failover can occur to this standby or forwarding-standby server only if the server from which it is journalcopy'ing is available at the time of the failover.</li></ul>
`ReplicatingFrom:`	Writable	Server ID of the server from which this server is replicating or journalcopy'ing. This field is required when the server is a `standby` or `forwarding-standby` server and the `mandatory` option is set for either.
`Name:`	Writable	The `P4NAME` associated with this server. You can leave this blank or you can set it to the same value as the `serverid`.
`Address:`	Writable	The `P4PORT` used by this server. Commit and edge servers require this field if you want to use "p4 reload" on page 436 in this way: `p4 reload -c client-name -p serverID`

Field Name	Type	Description
`ExternalAddress:`	Writabl e	■ This field contains the external address the commit server requires for connection to the edge server. Set to the "P4PORT" on page 677 used when creating the service user login ticket from commit to master server during the commit/edge setup. If the edge server uses ssl, the port must include the ssl prefix. ■ This field is required for parallel submit in a multi-server environment. For example: `tokyo-edge.your-company.com:1666` ■ Prior to 2019.2, for a Git Connector server, this optional field could contain a list of repos to be updated, with a space before each repo name. Beginning in 2019.2, `UpdateCachedRepos` is the field to use for this purpose.
`UpdateCachedRepos:`	Writabl e	Beginning in 2019.2, this optional field can contain a list of repos to be updated, with each repo name on a separate line. See the Upgrading Git Connector and Configuring Git Connector to Poll Repos from Helix4Git topics in the Helix4Git Administration Guide.
`Description:`	Writabl e	An optional description for this server.
`User:`	Writabl e	The service user name used by the server. For additional information about the use of this field, see the section on "Service users" in *Helix Core Server Administrator Guide*.
`AllowedAddresses:`	Writabl e	A list of addresses that are valid this server. At security level 6, this field is used to associate intermediary servers with specified service users. Connections through intermediary servers without matching server specs will be blocked.

Field Name	Type	Description

`ClientDataFilter:`	Writabl e	For a replica server, this optional field can contain one or more patterns describing how active client workspace metadata is to be filtered. Active client workspace data includes have lists, working records, and pending resolves.
		To include client data, use the syntax:
		`//client-pattern/...`
		To exclude client data, use the syntax:
		`-//client-pattern/...`
		All patterns are specified in client syntax.

`RevisionDataFilte r:`	Writabl e	For a replica server, this optional field can contain one or more patterns describing how submitted revision metadata is to be filtered. Submitted revision data includes revision records, integration records, label contents, and the files listed in submitted changelists.
		To include depot data, use the syntax:
		`//depot/pattern/...`
		To exclude depot data, use the syntax:
		`-//depot/pattern/...`
		All patterns are specified in depot syntax.

Field Name	Type	Description

`ArchiveDataFilter:`	Writable	For a replica server, this optional field can contain one or more patterns describing the policy for automatically scheduling the replication of file content. If this field is present, only those files described by the pattern are automatically transferred to the replica; other files are not transferred until they are referenced by a replica command that needs the file content.
		
		Files specified in the `ArchiveDataFilter:` field are transferred to the replica regardless of whether any users of the replica have made requests for their content.
		
		To automatically transfer files on submit, use the syntax:
		
		`//depot/pattern/...`
		
		To exclude files from automatic transfer, use the syntax:
		
		`-//depot/pattern/...`
		
		All patterns are specified in depot syntax.
		
		Warning
		It is best if this filter is kept static. You must reseed the server if you change this filter.

"Syntax" on page 19

Field Name	Type	Description
`DistributedConfig:`	Writable	For all server types, this field shows a line for each configurable that is set to a non-default value. In this field, the admin can edit certain values, add a new line to set certain configurables to a non-default value, or delete a line to reset certain configurables to their default value. Note that: ▪ the commit server's `any#` values, for example, `any#monitor=30`, apply throughout the multi-server environment, unless overridden locally. ▪ this server's local values, for example, `monitor=40`, override the `any#` values. When **p4 server** is invoked with the `-c` flag, the configuration values are populated with currently configured values, recommended default values if unset, or `unset` for unset values with no default. If this field is present when invoked with `-c`, the configuration commands in this field are run on the current server using the scope of the server specified in the `serverID` field. For an edge or commit server, this optional field, which is displayed only when you use the `-c` flag shows current configuration values, recommended default values for fields that are not set, or `unset` for fields that are not set and do not have default values.

Tip
If there is a line under a field, indent that line. For example,

```
Description:
    Created by maria
```

Options

`-c` ***`edge-server`*** `|` ***`commit-server`***	Allow the user to set, change or display configuration values used to set up the multi-server environment on an edge or commit server by using the `DistributedConfig:` field. Configuration fields are initially populated with: ■ the configured values if set ■ default values if unset, or ■ `unset` for unset values with no default After exiting from the form, any configuration commands in the `DistributedConfig:` field will be run on the current server for the scope of the ***`serverID`***. The commands only apply to the ***`serverID`*** server, and so the `server#` prefix is not allowed in these commands. The only supported services are edge-server and commit-server. The service dictates which configuration values are used to initially populate the form the first time that the server command is run.
`-d` ***`serverID`***	Delete the named server specification.
`-g`	Generate a new ***`serverID`*** as part of the form.
`-i`	Read a server specification from standard input. You can combine this option with the `-c` option to generate and run configuration variables used to set up an edge or commit server. When used with `-c`, only the fields explicitly set in standard input from the `"DistributedConfig:" on the previous page` field will be configured.
`-l`	Deprecated because of the functionality of the "DistributedConfig:" on the previous page field.
`-o`	Write the named server specification to standard output.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	see discussion below

"Syntax" on page 19

Only super can run `p4 server` in update mode (using `-i`, `-g`, and `-d` options). Non-operators can run `p4 server` in non-update mode (using `-o` or `-o -g` options). Operators cannot run `p4 server` at all.

Related Commands

To change a server's ID after creation	`p4 serverid`
To list all known servers	`p4 servers`

p4 serverid

Get or set the unique ID associated with a Helix server.

"Syntax" on page 19

```
p4 [g-opts] serverid [serverID]
```

Description

Important
We recommend that you use **p4 serverid** instead of `P4NAME`.

Unless a `P4NAME` value has been specified for the server, the server uses the serverid to determine the appropriate configuration settings. See `p4 configure`.

Important
To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting `serverID`, but support `P4NAME` for backward compatibility.

`p4 serverid` retrieves or sets the unique ID of a Helix server by reading or writing the `server.id` file in the server's root directory.

When you configure servers in a multi-server installation, assign to each server its own serverid and specify the server configuration for that serverid.

Use this command to create or update the `server.id` file after first generating a unique ID for the server with the `p4 server` command.

Important
The `server.id` file is in the server's root directory, and this file must be backed up. If you are using the `p4 server` command to configure your servers, and one of your servers suffers a catastrophic data loss, any attempt to restart the restored server requires that the `server.id` file be present (or be re-created).

To reset the serverid

1. Stop the server.
   For example:

```
p4 admin stop
```

2. Remove the existing `server.id` file.
   For example:

```
cd your/server/root-directory
rm server.id
```

3. Start the server.
4. Run the `p4 serverid` command to assign an ID to your server.
   p4 serverid *yourNewNameForTheServer*
5. Adjust any configurables associated with the `serverid`.
   For example:

```
p4 configure show allservers
p4 configure set servername#monitor=2
```

6. Adjust the fields of the server specification that depend on the `serverid` value, such as `ReplicatingFrom:`

Options

`serverID`	If supplied, update `server.id` with the unique ID of the server.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`, or `super` to set the server ID

Related Commands

To edit or view a server specification	`p4 server`
To list all known servers	`p4 servers`

p4 servers

Display list of all server specifications or evaluate replication status.

A user with operator privileges may execute **`p4 servers`** and **`p4 servers -J`**.

"Syntax" on page 19

```
p4 [g-opts] servers [-J | --replication-status]
```

Description

Syntax variants are described in the following subsections.

Listing server specifications

`p4 servers` lists all server specifications stored at a master Helix server.

```
edge-server_1 server edge-server_1 10.0.101.55:41261 edge-server 'edge-
server '
edge-server_2 server edge-server_2 10.0.101.55:47050 edge-server 'edge-
server '
```

Output lists the server ID, the type, the services provided, and the description supplied when the server was created.

The output of **`p4 servers`** is easier to parse if you retrieve it in tagged form:

`p4 -ztag servers`

```
... ServerID commit-1
... Name
... Address
... Type server
... Services commit-server
... Options nomandatory
... Description commit-1
... ServerID edge-server-1
... Name
... Address
... Type server
... Services edge-server
... Options nomandatory
... Description edge-server-1
... ServerID edge-server-2
... Name
... Address
... Type server
... Services edge-server
... Options nomandatory
... Description edge-server-2

... ServerID remote-standby
... Name
... Address
... Type server
... Services standby
... Options nomandatory
... ReplicatingFrom commit-1

... Description Remote DR server
... ServerID standby-1
... Name
... Address
... Type server
... Services standby
... Options mandatory
... ReplicatingFrom commit-1
... Description Local HA server
```

Note
The `Options` and `ReplicationFrom` fields that appear in tagged output are related to the failover feature. See "p4 failover" on page 190.

Evaluating replication status

Using the `-J` or `--replication-status` option allows you to check how efficiently one or more replicas are replicating the master server's records. Given a server `A` and a replica `B`, output for this command gives you two basic pieces of information:

- The size and update time of `A`'s journal.
- For every server, `B`, that has sent a `p4 pull` or `p4 journalcopy` request, information is given as to when that request was sent and what is the persisted and applied state of `B`'s journal. (In the case of a simple master and replica, the persisted and applied numbers are always the same: `B`'s journal is updated by the `p4 pull` command.

This assumes that the command is executed with the master server as the target. A standby server can replicate master server records using two operations:

- It uses the **`p4 journalcopy`** command to copy (*persist*) the master server's journal to the standby's journal.
- It uses the **`p4 pull -L`** command to *apply* the copied journal records to the standby's database and to update its state file.

You can look at the output to evaluate the load on various parts of your multi-server system and to see how well your replicas are keeping up with the master. Growing lag times might be a reason for concern.

The untagged output of **`p4 servers -J`** looks like this:

```
edge-server_1 '2014/09/18 13:14:58' edge-server 5/258 5/258 WaDl/10 1
edge-server_2'2014/09/18 13:14:57' edge-server 5/258 5/258 WaDl/10 1
```

It is easier to interpret this output in tagged form:

```
... ServerID edge-server_1
... Updated 2014/09/18 13:14:58
... ServerType edge-server
... PersistedJournal 5
... PersistedSequence 258
... AppliedJournal 5
... AppliedSequence 258
... JAFlags WaDl/10 1
... IsAlive 1

... ServerID edge-server_2
... Updated 2014/09/18 13:14:57
... ServerType edge-server
... PersistedJournal 5
... PersistedSequence 258
... AppliedJournal 5
... AppliedSequence 258
... JAFlags WaDl/10 1
... IsAlive 1
```

The meaning of the fields are described in the following table.

`ServerID`	The server ID of the server. **Important** To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting **`serverID`**, but support **`P4NAME`** for backward compatibility.
`Updated`	The date and time the requesting server last requested journal records from this server (normally the master).
`ServerType`	The server type. One of the following: **`standard`**, **`replica`**, **`forwarding-replica`**, **`build-server`**, **`edge-server`**, **`commit-server`**, **`depot-master`**, **`depot-standby`**, **`standby`**, **`forwarding-standby`**.
`PersistedJournal`	The rotation number of the journal to which records are being persisted.
`PersistedSequence`	The persisted journal position. For master servers, replicas, and workspace servers, the persisted and applied positions are always the same. They differ only for all types of standby servers.
`AppliedJournal`	The rotation number of the applied journal.
`AppliedSequence`	The applied journal position. For master servers, replicas, and workspace servers, the persisted and applied positions are always the same.

`JAFlags`	Set of fields printed in upper-case if set or lower-case if not. The numeric value of the flags is displayed after the alphabetic display. Common field displays with their associated pull or journalcopy commands are as follows: ■ `WAdl/12`: `p4 journalcopy -i 0` ■ `WaDl/10`: `p4 pull -i 0` ■ `wAdl/4`: `p4 journalcopy -i 1` ■ `waDl/2`: `p4 pull -i 1` ■ `wadL/1`: synthesized record for master status. You can compare the journal positions of each replica with that of this server to see if any replica is falling behind. Symbols are to be interpreted as follows: ■ `W/8`: wait, long-poll request ■ `w`: no wait ■ `A/4`: Acknowledging ■ `a`: non-acknowledging ■ `D/2`: durable ■ `d`: non-durable ■ `L/1`: data about the local journal; that is, the journal of the server that is the target of the `p4 servers` command. ■ `l`: request from a replica (shows progress in copying master's journal).
`IsAlive`	1 if the server is up; 0 if it's down.

Pull or journal-copy requests are recorded in the `db.jnlack` table only when made from a replica that has either a server ID or a `P4NAME`. Any replica that makes such a request but does not have a server ID or `P4NAME` is not recorded in the table.

Options

`-J|--replication-status`	Provides information about the server's journal and about the replication status of all replicas that replicate from this server.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Related Commands

To edit or view a server specification	`p4 server`
To set a server's unique ID	`p4 serverid`

p4 set

Set Helix server system variables.

"Syntax" on page 19

```
p4 [g-opts] set [-q] [-s] [-S svcname] [var=[value]]
```

Description

Both Helix server client applications and the shared versioning service make use of certain system variables. Depending on the operating system and other factors, variable definitions may be stored in the file defined by `"P4CONFIG" on page 650`, in the file defined by `"P4ENVIRO" on page 657`, or in the Windows registry.

- On Linux, values defined with **`p4 set`** are stored in the **`P4ENVIRO`** file.
- On Windows, values defined with **`p4 set`** are stored in the **`P4ENVIRO`** file if this is set. If it is not set, they are stored in the Windows registry.

  Windows administrators running Helix server as a service can set environment variables used by the service with **`p4 set -S`** ***`svcname var=value`***. These variables are always stored in the Windows registry.

  To change a variable setting that applies to the current user, use **`p4 set`** ***`var=value`***. Administrators can use **`p4 set -s`** ***`var=value`*** to set the variable's default values for all users on the machine.

Precedence

Tip
You can specify client settings such as port, user, and workspace names by using any of the following:

1. On the command line, using options.
2. In the configuration file(s) specified by a P4CONFIG environment variable, where each config file can be specific to a workspace.
3. In the P4ENVIRO configuration file, which is for variables that remain constant for all the workspaces on a given computer.
4. User environment variables.

5. System environment variables (on Windows, system-wide environment variables are not necessarily the same thing as user environment variables)
6. In the user registry or settings set by issuing the `p4 set` command.
7. In the system registry or system settings set by issuing the `p4 set -s` command.

where

- Command line overrides P4CONFIG
- P4CONFIG overrides P4ENVIRO, and so on.

The output of `p4 set` lists the values of the variables (and if a given variable was set by `config`, `enviro`, `set`, or `set -s`).

Unsetting and viewing variable values

To unset the value for a particular variable, leave ***value*** empty.

To view a list of the values of all Helix server variables, use `p4 set` without any arguments. If a `P4CONFIG` file was used to set the variable, its location is displayed.

- On UNIX, this displays the values of the associated environment variables.
- On Windows, this displays either the environment variable (if set), or the value in the registry and whether it was defined with `p4 set` (for the current user) or `p4 set -s` (for the local machine).

Note how the source of the variable definition is shown in this sample output to the `p4 set` command. If no source is given, the value is stored in the variable itself rather than in a file containing the definition.

```
P4CLIENT=symlinks-nix (config)
P4CONFIG=p4config.txt (config '/home/perforce/p4clients/symlinks-
nix/p4config.txt')
P4EDITOR=/usr/bin/vi
P4IGNORE=p4ignore.txt (enviro)
P4PORT=win-bruno:20151 (config)
P4USER=bruno (set -s)
P4_20151_CHARSET=none (set)
```

Options

`-q`	Reduce the output. When listing files, don't display the origin of the setting. The output is suitable for parsing with scripts.
`-s`	Set the value of the registry variable for the local machine. On Windows, without this option, `p4 set` sets the variables in the `HKEY_CURRENT_USER` hive. When you use the `-s` option (and have Windows administrative privileges), the variables are set in the `HKEY_LOCAL_MACHINE` hive. The location is reflected in the output of `p4 set` on Windows.
`-S` *`svcname`*	Set the value of the registry variables as used by service *`svcname`*. You must have Windows administrator privileges to do this.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`none`

- You'll find a listing and discussion of the Helix server variables in the "Environment and registry variables" on page 637 section of this manual.
- Changes to registry settings under Windows affect the local machine only. An administrator setting `P4JOURNAL` for a Helix server Windows service must be present at the machine running the service. See also the Knowledge Base article, "Windows Environment Variable Precedence".
- If you're working in a UNIX-like environment on a Windows machine (for example, Cygwin), use environment variables instead of `p4 set`. (In these cases, the Helix server Command-Line Client behaves just as though it were in a UNIX environment.)

Examples

`p4 set`	On all platforms, display a list of Helix server variables and their origins without changing their values.
`p4 set -q`	On all platforms, display a list of Helix server variables in a format suitable for scripts to parse without changing their values.

`p4 set P4MERGE=`	On Windows or OS X, unset the value of `P4MERGE`.
`p4 set P4PORT=ssl:tea:1666`	On Windows, set a variable telling Helix server applications to connect to a Perforce service at host `tea`, port `1666`, via SSL. The variable is set only for the current local user.
`p4 set -s P4PORT=ssl:tea:1666`	Set `P4PORT` as above, but for all users on the system. You must have administrative privileges to do this.
`p4 set -S p4svc P4PORT=1666`	For the Windows service `p4svc`, instruct `p4s.exe` to listen on port `1666` for incoming connections from Helix server applications. You must have administrative privileges to do this.
`p4 set -S Perforce P4DEBUG="` `"net.keepalive.idle" on page 774` `=2700"`	For the Windows service, which defaults to the name `Perforce`, instruct `p4s.exe` to wait `2700` seconds before starting to send keepalives. You must have administrative privileges to do this.
`p4 set P4EDITOR="C:\File Editor\editor.exe"`	On Windows, for the current local user, set the path for the default text editor. The presence of spaces in the path to the editor's executable requires that the path be enclosed in quotation marks.

p4 shelve

Store files from a pending changelist in the depot, without submitting them.

"Syntax" on page 19

```
p4 [g-opts] shelve [-Af | -As] [-p] [files]
p4 [g-opts] shelve [-Af | -As] [-a option] [-p] -i [-f | -r]
p4 [g-opts] shelve [-Af | -As] [-a option] [-p] -r -c changelist#
p4 [g-opts] shelve [-Af | -As] [-a option] [-p] -c changelist# [-
f] [files]
p4 [g-opts] shelve [-Af | -As] -d -c changelist# [-f] [files] --
parallel=threads=N[,batch=N][,min=N]
```

Description

Shelving is the process of temporarily storing work in progress in Helix server without submitting a changelist. Shelving is useful when you need to:

- perform multiple development tasks on the same set of files, such as testing across multiple platforms

or

- share files for code review before committing your work to the depot

The **`p4 shelve`** command creates, modifies, or discards shelved files in a pending changelist. Shelved files persist in the depot until they are discarded (by means of **`p4 shelve -d`**) or replaced by subsequent **`p4 shelve`** commands. **`p4 shelve`** displays the working revision for the files being shelved.

In addition to the files being shelved, **`p4 shelve`** also shelves any open stream specification. For open stream specifications, see **`p4 stream`**.

After shelving files, you can:

- revert or modify them in your client workspace
- restore the shelved versions of those files to your workspace with the **`p4 unshelve`** command.

While files are shelved, other users can unshelve the shelved files into their own workspaces, or into other client workspaces.

Files that have been shelved can also be accessed with the **`p4 diff`**, **`p4 diff2`**, **`p4 files`**, and **`p4 print`** commands, using the revision specifier `@=`***`change`***, where ***`change`*** is the pending changelist number.

If you are working in a multi-server environment, use the `-p` option to promote a shelved change from an edge server to a commit server where it can be accessed by other edge servers in the multi-server configuration. When an existing shelved change is promoted, it is promoted without modification unless the `-f` or `-r` options are also used to change the shelved file content. See "Usage Notes" on page 520 and "Promoting shelved changelists" in *Helix Core Server Administrator Guide*.

If no arguments are specified, `p4 shelve` creates a new changelist, adds files from the user's default changelist, and (after the user completes a form similar to that used by `p4 submit`), shelves the specified files into the depot. If a file pattern is given, `p4 shelve` shelves only the files that match the pattern.

To add a file to a pre-existing shelve, the file must first be opened in the shelve's changelist.

To move an opened file from one changelist to another, use the `p4 reopen` command.

Note
`p4 obliterate myfile` does not obliterate a shelve of the file (archive or metadata). When you attempt to unshelve a file that has been obliterated, you will get an error. To recover the content of that file, print the file. To get rid of the shelve, delete the shelf.

Tip
When you create a shelf from files opened in the default changelist, the syntax of `[files]` means a **single** file pattern, such as:

```
p4 shelve ....html
```

(Note that wildcards are allowed to specify multiple files.)

However, when opened files are in a numbered change, **multiple** file arguments are possible:

```
p4 shelve -c 12108 ....html "*.c"
```

streams and p4 shelve

By default, if the stream spec is open, it will also be included with any shelved changelist. See the command line output of `p4 help streamcmds`

By default, a stream spec is not deleted until all files have been deleted.

Options

`-Af`	Specifies that only files be shelved with this changelist.
`-As`	Specifies that only an opened stream specification should be shelved with this changelist. By default, if the stream spec is open and neither `-Af` nor `-As` is given, the stream specification will also be included with any shelved files. (See `p4 help streamcmds`)

`-a` *`option`*	The **`submitunchanged`** (default) option shelves all files. The **`leaveunchanged`** option shelves only the changed files. It leaves the unchanged files opened at the numbered pending changelist.
`-c` *`change`*	Specify the pending changelist in which shelved files are to be created, discarded, or modified. Only the user and client that owns the pending changelist can add or modify its shelved files. (Administrators can use `-f` to discard files.) Any files specified by a file pattern must already be open in the specified changelist. To move an opened file from one changelist to another, use `p4 reopen`.
`-d`	Using `-d -c` flag deletes the shelved files in the specified changelist so that they are no longer available for `p4 unshelve` operations. By default, only the user and client of the pending changelist can delete its shelved files. A user with **`admin`** access can delete shelved files by including the `-f` flag to force the operation. If the shelved changelist includes a stream spec, by default it is deleted when all files have been deleted. The combinate of `-d -As` forces the the stream spec to be deleted even if files remain.
`-f`	To force an overwrite any existing shelved files in a pending changelist, use the `-f` option with the `-c` or `-i` option. Force the overwriting of any existing shelved files in a pending changelist with the contents of their client workspace copies. Helix server administrators can use this option with `-d` to force the discarding of shelved files in a specified changelist. Using this option will delete shelved files that are the source of pending resolves. If this happens, the resolving user will not be able to merge content from the shelf. The user must either ignore (`-ay`) the missing shelf or revert.
`-i`	Reads the pending changelist specification with shelved files from the standard input. The user's editor is not invoked. To modify an existing changelist with shelved files, specify the changelist number using the `-c` option.
`-p`	**Promote** a shelved change from an edge server to a commit server where it can be accessed by other edge servers participating in the multi-server configuration. Once a shelved change has been promoted, all subsequent local modifications to the shelf are also pushed to the commit server and remain until the shelf is deleted. See "Usage Notes" on the next page for more information. The combination of `-p -c` promotes the shelf without modification unless `-f` or `-r` are also used to update the shelved files before promotion.

`-r`	Replace all shelved files in the changelist with the files that are opened in your workspace. The `-r` option (used with `-c` or `-i`) enables you to replace all shelved files in that changelist with the files opened in your own workspace at that changelist number. Previously shelved files will be deleted. Only the user and client workspace of the pending changelist can replace its shelved files.
`--parallel`	Specifies that multiple files should be transferred in parallel, using independent network connections from automatically-invoked child processes. See also the configurables "net.parallel.shelve.batch" on page 782, "net.parallel.shelve.min" on page 782, and "net.parallel.shelve.threads" on page 783.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`write`

A **promoted shelf** is a shelf that exists on the commit server of a multi-server configuration. It is there either because it was directly created on the Commit server or because it was promoted with the `-p` option of the `p4 shelve` command. Commands that access shelves know how to handle promoted shelves.

To unpromote a shelf, delete the shelf and create a new one.

A shelf can be promoted when it's first created. A normal shelf can be promoted after it is created by running one of the following commands:

```
$ p4 shelve -p -f -c myChange
$ p4 shelve -p -r -c myChange
```

Promoting a shelf gives you a way to move a shelf from one server to another. To do this, you must complete the following steps:

1. Promote the shelf you want to copy on the server from where you want to copy it, say server X.
2. Unshelve the shelf in the server to which you want to copy it, say server Y.
3. Shelve the change on server Y; this opens the files in a change that is owned by server Y. The new shelf is created as a non-promoted shelf; but you can promote it if you like.

To determine whether a shelved change is promoted, you can try to access the shelf on a server other than the server that owns the change, or you can look at the output of the `p4 -ztag changes` command.

Observe the following limitations when working with promoted shelves:

- You can’t unload an Edge server workspace if you have promoted shelves.
- Use promoted shelves sparingly. Shelf promotion and shelf access are time-consuming operations.

Related Commands

To restore shelved files into a workspace	`p4 unshelve`

p4 show-permission (graph)

Display the permissions for the specified depot of type `graph` or a repo.

> **Note**
> For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] show-permission -d graphDepot1 [-g group | -u user] [-
r ref] [-p permission]
```

```
p4 [g-opts] show-permission -n //graphDepot1/reponame [-g group |
-u user] [-r ref] [-p permission]
```

Description

> **Note**
> An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

The administrator uses this command to list the permission assignments for the specified depot of type `graph` or repo.

> **Note**
> Certain permissions implicitly include other permissions. If you specify a permission that can be implicit or explicit, such as `write-ref`, the list shows users that have the specified permission, either explicitly or implicitly, such as users with `write-all` and `admin`. See the "permissions" section of p4 grant permission.

Options

`-d`	Applies to the depot of type `graph` with the specified name.
`-n`	Applies to the repo with the specified name.
`-g`	Applies to the specified group.
`-u`	Applies to the specified user.

`-p`	Applies to the specified permission.
`-r`	(Optional) Applies to the specified branch or tag.
`g-opts`	See "Global options" on page 690.

Examples

To list the permissions to a depot of type `graph`, which includes all of its repos, use the `-d` option:

```
p4 show-permission -d graphDepot1
```

To list the permissions to the specified repo, use the `n` option with the //*depot*/*repo* syntax:

```
p4 show-permission -n //graphDepot1/repo1
```

To list the permissions of the specific user to the specified repo, use the `-u` and `-n` options:

```
p4 show-permission -u bruno -n //graphDepot1/repo1
```

To list all the users with the specified permission to the specified repo, use the `-p` option:

```
$ p4 show-permission -p read  -n //graphDepot1/repo1
```

Related Commands

To assign a permission	"p4 grant-permission (graph)" on page 227
To remove a permission	"p4 revoke-permission (graph)" on page 490
To see a user-centric view across multiple repos or depots of type graph	"p4 show-permissions (graph)" below
Check access permission(s) granted to a user of a repo	"p4 check-permission (graph)" on page 95

p4 show-permissions (graph)

Display a user-centric view of the permissions for repos.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 [g-opts] show-permissions -d graphDepot1 [-g group | -u user]
```

```
p4 [g-opts] show-permissions -n //graphDepot1/reponame [-g group |
-u user]
```

Description

Unlike "p4 show-permission (graph)" on page 522, which requires specifying a repo (or graph depot), this command can report across multiple repos (or graph depots). For some queries, this is convenient.

> **Note**
> An administrator is the owner, or a user that has been granted the `admin` permission for that specific graph depot or repo.

User type	Options
Users	Running with no arguments shows the current user the permissions that user has across all the repos for which that user has permissions. Adding `-d` or `-n` restricts the view to the specified depot or repo.
Administrators	Can also use the `-u` option, but only in conjunction with the `-d` or `-n` option, to view the permissions of another user of this admin's depot or repo.
Super users	Can also use the `-u` flag to specify another user.

> **Note**
> Certain permissions implicitly include other permissions. If you specify a permission that can be implicit or explicit, such as `write-ref`, the list shows users that have the specified permission, either explicitly or implicitly, such as users with `write-all` and `admin`. See the "permissions" section of p4 grant permission.

Options

`-d`	Applies to the depot of type `graph` with the specified name.
`-n`	Applies to the repo with the specified name.
`-u`	Applies to the specified user.
`-g`	Applies to the specified group.

`-r`	(Optional) Applies to the specified branch or tag.
`g-opts`	See "Global options" on page 690.

Related Commands

To show permissions for a specific repo	"p4 show-permission (graph)" on page 522
Check access permission(s) granted to a user of a repo	"p4 check-permission (graph)" on page 95
To get user-centric "classic" permissions	"p4 protects" on page 409

p4 show-ref (graph)

Display reference values.

Note
For depots of type `graph` only.

"Syntax" on page 19

```
p4 graph show-ref [-n //repo/name] [-a -u user -m max -t ref-type
] [[-e|-E] nameFilter]
```

Description

The graph show-ref command displays reference values, which is similar to the Git command `git-show-ref`.

Options

`-n`	Applies to the repo with the specified name. If a repo name is specified, you have to have read permission for that repo. Otherwise you must be a super user.
`-a`	Displays all references from all repos.
`-u`	Displays only references readable by the user argument.

`-m`	Specifies the maximum references to display.
`-e nameFilter`	Lists references with a name that matches the *nameFilter* pattern using the server's normal case-sensitivity rules. For example: **`-e 'refs/heads/dev...'`**
`-E nameFilter`	Lists references with a name that matches the *nameFilter* pattern with case-insensitive matching, even on a case-sensitive server. For example: **`-E 'Refs/HEADS/Dev...'`** gets the same results as **`-e 'refs/heads/dev...'`**

p4 sizes

Display size information for files in the depot.

"Syntax" on page 19

```
p4 [g-opts] sizes [-a -S] [-s|-z] [-b blocksize] [-h | -H] [-m
max] FileSpec[revSpec]
p4 [g-opts] sizes -A [-a -s] [-b blocksize] [-m max]
archiveFileSpec
p4 [g-opts] sizes -U unloadFileSpec
```

Description

The `p4 sizes` command displays the sizes of files stored in the depot. When called with no options, only the size of the head revision of the file or files is displayed. One line of output is provided per file.

Use the `-a` option to see how much space is occupied by each individual revision in the specified revision range, rather than just the highest revision in the specified range. One line of output is provided per file, per revision.

Use the `-s` option to obtain the sum of all files specified. Only one line of output is provided, showing the file specification, the number of files summarized, the total number of bytes required, and (if the `-b` option is provided) the total number of blocks required.

The `-h` or `-H` option displays size in human-readable form, using a scaling factor of 1,024 for `-h` or 1,000 for `-H`. The size displayed will be automatically scaled to bytes, kilobytes, megabytes, gigabytes, or terabytes, as needed. For example, if you specify `-h`, the output of 75,883,921 bytes, would be represented as 72.36 M.

The `-z` option works the same way as `-s`, but excludes space occupied by lazy copies (files that exist by virtue of integration operations). Use `-z` to estimate the space occupied by files on a Helix server installation, and use `-s` to estimate the local diskspace requirement if files were synced to a client workspace.)

Options

`-a`	Include all revisions within the range, rather than just the highest revision in the range.
`-A`	Display files in archive depots. See `p4 archive` for details.

`-b` ***`blocksize`***	Display results in blocks of ***`blocksize`*** bytes. Each accumulated file size is rounded up to the nearest ***`blocksize`*** bytes.
`-m` ***`max`***	Limit output to ***`max`*** lines of output.
`-h` or `-H`	Display size in human-readable form, using a scaling factor of 1,024 for `-h` or 1,000 for `-H`. The size displayed will be automatically scaled to bytes, kilobytes, megabytes, gigabytes, or terabytes, as needed.
`-s`	Calculate the sum of the file sizes for the specified file argument.
`-S`	Display size information for shelved files only. If you use this option, revision specifications are not permitted.
`-U` *`unloadfile`*	List only file sizes in the unload depot. See `p4 unload` for details.
`-z`	When calculating size information, exclude lazy copies.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- The **`p4 sizes`** command is functionally similar to the UNIX **`du`** command.
- If no revision range is specified, the implicit revision range of `#1` through `#head` is assumed.
- File sizes are based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.

Examples

`p4 sizes file.c`	Show the size of the head revision of `file.c` in the depot.
`p4 sizes -a file.c`	Show the sizes of each revision of `file.c` stored in the depot.
`p4 sizes -s -a file.c`	Show the total size of all revisions of `file.c` stored in the depot.

"Syntax" on page 19

`p4 sizes -s -a -b 512 //depot/...`	Show the number of files and the total diskspace (in bytes and 512-byte blocks) currently used by a Helix server installation hosting **`//depot/...`**
`p4 sizes -s //workspace/...`	Show the number of files and the total local diskspace (in bytes) required to sync the head revisions of files mapped to the client workspace named **`workspace`**.

p4 status

Previews output of open files for add, delete, and/or edit in order to reconcile a workspace with changes made outside of Helix server.

The **`p4 status`** command produces output in local syntax. To see file names and paths in depot syntax, use the **`-n`** option to **`p4 reconcile`**.

"Syntax" on page 19

```
p4 [g-opts] status [-c change] [-A | [-e -a -d] | [-s]] [-f -I -
m] [file ...]
```

Description

When called without arguments, **`p4 status`** only previews the results of the workspace reconciliation. To limit the scope of **`p4 status`** to add, edit, or delete, use the **`-a`**, **`-e`**, or **`-d`** options. You must use either **`p4 status -A`** (or **`p4 reconcile`**) to actually open the files in a changelist.

The **`p4 status`** command finds unopened files in a client's workspace and detects the following three types of inconsistencies between your workspace and the depot:

1. Files present in the depot, present in your have list, but missing from your workspace. By default, these files are then opened for **`delete`**.
2. Files present in your workspace, but missing on the depot. By default, these files are opened for **`add`**.
3. Files modified in your workspace that are not open for edit. By default, these files are opened for **`edit`**.

If the list of files to be opened includes both adds and deletes, the missing and added files are compared and converted to pairs of **`move/delete`** and **`move/add`** operations (as long as the files' sizes and contents are similar.)

By default, **`p4 status`** displays opened files as well as files that need to be reconciled. If you use the **`-A`**, **`-e`**, **`-a`**, or **`-d`** options or client applications earlier than 2015.1, opened files are not displayed.

By default, **`p4 status`** does not check files and/or paths mentioned in the **`P4IGNORE`** file. Use the **`-I`** option to override this behavior and ignore the **`P4IGNORE`** file.

Reconcile and implicit p4 move affects p4 status

If you do a "p4 reconcile" on page 431 and some of the files show up as potential "adds" while others show up as "deletes", Helix server compares the new files and the missing files to see if any of them appear to be the same file. If so, Helix server links them as if you had used `"p4 move" on page 377` to open them. The output of **`p4 reconcile`** and **`p4 status`** will include a "**`... moved from`**" line so that you can preview how Helix server matched the files before you submit them. You can also use "p4 resolved" on page 472 after opening the files.

Options

`-a`	Display files to be opened for add.
`-A`	Add, edit, *and* delete files. Files in the client workspace not under Helix server control are opened for add. Changed files are opened for edit. Files in the user's have list that have been removed from the workspace are opened for delete. **`p4 status -A`** is equivalent to **`p4 reconcile -ead`**.
`-c` ***`change`***	The changelist containing the files whose status is sought.
`-d`	Display files to be opened for delete.
`-e`	Display files to be opened for edit.
`-f`	Display files to be added whose names contain special (wildcard) characters. Files containing the special characters `@`, `#`, `%`, or `*` are reformatted to encode the characters using hex notation. After these files are added, you must refer to them using their reformatted filenames.
`-I`	Do not perform any ignore checking; ignore any settings specified by **`P4IGNORE`**.
`-m`	Use in conjunction with the **`-e`** option to minimize costly digest computation on the client by checking file modification times before checking digests to determine if files have been modified outside of Helix server.
`-s`	Generate summarized output for files to open for add. Using this option causes the command to preview files needing to be reconciled, but provides shorter output for files to be opened for add. Files in the current working directory are listed, but subdirectories containing files to be opened for add are listed rather than the individual files. This provides the shorter output. **Note** This requires version 2015.1 or later of both server and client.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

Related Commands

To reconcile a workspace that has been modified outside Helix server	`p4 reconcile`

p4 storage

Display, verify, or update physical archive storage.

"Syntax" on page 19

```
p4 [g-opts] storage [-v [-q]] [-c change] [-T tags -F filters] [-m
max] archive...
p4 [g-opts] storage -u [-c change] [-T tags -F filters] [-m max]
archive...
p4 storage -d [-c change] [-y] [-D secs] [ -t target ] [-q] archive...
p4 storage -w
p4 storage -U [-q] archive...
p4 storage -l start|pause|restart|wait|status|cancel //depotdirectory/...
```

Description

A user with **`admin`** access can display information about the server archive files and their usage with

`p4 storage [-v [-q]] [-c change] [-T tags -F filters] [-m max] archive...`

The **`-l`** option provides a group of commands to locate orphaned archive files and mark them for removal by the **`storage -d`** command.

See also "Reclaiming disk space by removing orphaned archive files" under Managing disk space in *Helix Core Server Administrator Guide*.

The fields that storage displays are:

`lbrFile`	the path of the archive file
`lbrRev`	the revision of the archive file
`lbrType`	the file type of the archive file, such as `text+C` See "File types" on page 707.
`lbrRefCount`	how many revs refer to the librarian file
`headTime`	timestamp of when the record was written
`filesize`	size of the librarian file
`digest`	MD5 digest of the librarian file
`serverSize`	Size of the librarian file on the server at the time that revison was submitted or shelved
`compCksum`	MD5 of the librarian file containing the revision on the server if the file is compressed and submit.storagefields is 1

Additional fields when the `-v` flag is specified:

`actualSize`	actual size of librarian file
`actualDigest`	actual MD5 digest of librarian file
`status`	outcome of verify, `OK`, `BAD` or `MISSING`

If upgrading from a verion prior to 2019.1

If you are upgrading your Helix server from a version prior to 2019.1 and your Perforce database has many keyed revisions, you might want to avoid the risk of delay incurred by calculating storage digest updates *during* the upgrade process. If so, consider using the `-U` option.

Options

`-v`	Verifies that the specified archive files are intact. Computes and displays the digest of each archive. ■ If an archive cannot be reproduced (for example, if the file is missing from the archive), the archive's output line ends with `MISSING!` ■ If there is a saved digest, `p4 storage` compares it with the computed one. If they differ, the output line ends with `BAD!` `p4 storage -v` is similar to "p4 verify" on page 627 except that: ■ for `storage -v` the ***`archive`***`...` argument is matched against the `lbrPath` of the storage tables ■ for `p4 verify` the ***`archive`***`...` is matched against the `depotpath` field of the revision tables
`-u`	Computes and saves the digest only for revisions that have no saved digest.
`-q`	Minimizes the output of the `storage -v` command, displaying only errors from mismatched digests or unreproducible revisions.
`-d`	Deletes unreferenced archives. If the specified archive has a reference count of `0`, the database row describing the archive is removed as well as the revision of the underlying archive. Without the `-y` as confirmation, the command merely lists the candidate files and revisions it would remove. With the `-y` as confirmation, files and revisions as well as zero referenced storage records are removed. The expiry time is set by the "lbr.storage.delay" on page 760 configurable. The expiry time is intended to protect the storage records and archive files of in-progress submits and in-progress shelves from being considered for deletion. To override this configurable, specify the `-D secs` parameter on the command line. Archives with a reference count of `0` are rare, but can be created by: ■ incomplete "p4 submit" on page 558 or "p4 shelve" on page 517 commands, system crashes, other similar errors ■ `p4 storage -l start //`***`depotdirectory`***`/...` The listing of the candidate revisions can be suppressed with the `-q` option. The `-t` target option copies the unreferenced archives to the target directory you specify before deleting the unreferenced archives from the server's "P4ROOT" on page 680.
`-c`	Applies to the specified changelist.

`-T`	Applies to the specified tags, which can be specified using a comma- or space-delimited list. Example: `-T "lbrFile, lbrRev"`
`-F`	Applies to the specified filter. Lists only records satisfying the filter expression. This filter syntax is similar to the one used for `jobs -e jobview` and is used to evaluate the contents of the fields in the preceding list. Filtering is case-sensitive. Example: `-v -F "lbrRefCount = 1 & status=OK"`
`-m`	Limits output to the specified number of records
`-w`	Provides a message that indicates when the upgrade process is complete by displaying "`The storage upgrade process is complete`". For details, see *Helix Core Server Administrator Guide* on upgrading to 2019.1 or later.
`-U`	To avoid the delay incurred by calculating the digest *during* the upgrade, consider using the `-U` option *after* a storage upgrade has been performed with the lbr.storage.skipkeyed configurable set to skip keyword revisions. This option reads all the storage records looking for an unset digest field. When it detects an unset digest, it recomputes the digest and size fields, then updates the record. Invoking `p4 storage -U -q` suppresses the progress messages. The use of `-U` or `-U -q` requires `super` access.

`-l`	Allows the administrator to create, pause, restart and cancel background processes that can locate and mark orphan files. This process looks for files in the specified `//`***`depotdirectory`*** and its subdirectories that match the naming convention for the server's archive file format. Each file that is found, along with its revision, is checked for a matching entry in the `db.storage` table. If no such entry is found, a new record is created that describes the file and revision combination with a reference count of `0`. A later call to **`p4 storage -d`** will delete both those records with zero references and the associated the archive revision. **`p4 storage -l`** has subcommands:

`start`	Start a new background depot scanning task in the specified `//`***`depotdirectory`*** to scan it and all its subdirectories. The special `//`***`depotdirectory`*** argument of `//...` is handled by searching for all the local depots, including local stream depots, listed in the depot table and scanning their top and subdirectories. There is no limit on the number of background processes that can be started, except the limits of the hardware and operating system. Each directory in the archive can only be the subject of a single scan. If the server shuts down while a scan is unfinished, the scan is marked paused and the administrator can manually restart the scan again when the server is started.
`pause`	Suspend the named background scan process
`restart`	Resume the named paused background scan process. If the server shuts down while a scan is unfinished, the scan is marked **`paused`** so that when the server is running again, the administrator can manually **`restart`** the scan.
`wait`	Wait until the named background scan process completes or fails due to an error. The **`wait`** subcommand: ▪ returns no output. It only returns when the scanner is no longer active, which means its state is **`done`**, **`error`**, or **`paused`**. ▪ waits on a single scanner task. For example, with `//...` there is a single scanner task that searches all the local depots

`status`	Report on all the background scan processes if the special argument of `//...` is specified, or report on the named background scan process. For each scan, the following fields are displayed:

`scanName`	Name of the scan, the argument passed to the `start` subcommand, which is the path to scan. See the "Example of a possible workflow" below
`state`	`done`, `paused`, `run`, `busy`, or `error`
`dirsProc`	The number of directories that have been processed
`filesProc`	The number of files that have been processed
`zeroRecs`	The number of zero referenced storage records created. This is the number of orphan revisions that have no matching storage record.
`dirsErrs`	The number of directories skipped due to errors
`errmsg`	The error message if the status is `error`

`cancel`	Remove the scan from the system. A scan remains in the system until it is canceled. To re-run a scan, first `pause` the scan (if it is active), then `cancel` the scan (even if it has completed or resulted in an error).

Warning

- Scanning requires that each file only be addressed by a single path under the scanning directory. This requirement might be broken by the presence of symlinks. By default the scanner returns an error if it detects a symlink. To allow symlinks, set the `"lbr.storage.allowsymlink" on page 759` configurable to `1`. This should only be done if all the symlinks only reference files and directories that are not under any scanned directory.
- The scanner does not run if the depot is shared between servers.

g-opts	See "Global options" on page 690.

Example of a possible workflow

Locating the orphaned files

To start the scan:

```
p4 storage -l start //depot/path/...
```

To get the status of the scan:

```
p4 storage -l status //depot/path/...
```

To pause the scan:

```
p4 storage -l pause //depot/path/...
```

To end the pause and resume the scan:

```
p4 storage -l restart //depot/path/...
```

To cancel the scan:

```
p4 storage -l cancel //depot/path/...
```

Note
A scan remains in the system until it is canceled. To re-run a scan, first `pause` the scan (if it is active), then `cancel` the scan (even if it has completed or resulted in an error).

Reclaim disk space on the server

To save the orphaned files to your target directory and delete the zero references and associated the archive files from the server's "P4ROOT" on page 680:

```
p4 storage -d -y -t /myTargetDirectory //depot/path/...
```

where `/myTargetDirectory` is any operating system target directory, and, for Windows, might be something like `D:\myCopyOfOrphanedFiles`

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`admin` or `super` to use the `-U` option

p4 stream

Create or edit an instance of a stream (also known as a stream definition).

"Syntax" on page 19

```
p4 [g-opts] stream [-P parent] -t typename name
p4 [g-opts] stream [-f -d] [-o [-v]] [-P parent] -t typename [name
[@change]]
p4 [g-opts] stream -i [-f] name
p4 [g-opts] stream edit
p4 [g-opts] stream resolve [-a flag] [-n] [-o]
p4 [g-opts] stream revert
```

Description

The `p4 stream` command enables you to maintain Helix server streams, which are hierarchical branches with policies that control the structure and the flow of change. Stream hierarchies are based on the stability of the streams, specified by the **type** you assign to the stream:

mainline	▪ somewhat stable ▪ the parent of all streams in the stream depot
development	▪ least stable (most subject to change) ▪ flow is controlled
release	▪ highly stable ▪ flow is controlled
task	▪ lightweight short-lived stream that only promotes edited files to the repository ▪ branched and integrated files are stored in shadow tables that are removed when the task stream is deleted or unloaded
virtual	▪ used to copy and merge between parent and child streams without storing local data ▪ not a stream but an alternate view of its parent stream

"Syntax" on page 19

Stream contents are defined by the **paths** that you map. By default, a stream has the same structure as its parent (the stream from which it was branched). However, you can override the structure. For example, your override might ensure that specified files cannot be submitted or integrated to other streams.

By default, `p4 stream` edits the stream associated with your current workspace. It throws an error if you're not using a stream workspace. For more information, see the "Streams" chapter of the *Helix Core Server User Guide*.

For details on the recommended workflow to edit, resolve, and revert a stream, see "Update Streams" in *Helix Core Server User Guide*.

> **Note**
> If you integrate from a classic branch or other stream depot to a task stream, the files are not copied up to the parent unless they are edited and submitted first.

For a detailed discussion of streams, see the Streams chapter of the *Helix Core Server User Guide*.

Private editing of streams

If a user edits a stream spec publicly and saves the spec, the changes take effect immediately. In some cases, this can affect other users in unwanted ways. For example, changing any dependency, such as product component, might break a product build unless corresponding file changes are made as well.

Private editing of streams is a 2019.1 feature that allows you to:

- modify a stream spec and stream depot files in isolation from other users of that stream
- test the modifications before submitting them, and you can shelve the modifications for another user to test
- improve the visibility and tracking of how the stream spec evolves because a single "atomic" changelist can contain both your edits to the stream spec and any associated code changes

> **Note**
> Prior to the 2019.1 release, a user could edit a stream spec privately by using the `p4 stream edit` command, which is still supported. For example,
> ```
> p4 -c aStreamClient stream edit
> Stream //Ace/Main@15973 - edit stream spec default change
> ```
> However, if you use `p4 stream edit`, the changelist does not track edits to the stream spec.

The workflow for private editing of streams corresponds to your normal workflow with the "p4 edit" on page 180, "p4 submit" on page 558, "p4 revert" on page 481, and "p4 resolve" on page 458 commands. These commands have been enhanced with the `-So`, `-Si`, or `-Sx` options to include or exclude stream specs edits.

See also the `p4 help streamcmds` and `p4 help openablestreamspecs` command-line output.

Tip

- To learn how to create a stream depot and set its `StreamDepth`, see "Working with stream depots" on page 151 in the "p4 depot" on page 146 command.
- For a complete explanation of the different types of streams and how to use them, see the Streams chapter in the *Helix Core Server User Guide*, including the "Update streams" topic.

Stream and graph depot - .git suffix and repo path

A stream can import a repo from a depot of type graph. Within the stream, the content of the import is read-only. The examples below show that:

- the `.git` suffix is required
- the repo path must include a specific `SHA-1` or `ref`

`@sha`	Syntax:
	`import path/... //repo/name.git/restricted/view/path/...@sha`
	Example:
	`import d3/... //repo/d3.git/...@c2e37352ac84eb8f90bc2866f715`
	The imported data is locked at the time of the `@sha`.

`@ref`	Syntax:
	`import path/... //repo/name.git/restricted/view/path/...@ref`
	Example:
	`import d3/... //repo/d3.git/...@refs/heads/master`
	or
	`import d3/... //repo/d3.git/...@master`
	A sync updates the data to the current time of the `@ref`.

See "Working with depots of type graph" on page 153.

"Syntax" on page 19

Form Fields

Field Name	Type	Description
`Stream:`	Writable, mandatory	Specifies the stream's name (permanent identifier) and its path in the stream depot, in the form ***`//depotname/streamname`***. Be aware of the "Limitations on characters in filenames and entities" on page 699.
`Update:`	Read-only	The date the stream specification was last modified.
`Access:`	Read-only	The date and time that the stream specification was last accessed by any Helix server command.
`Owner:`	Writable, mandatory	The Helix server user or group who owns the stream. The default is the user who created the stream.
`Name:`	Writable	Display name of the stream. Unlike the `Stream:` field, this field can be modified. Defaults to the ***`streamname`*** portion of the stream path.
`Parent:`	Writable	The parent of this stream. Must be `none` if the stream's `Type:` is `mainline`, otherwise must be set to an existing stream identifier of the form ***`//depotname/streamname`***.

Field Name	Type	Description
`Type:`	Writable, mandatory	The stream's type determines the expected flow of change. Valid stream types are **`mainline`**, **`development`**, **`release`**, **`virtual`**, and**`task`**. - **`mainline`** The mainline stream is the parent of all streams in the stream depot. Every stream depot must have at least one mainline stream. - **`virtual`** Virtual streams allow merging and copying between parent and child streams without storing local data. Data is passed through to the destination (a non-virtual stream) after applying restrictions on the scope of files defined in the virtual stream's view. Because virtual streams do not have files in their depot namespace, it is impossible to import a virtual stream. - **`release`** More stable than the mainline. Release streams copy from the parent and merge to the parent. - **`development`** Less stable than the mainline. Development streams expect to merge from parent streams and copy to the parent. **Note** The default is stream type is **`development`**. - **`task`** Task streams are lightweight, short-lived branches that are useful for bug fixing or new features that only modify a small subset of the branch data. To keep repository metadata to a minimum, shadow tables track only branched (copied) files. The shadow tables are removed when the task stream is deleted or unloaded. **Note**

Field Name	Type	Description
		Workspaces associated with task streams see all branched files, but only modified and promoted files are visible to users with access to the stream's namespace. For example, if you run "p4 describe" on page 157 for a task stream changelist from a workspace that is not associated with the task stream, you might see only a subset of the submitted files **Tip** You cannot submit files to an **import+** path in a task stream. See the explanation of the path type, "import+ view_path [depot_path]" on page 546.
`Description:`	Writable, optional	Description of the stream.

Field Name	Type	Description
`Options:`	Writable	Settings that configure stream behavior as follows: ■ `[un]locked` Enable/disable other users' ability to edit or delete the stream. If locked, the stream specification cannot be deleted, and only its owner can modify it. The default is `unlocked`. ■ `[all|owner]submit` Specifies whether all users or only the owner of the stream can submit changes to the stream. The default is `allsubmit`. If the `Owner:` of a stream marked `ownersubmit` is a group, all users who are members of that group can submit changes to the stream. ■ `[no]toparent` Specifies whether integrations from the stream to its parent are expected. The default is `toparent`. ■ `[no]fromparent` Specifies whether integrations to the stream from its parent are expected. The default is `fromparent` for mainline and development streams, and `nofromparent` for release streams. ■ `mergeany | mergedown` Specifies whether the merge flow is restricted or whether merge is permitted from any other stream. For example, the `mergeany` option would allow a merge from a child to a parent with no warnings. A `virtual` stream must have its flow options set to `notoparent` and `nofromparent`. Flow options are ignored for `mainline` streams.

"Syntax" on page 19

Field Name	Type	Description
`Paths:`	Writable	Paths define how files are incorporated into the stream structure. Specify paths using the following format: `path_type view_path [depot_path]` where ***`path_type`*** is a single keyword, ***`view_path`*** is a file path with no leading slashes, and the optional ***`depot_path`*** is a file path beginning with `//`. The default path is `share ...` Valid path types are: ■ `share` ***`view_path`*** Specified files can be synced, submitted, and integrated to and from the parent stream. ■ `isolate` ***`view_path`*** Specified files can be synced and submitted, but cannot be integrated to and from the parent stream. ■ `import` ***`view_path`*** `[`***`depot_path`***`]` Specified files can be synced, but cannot be submitted or integrated to and from the parent stream. The ***`view_path`*** is mapped as in the parent stream's view, or to an (optional) ***`depot_path`***. The ***`depot_path`*** can include a changelist specifier. Client workspaces associated with the stream only see revisions at that change or lower within that depot path. For example, if you specify `//depot/import/...@1000` as the depot path, revisions from changelists greater than 1000 are hidden from most commands. **Note** Instead of a changelist number, you can use an automatic label specification. The changelist limits for a given stream workspace are displayed in a read-only client workspace specification field called `ChangeView`. ■ `import+` ***`view_path`*** `[`***`depot_path`***`]`

Field Name	Type	Description
		Functions like a standard **`import`** path, enable you to map a path from outside the stream depot to your stream, but unlike a standard import path, you can submit changes to the files in an **`import+`** path. ■ **`exclude`** ***`view_path`*** Specified files cannot be synced, submitted or integrated to and from the parent stream. By default, streams inherit their structure from the parent stream (except mainlines, which have no parent). Paths are inherited by child stream views. A child stream's path can downgrade the inherited view, but not upgrade it. (For example, a child stream can downgrade a **`shared`** path to an **`isolated`** path, but if the parent stream defines a path as **`isolated`**, its child cannot restore full access by specifying the path as **`shared`**.) Note that the ***`depot_path`*** is relevant only when the ***`path_type`*** is **`import`** or **`import+`**.

Tip
In a virtual stream, when you import paths, be specific and provide **`import view-path`** with a depot-path argument. For example, suppose that you create this set of streams:

```
Stream //project1/mainQ mainline none
'mainQ'
Stream //project1/devQ1 development
//project1/mainQ 'devQ1'
Stream //project1/devQ1-virt virtual
//project1/devQ1 'devQ1-virt'
```

and you use **`Paths: share ...`** for all except the virtual stream, which has:

Paths:

```
import foo/...
share foo/src/libs/project1/...
```

Field Name	Type	Description

		Although you might expect that the `foo` path in the virtual stream comes from its parent, `//project1/devQ1`, the client view shows:
		`//project1/mainQ/foo/... foo/...`
		Therefore, we recommend that you use this more explicit syntax:
		`import foo/... //project1/devQ1/foo/...`

		Note
		To manage files of similar file-type in your stream specs, consider using wildcards (... and *) explicitly following the final slash in the path definition:
		`share ...`
		` isolate BIN/....CLASS`
		` isolate BIN/....exe`
		` import imports/lib/*.a //depot/3rd_party/lib/*.a`
		` import imports/lib/*.dll //depot/3rd_party/lib/*.dll`
		` import imports/src/*.h //depot/3rd_party/src/*.h`
		` exclude doc/....fm`

"Syntax" on page 19

Field Name	Type	Description
`Remapped:`	Writable, optional	Reassigns the location of workspace files. To specify the source path and its location in the workspace, use the following syntax: ***`view_path_1 view_path_2`*** where ***`view_path_1`*** and ***`view_path_2`*** are Helix server view paths (omit leading slashes and leading or embedded wildcards, but terminal wildcards are fine). For example, to ensure that files are synced to the local ProjectX folder, remap as follows: `... projectX/...` Line ordering in the `Remapped:` field is significant: if more than one line remaps the same files, the later line takes precedence. Remappings are inherited by child streams and the workspaces associated with them.

`Ignored:`	Writable, optional	A list of file or directory names to be ignored in client views. For example:
		`/tmp # ignores files named "tmp"`
		`/tmp/... # ignores directories named "tmp"`
		`.tmp # ignores file names ending in .tmp`
		Lines in the `Ignored:` field can appear in any order. Ignored files and directories are inherited by child stream client views.

Options

`-d` ***`streamname`***	Delete the stream specification. A stream specification cannot be deleted if it is referenced by child streams or stream client workspaces. Deleting a stream does not remove its files. However, changes can no longer be submitted to the stream.
`-f`	Administrators can use the `-f` option to delete or modify locked streams owned by other users.
`-i`	Read the stream specification from standard input.

`-o`	Write the stream specification to standard output. By default, it outputs the stream associated with the current workspace. If you pass the **`stream@change`** argument, Helix server uses the version of the stream as of the specified changelist. **Warning** Limitation: Although this option requires the user have at least the `list` access to the stream path, it ignores any other entry in the protections table, including any minus sign (`-`) that would otherwise block the operation.
`-o -v`	Verbose option; includes the automatically-generated client view for this stream.
`-P` *`parent`*	When creating a new stream specification, specify the stream's parent. (This option has no effect on an existing stream specification.)
`-t` *`type`*	When creating a new stream specification, you must specify the stream's type: either `mainline`, `development`, `release`, `task`, or `virtual`.
`-as`	For **`p4 stream resolve`**, performs a "safe" resolve; it skips fields that need merging.
`-am`	For **`p4 stream resolve`**, resolves by merging; skips fields with conflicts.
`-af`	For **`p4 stream resolve`**, forces a concatenation of text fields with conflicts.
`-at`	For **`p4 stream resolve`**, forces acceptance of theirs, overwrites yours.
`-ay`	For **`p4 stream resolve`**, forces acceptance of yours, overwrites theirs.
`-o -v`	For **`p4 stream resolve`**, previews which fields require resolve.
`-n`	For **`p4 stream resolve`**, outputs the base used for the merge.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open` or `list` to use the `-o` option

- As the name implies, task streams are intended to be short-lived; after you have finished using a task stream by promoting your changes to its parent, delete the task stream.

Examples

`p4 stream -t mainline //streams/product1`	Create a mainline stream in the depot named streams
`p4 stream -t development -P main //projectX/bruno-dev`	Create a development stream for project X by branching the mainline.

Related Commands

List streams	`p4 streams`
Edit template for streams	"p4 streamspec" on page 554
Create stream depot	`p4 depot`

p4 streamlog

List revision history of the listed streams.

"Syntax" on page 19

```
p4 [g-opts] streamlog [ -l -L -t -m max ] stream1 ...
```

Description

You can specify any number of streams. For example,

```
p4 streamlog //streams/main //streams/dev2.3 //streams/rel2.3
```

List the revision history of the specified streams, from the most recent revision to the first. If the stream was opened for edit and submitted, the changelist information is displayed. Otherwise only the maximum change number at the time of edit is displayed.

Options

`-t`	Display the time as well as the date.
`-l`	List the full text of the changelist descriptions.
`-L`	List the full text of the changelist descriptions, truncated to 250 characters if longer.

`-m`	Display at most ***`max`*** revisions per stream argument specified.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`open`

p4 streams

Display a list of streams.

Syntax

```
p4 [g-opts] streams [-U] [-F filter] [-T fields] [-m max]
[streamPath ...]
```

Description

Lists the streams defined in the currently connected service. To filter the list, for example, to list streams for a particular depot, specify the ***`streamPath`***.

Options

`-F` *`filter`*	Filter the output according to the contents of specified fields.
`-m` *`max`*	Maximum number of streams to list.
`-T` *`fields`*	Limit field output to fields specified in a list of ***`fields`***. Field names may be separated by a space or comma. Intended for scripting. This option forces tagged output.
`-U`	Display task streams unloaded with `p4 unload`.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

Examples

List the streams in the jam depot	`p4 streams //jam/...`

List the release streams owned by Bruno	`p4 streams -F "Owner=bruno Type=release"`

Related Commands

Create, edit or delete a stream	`p4 stream`

p4 streamspec

Edit the streams template to change, add, or remove spec fields for stream forms.

"Syntax" on page 19

```
p4 [g-opts] streamspec
p4 [g-opts] streamspec -o
p4 [g-opts] streamspec -i
```

Description

This format is used by "p4 stream" on page 539 when streams are entered or updated, and by "p4 streams" on the previous page when displaying a list of streams.

This command brings up a form with the following fields:

`Fields:`	A list of the fields maintained for each stream, one line per field. Each line has five words: code, name, data-type, len, and field-type.

`code`	a unique integer identifier for storing the data of the field. **Important** When adding a new field in 2019.2 or later, the administrator can enter its code by using the optional placeholder value **`NNN`**. If the administrator choses this option, the server will assign an appropriate value. Upon saving the streamspec, the next available code value in the `701-749` range will be automatically generated and saved. The values `701` through `712` are reserved for Helix Core, so the effective range is currently `713-749`. If the number of fields exhausts the range, new codes are assigned unique values greater than or equal to `10000`. Alternatively, the administrator can specify a numeric value. When the administrator saves the spec, a message might indicate that the specified value is not available.
`name`	the name of the field for the stream.
`data-type`	indicates the format of the field: `word`: a single word (any value) `date`: a date/time field `select`: one of a set of words `line`: a one-liner `text`: a block of text
`len`	the recommended character length of a display box for the field. If `0`, a text box is assumed.
`field-type`	indicates how to handle the setting of the field: `optional`: no default, and not required to be present `default`: default provided, but not required `required`: default provided, value must be present `once`: set once to the default and never changed `always`: always set to the default when saving the form

`Values:`	A list of `select` fields and the values that the fields can have. Each line has two words: the field name and the values list, with individual values separated by a forward slash `/` (no spaces).
`Presets:`	A list of fields and their default values, for fields whose `setting` flag is other than `optional`. Each line has two words: the field name and the default value. If the value has spaces, it must be enclosed in double quotes. The following special defaults are recognized: `$user`: the user entering the stream `$now`: the current date `$blank`: the words `<enter description here>`
`Openable:`	A list of versioning options when editing a stream spec. Each line has two words: the field name and the value. Values are: `none`: cannot be changed in a stream edit `isolate`: can be changed only in a stream edit `propagate`: can be changed in a stream edit or integ
`Comments:`	Text to be included at the top of each stream specification, to help the user fill out the form. Each line must begin with the comment character `#`
`g-opts`	See "Global options" on page 690.

Options

`-i`	Read the stream template from the standard input. The user's editor is not invoked.
`-o`	Write the stream template to the standard output. The user's editor is not invoked.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`admin`

Related Commands

List streams	`p4 streams`

Create a stream	"p4 stream" on page 539
Create stream depot	`p4 depot`

p4 submit

Commit a pending changelist and the files it contains to the depot, or submit an open stream spec.

"Syntax" on page 19

```
p4 [g-opts] submit [-r -s] [-f submitoption] [-b | --noretransfer
0|1]
p4 [g-opts] submit [-r -s] [-f submitoption -b] file
p4 [g-opts] submit [-r] [-f submitoption -b] [-So] -d description
p4 [g-opts] submit [-r] [-f submitoption -b] [-Sx] -d description
[file]
p4 [g-opts] submit [-r] [-f submitoption] [-b | --noretransfer
0|1] -c change
p4 [g-opts] submit -e shelvedchange [-b]
p4 [g-opts] submit -i
p4 [g-opts] submit [-r -s] [-f submitoption -b] --
parallel=threads=N[,batch=N][,min=N]
```

Description

When a file has been opened by `p4 add`, `p4 edit`, `p4 delete`, or `p4 integrate`, the file is listed in a ***changelist***. The user's changes to the file are made only within the client workspace copy until the changelist is sent to the depot with `p4 submit`.

In addition to the files being submitted, any open stream specification is also submitted. To submit only files and not an open stream spec, see "streams and p4 submit" on page 563. For more information on open stream specifications, see `p4 stream`.

By default, files are opened within the default changelist, but you can also create new numbered changelists with `p4 change`.

- To submit the default changelist, use `p4 submit`.
- To submit a numbered changelist, use `p4 submit -c changelist`.

  Using the `-c` option also allows you to change the description information for a numbered changelist.

By default, all files in the changelist are submitted to the depot, and files open for `edit`, `add`, and `branch` are closed when submitted, whether there are any changes to the files or not. To change this default behavior, set the `SubmitOptions:` field in the `p4 client` form for your workspace. To override your workspace's `SubmitOptions:` setting from the command line, use `p4 submit -f` *`submitoption`*.

When used with the default changelist, `p4 submit` brings up a form for editing in the editor defined by the `EDITOR` (or `P4EDITOR`) environment variable. Files can be deleted from the changelist by deleting them from the form, but these files will remain open in the next default changelist. To close a file and remove it from all changelists, use `p4 revert`.

All changelists have a `Status:` field. The value of this field is `pending` or `submitted`:

- Submitted changelists have been successfully submitted with `p4 submit`.
- Pending changelists have been created by the user but not yet been submitted successfully.

To supply a changelist description from the command line, use the `-d` option. No change description dialog is presented. The `-d` option works only with the default changelist, not with numbered changelists.

A file's location in the depot is determined by its location in the local filesystem and by the client workspace definition, which is specified in the `p4 client` form. See the "Configure clients" chapter in the *Helix Core Server User Guide*.

Submit processing

`p4 submit` works atomically: either all the files listed in the changelist are saved in the depot, or none of them are. The atomic nature of `p4 submit` allows files to be grouped in a changelists according to their purpose. For example, a single changelist might contain changes to three files that fix a single bug. `p4 submit` fails if it is interrupted, or if any of the files in the changelist are not found in the current client workspace, are locked in another client workspace (with `p4 lock`), or require resolution and remain unresolved.

A progress indicator is available for `p4 submit` if you request it with `p4 -I submit`.

Before committing a changelist, `p4 submit` briefly locks all files being submitted. If any file cannot be locked or submitted, the files are left open in a numbered pending changelist. By default, the files in a failed submit operation are left locked unless the "submit.unlocklocked" on page 827 configurable is set. If `submit.unlocklocked` is set, files are unlocked even if they were manually locked prior to submit if submit fails.

If `p4 submit` fails while processing the default changelist, the changelist is assigned the next number in the changelist sequence, and the default changelist is emptied. The changelist that failed submission must be resubmitted by number after the problems are fixed.

If `p4 submit` fails, some or all of the files might have been copied to the server. By default, retrying a failed submit transfers all these files again unless the "submit.noretransfer" on page 823 configurable is set, in which case the server attempts to detect if the files have already been transferred and does not re-transfer all files when retrying a failed submit. You can use the `--noretransfer` option to override the "submit.noretransfer" on page 823 configurable and allow the user to choose the preferred re-transfer behavior for the current submit operation.

"Syntax" on page 19

Parallel submits

You can transfer files in parallel during the submit process. If there are sufficient resources, a submit command might execute more rapidly by transferring multiple files in parallel. For this feature to work, you must have both server and client upgraded to version 2015.2 or later. Please read this section in its entirety to make sure that you are using this feature appropriately.

To enable parallel submits, set the "net.parallel.max" on page 781 configurable.

- Specify `threads=N` to request that files be sent concurrently using the specified number of independent network connections. The threads grab work in batches. You specify `batch=N` to control the number of files in a batch.

  A submit that is too small will not initiate parallel file transfers. Use the `min` option to control the minimum number of files in a parallel submit.

  If the `"net.parallel.max" on page 781` configuration variable is not set, the command will execute without using parallel threads.

  If the requested number of parallel threads exceeds the value set for `net.parallel.max`, the command will use the maximum number of allowed threads.

- Parallel submits from an edge server to a commit server use standard pull threads to transfer the files. The administrator must ensure that pull threads can be run on the commit server by doing the following:

  - Make sure that the service user used by the commit server is logged into the edge server.
  - Make sure the `ExternalAddress` field of the edge server's server spec is set to the address that will be used by the commit server's pull threads to connect to the edge server.

    If the commit and edge servers communicate on a network separate from the network used by clients to communicate with the edge server, the `ExternalAddress` field must specify the edge server ip address and port number that is used for connections from the commit server. Furthermore, the edge server must listen on the two (or more) networks.

    Important
    Limitation: Parallel submit is not supported from an edge server if the commit server is on a Windows platform. In this case `--parallel` or automatic parallel processing for submit will be silently ignored, and the submit will run without using parallel threads.

- The `--parallel` option is ignored when the archives are shared, for `p4 submit -e`, and when progress indicators are used.

  Note
  To enable automatic parallel processing for **p4 submit**, set non-zero values for both configurables: "net.parallel.max" on page 781 and "net.parallel.submit.threads" on page 784. For example:

  ```
  "p4 configure" on page 122 set net.parallel.max=50
  p4 configure set net.parallel.submit.threads=3
  ```

If automatic parallel processing is enabled, the following configurables no longer have the default value of `0`:

Configurable	New value	Description
"net.parallel.batch" on page 780	8	for **p4 sync**, the number of files in a batch
"net.parallel.batchsize" on page 780	512 KB	for **p4 sync**, the number of bytes in a batch
"net.parallel.min" on page 782	9	for **p4 sync**, the mininum number of files in a batch
"net.parallel.minsize" on page 782	576 KB	for **p4 sync**, the minimum of bytes in a parallel sync
"net.parallel.submit.batch" on page 783	8	for **p4 submit**, number of files in a batch
"net.parallel.submit.min" on page 783	9	or **p4 submit**, mininum number of files in a batch

In this case, you can use any of the syntax variants for the `p4 submit` command (without specifying the `--parallel` option) and processing will be automatically done in parallel.

- If you do use the `--parallel` option explicitly and you have the `net.parallel.*` configurables set, the configurable values you specify on the command line override the value of the configurables.
- You can turn off automatic parallel submit by unsetting the `net.parallel.submit.threads` configurable.
- You can disable the parallel submit configurable settings by specifying `p4 submit --parallel=0`.

Performance and parallel submits

Using parallel submits improves performance in cases like the following:

- Significant network latency exists somewhere along the path through which the submitted file content travels from the client to the repository where the file content is stored.

  This includes significant network latency between a Proxy and Server, or between an Edge Server and a Commit Server. When using parallel submit in such a configuration, the inherent TCP delays related to network latency occur concurrently, rather than sequentially when not using parallel submit.

- Significant resources are required during the transfer of the submitted file, and those resources are available.

  For example, if significant CPU cycles are required to compress ctext or binary file content as it is transferred from a client to a server, the compression of the file content can occur on one CPU core per parallel submit thread compressing either a ctext or binary file, so long as there are enough available CPU cores.

In other cases, using parallel submit might not result in significant performance benefits:

- In some environments, network bandwidth can be a precious resource.

  If network latency is minimal, it might not take many parallel submit threads to use the available network bandwidth. Once the available network bandwidth is used, adding parallel submit threads might not improve performance. This is especially true when transferring file content for which only network bandwidth resources are needed, such as when transferring ubinary files.

- Using a small value for the **`batch`** and **`min`** arguments specified with the **`--parallel`** option is only practical in some cases.

  For example, if a small number of large ctext or binary files are submitted using parallel submit, transferring a small number of files per parallel submit thread can result in the best performance, provided that adequate CPU and network bandwidth resources are available. In order for parallel submit to transfer an evenly-distributed number of files over the number of parallel submit threads specified (which defaults to four), the **`batch`** argument might need to be set to a value lower than its default of eight. (For example, if submitting eight large ctext or binary files using four parallel submit threads, the **`batch`** argument should be set to two.) And it follows that the value for the **`min`** argument, which defaults to nine, should be set to less than or equal to the number of large ctext or binary files being submitted.

  On the other hand, using a small value for the **`batch`** argument can degrade performance when submitting many small files using parallel submit. The overhead of the server frequently querying **`db.sendq`** for each batch by each parallel submit thread can result in **`db.sendq`** concurrency issues. This is because as the size of the files submitted using parallel submit decreases, the more frequently the server queries **`db.sendq`** for the next batch processed by a parallel submit thread.

Background archive transfer for edge server submits

This feature is designed to reduce the amount of wait time end-users perceive when submitting files.

The **`-b`** option of the **p4 submit** command can be used on an edge server to enable the edge archive to be transferred to the commit server in the background. The submit process on the commit server schedules the transfers and completes the submit **without the end user waiting for transfers to finish**. The archives on the originating edge server are transferred to the commit server by pull threads initiated by submit on the commit server.

Submitting pending (not shelved) changes using the **`-b`** option causes any change-submit triggers to be fired before edge-content triggers. An administrator must ensure that pull threads can be run on the commit server. The edge server IP address and port number used by the commit server to connect to the edge server must be specified in the **`ExternalAddress:`** field of the edge server spec.

Background archive transfer is incompatible with:

- the command-line `--noretransfer` option
- the "submit.noretransfer" on page 823 configurable

> **Note**
> - Archives of `ktext` files are transferred serially because archive content depends on commit server work.
> - This feature does not support graph depots.

To learn how to enable this feature, see "Background archive transfer for edge server submits" in the *Helix Core Server Administrator Guide*.

streams and p4 submit

2019.1 introduced the `-So` option to **s**ubmit **o**nly the open stream spec.

By default, an open stream spec is not submitted and `-Sx` makes this behavior explicit.

See "Examples for submitting streams" on page 568.

Form Fields

Field Name	Type	Description
`Change:`	Read-only	The change number, or `new` if submitting the default changelist.
`Client:`	Read-only	Name of current client workspace.
`User:`	Read-only	Name of current Helix server user.
`Status:`	Read-only, value	One of `pending`, `submitted`, or `new`. Not editable by the user. The status is: ■ `new` when the changelist is created. ■ `pending` when it has been created but has not yet been submitted to the depot with `p4 submit`. ■ `submitted` when its contents have been stored in the depot with `p4 submit`.
`Description:`	Writable	Textual description of changelist. This value *must* be changed.

Field Name	Type	Description
`Jobs:`	List	A list of jobs that are fixed by this changelist. This field does not appear if there are no relevant jobs. Any job that meets the jobview criteria as specified on the `p4 user` form are listed here by default, but can be deleted from this list.
`Type:`	Writable, value	Type of change: `restricted` or `public`. A restricted shelved or committed changelist denies access to users who do not own the changelist and who do not have list permission to at least one file in the changelist. A restricted pending (unshelved) changelist denies access to non-owners of the changelist. Public changes are displayed without these restrictions.
`Files:`	List	A list of files being submitted in this changelist. Files can be deleted from this list, but cannot be changed or added.

Options for files

`-b`	See the section above, ""Background archive transfer for edge server submits" on page 562".
`-c` ***`change`***	Submit changelist number ***`change`***. Changelists are assigned numbers either manually by the user with `p4 change`, or automatically by Helix server when submission of the default changelist fails.
`-d` ***`description`***	Immediately submit the default changelist (and the stream, if open) with the ***`description`*** supplied on the command line, and bypass the interactive form. This option is useful when scripting, but does not allow for jobs to be added, nor for the default changelist to be modified.

`-e` ***`shelvedchange`***	Submit shelved changelist number ***`shelvedchange`***. The **`-e`** option submits a shelved changelist without transferring files or modifying the workspace. The shelved change must be owned by the person submitting the change, but the workspace can be different. Files shelved to a stream target can only be submitted by a stream workspace that is mapped to the target stream. In addition, files shelved to a non-stream target cannot be submitted by a stream workspace. To submit a shelved change, all files in the shelved change must be up-to-date and resolved. No files can be open in any workspace at the same change number. The **`p4 client`** form's **`SubmitOptions:`** settings (such as **`revertunchanged`**) are ignored. If the submit is successful, the shelved change and files are no longer available to be unshelved or submitted. This is the only submit option supported for files with propagating attributes from an edge server in a multi-server environment.

"Syntax" on page 19

`-f` ***`submitoption`***	Override the `SubmitOptions:` setting in the `p4 client` form. Valid ***`submitoption`*** values are: ▪ `submitunchanged` All open files (with or without changes) are submitted to the depot. This is the default behavior of Helix server. ▪ `submitunchanged+reopen` All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist. ▪ `revertunchanged` Only those files with content or type changes are submitted to the depot. Unchanged files are reverted. **Note** For DVCS, `revertunchanged` is the default. ▪ `revertunchanged+reopen` Only those files with content or type changes are submitted to the depot and reopened in the default changelist. Unchanged files are reverted and *not* reopened in the default changelist. **Note** Since the two `revertunchanged` options do not revert a file back to the state where the have revision was obtained, you must run `p4 sync -f` to revert the file back to the state where the have revision was obtained. ▪ `leaveunchanged` Only those files with content or type changes are submitted to the depot. Any unchanged files are moved to the default changelist. ▪ `leaveunchanged+reopen` Only those files with content or type changes are submitted to the depot. Unchanged files are moved to the default changelist, and changed files are reopened in the default changelist. This option is similar to `submitunchanged+reopen`, except that no unchanged files are submitted to the depot.
`-i`	Read a changelist specification from standard input. Input must be in the same format as that used by the `p4 submit` form.

`--noretransfer 0|1`	Set to `1` to have the server avoid re-transferring files that have already been archived after a failed submit operation. Set to `0` to have the server retransfer all files after a failed submit operation. This setting overrides the setting of the `submit.noretransfer` configurable for the current submit operation.
`--parallel`	Specify options for parallel file transfer. The configuration variable `net.parallel.max` must be set to a value greater than 1 to enable the `--parallel` option. ▪ `threads=`*`n`* sends files concurrently using *`n`* independent network connections. The specified threads grab work in batches. ▪ `batch=`*`n`* specifies the number of files in a batch. ▪ `min=`*`n`* specifies the minimum number of files in a parallel sync. A sync that is too small will not initiate parallel file transfers. See "Parallel processing" on page 578 .
`-r`	Reopen files for `edit` in the default changelist after submission. Files opened for `add` or `edit` in will remain open after the submit has completed.
`-s`	Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of `closed`. To leave a job unchanged, use the special status of `same`. On new changelists, the fix status is displayed as the special status `ignore`. (If the status is left unchanged, the job is not fixed by the submission of the changelist.) This option works in conjunction with the `-s` option to `p4 fix`, and is intended for use in conjunction with defect tracking systems.
`file`	A single parameter that can be a **path** with '...' as a wildcard. This file pattern parameter can only be used when submitting the default changelist. The files in the default changelist that match the specified pattern are submitted. Files that don't match the file pattern are moved to the next default changelist.
`g-opts`	See "Global options" on page 690.

Options for streams

`-So`	Submit only the only stream spec. No list of files allowed.
`-Sx`	Excludes the open stream spec when submitting the specified list of files

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`write`

Examples for submitting streams

`p4 submit`	Submit the default changelist, including the current open stream spec.
`p4 submit -c 4739`	Submit the numbered changelist, including the current open stream spec.
`p4 submit -So`	Submit only the open stream spec.
`p4 submit -Sx`	Submit the default changelist, but do not submit the open stream spec.

`p4 submit foo bar`
`p4 submit -Sx foo bar`
	Submit the specified changelist, but do not submit the open stream spec.

Related Commands

To create a new, numbered changelist	`p4 change`

To open a file in a client workspace and list it in a changelist	`p4 add`
	`p4 edit`
	`p4 delete`
	`p4 integrate`

To move a file from one changelist to another	`p4 reopen`
To remove a file from all changelists, reverting it to its previous state	`p4 revert`
To view a list of changelists that meet particular criteria	`p4 changes`
To read a full description of a particular changelist	`p4 describe`
To read files from the depot into the client workspace	`p4 sync`
To edit the mappings between files in the client workspace and files in the depot	`p4 client`

p4 submit (graph)

Commit open files to the repo.

"Syntax" on page 19

```
p4 submit [-i -c changelistNumber -d desc --allow-empty]
```

Description

Commits a pending `changelistNumber` and its files to the repo, constructing a new commit and updating the current branch to refer to the new commit. By default, this command attempts to submit all files in the `default` changelist. Submit displays a dialog where you enter a description of the change and, optionally, delete files from the list of files to be checked in. To add files to a changelist before submitting, use any of the commands that open client workspace files such as "p4 add" on page 53 or "p4 edit" on page 180.

Options

`--allow-empty`	Permit submitting with no file changes.
`-c` ***`changelistNumber`***	Submit the specified pending changelist instead of the default changelist. Additional changelists can be created manually, using the "p4 change" on page 83 command, or automatically as the result of a failed attempt to submit the default changelist. The pending changelist is deleted during the submit process. The result of the submit is a new commit identified by its SHA-1.
`-d` ***`description`***	Immediately submit the default changelist with the ***`description`*** supplied on the command line, and bypass the interactive form. This option is useful when scripting, but does not allow for the default changelist to be modified.
`-i`	Read a changelist specification from the standard input. The user's editor is not invoked.

Examples

```
p4 submit -d 'upgrade readme'
```

```
Change 6797 renamed 296fcac8ce99769c3d1bb28a3321c491ec9c81fc and submitted
on //gd1/repo1.
```

p4 switch

Create a stream or switch to a different stream, with an option to populate that stream, or to display current streams.

"Syntax" on page 19

```
p4 [g-opts] switch [-c -m -v -P parent] [-Rx] [-r] [ --no-sync ]
streamstream
p4 switch [-r -v] [-Rx] [ --no-sync ] [stream]@change
p4 switch -l -L
p4 switch
```

Description

This command allows you to create, manage, and switch between your streams. Note that **`p4 switch`** automatically performs a **`p4 reconcile`** and **`p4 sync`** and automatically shelves work in progress when switching between streams.

`p4 switch [-r -v] [-Rx] [ --no-sync ] [stream]@change`
is for reproducing back-in-time view and revision synchronization. Specify a changelist number and, optionally, a stream. The client switches to the stream specification that was available at the time of that change. The workspace syncs to that changelist. If the stream is omitted but **`@change`** is specified, no stream switch is performed, but the workspace is synced to that change.

The **`stream@change`** argument sets the workspace view to match the version of a stream as of the specified changelist, and syncs the files to the versions matching that same changelist.

You cannot switch to a new stream if files are open in a numbered changelist.

If files are open in the default changelist:

- they are shelved and reverted prior to switching to the new stream
- they are automatically unshelved when switching back to this stream

Option to make switching between streams faster

If you want to limit the amount of data and metadata transferred when switching between streams, consider using the **`--no-sync`** option. You can then use " p4 sync" on page 574 on a subset of the files or directories included in that stream.

Options

With no options specified, `p4 switch` displays the current stream.

`-c`	Local (DVCS) server only Specifies that the new stream be created and populated with a copy of the files that are in the current stream.
`-l`	Lists all known streams.
`-L`	Lists all streams that were switched at the specified change with open files.
`-m`	Local (DVCS) server only Specifies that `switch -c` creates a new, empty stream with no parent. This is an independent mainline.
`-P` *`parent`*	Local (DVCS) server only Specifies that `p4 switch -c` creates a new stream with the specified stream as its parent. The new stream is populated with the files from the specified stream, rather than with the files from the current stream.
`-r`	Reopens files in the new mapped location of the specified stream. If you omit this option and you have opened files in the current stream, those files are: ■ shelved before switching to the new stream ■ unshelved when you switch back to this stream
`-Rx`	Controls how files are reconciled when switching between streams: ■ `Ra` reconcile files not currently under Helix server control. (Files open for add.) ■ `Re` reconcile files that have been modified. (Files open for edit.) ■ `Rd` reconcile files that have been removed. (Files marked for delete.) ■ `Rn` reconcile does not run. Without this option, `reconcile` runs as if all options were set: `-Raed`
`--no-sync`	Prevents sync of file content for unopened files.
`-v`	Verbose mode.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open` to use the `-c` or `-r` options, `list` to use the `-L` option, or `write` for default switching

Examples

`p4 switch -r bugfix17`	Switch to the `bugfix17` stream and open all the files in it.

Related Commands

Merge	`p4 merge`
Resolve	`p4 resolve`

p4 switch (graph)

Switch to a different branch, or create a branch.

"Syntax" on page 19

```
p4 switch [ -a | -n repo ] branch
p4 switch -c [ -a | -n repo ] branch
p4 switch -d [ -a | -f | -n repo ] branch
p4 switch -l
```

Description

The switch command moves the client to a different branch, optionally creating that branch.

Use the first form of the command to switch the current branch for your client.

Options

`-l`	List the branches
`-a`	Applies for clients having more than one repo. All repos that contain the specified branch switch to it. Repos that do not contain the specified branch remain on their current branch.
`-c`	Create the specified branch and switch to that branch.
`-f`	Force the deletion of a branch. This option is required when a branch was not merged to master.

`-n`	Named repo. For example, to create a branch by specifying a named repo:
	`p4 switch -c -n //gd1/repo2 dev`
	`Branch refs/heads/dev created in repo //gd1/repo2.`

`-c -a`	Create the specified branch and switch all repos to that branch. Example: **`p4 switch -c -a branch2`**
`-d`	Delete the specified branch.
`-d -a`	Delete the specified branch on all repos. Example: **`p4 switch -d -a`** ***branch2***
`-d -n`	Delete the specified branch on the specified repo. Example: **`p4 switch -d -n //graphDepot/repo2 branch2`** **Tip** We recommend that you not delete the branch you are currently on because this detaches your current branch HEAD from any tag or branch. This means your commit is unnamed and can be removed by a `git gc` (garbage collection).

p4 sync

Update the client workspace to reflect the contents of the depot.

"Syntax" on page 19

```
p4 [g-opts] sync [-f -k -L -n -N -q -r] [-m max] [[FileSpec]
[revSpec]]
```

```
p4 [g-opts] sync [-L -n -N -q -s] [-m max] [[FileSpec][revSpec]]
```

```
p4 [g-opts] sync [-L -n -N -p -q] [-m max] --parallel=threads=n
[,batch=n][,batchsize=n][,min=n][,minsize=n [[FileSpec][revSpec]]
```

Description

p4 sync brings the client workspace into sync with the depot by copying files matching its file pattern arguments from the depot to the client workspace.

When no file patterns are specified on the command line, **p4 sync** copies a particular depot file if it meets all three criteria:

- Visible through the client view
- Not already opened by **p4 edit**, **p4 delete**, **p4 add**, or **p4 integrate**
- Does not already exist in the client workspace at its latest revision, that is, the head revision

In new, empty workspaces, all depot files meet the last two criteria, so all the files visible through the workspace view are copied into the user's workspace.

If file patterns are specified on the command line, only those files that match the file patterns and that meet the above criteria are copied.

If the file pattern contains a revision specifier, the specified revision is copied into the client workspace.

If the file argument includes a revision range, only files included in the revision range are updated, and the highest revision in the range is used. Files that are no longer in the workspace view are not affected if the file argument includes a revision range. See "File specifications" on page 695 on "Using revision ranges", and "p4 archive" on page 67 "Usage Notes", and **p4 help revisions**.

The **p4 sync** command gets the latest version from the depot and updates your local workspace files . However, to protect the person working in the local workspace from accidentally losing work, any files that are open in the local workspace when you run **p4 sync** need to be manually resolved using the p4 resolve command before submitting.

- Newly synced files are read-only
- `p4 edit` and `p4 delete` make the files writable

Important
To make files writable, use Helix Core server commands. Do NOT use operating system commands.

Note
For users of Helix Server for Distributed Versioning (DVCS), when specifying file paths, you can use the global changelist ID from the submitted change spec instead of the actual change number. For example:

```
p4 sync //depot/...@30E7C829-08C504-4109-89AA-904D0C2194B8
```

For more information, see the Using Helix Core Server for Distributed Versioning topic on global changelist IDs, "Track a changelist's identity from server to server".

Options

`-f`	Force the sync. Helix server performs the sync even if the client workspace already has the file at the specified revision. If the file is writable, it is overwritten. This option does not affect open files, but it *does* override the `noclobber` client option (see "p4 client" on page 100).
`-k`	Keep existing workspace files; update the have list without updating the client workspace. Use `p4 sync -k` only when you need to update the have list to match the actual state of the client workspace. `p4 sync -k` is an alias for the `p4 flush`. For additional details and a description of the relevant use cases, see "p4 flush" on page 212. If your administrator has set the `zerosyncPrefix` configurable, *all* workspaces with names that begin with the specified prefix assume `p4 sync -k`.
`-L`	For scripting purposes, perform the sync on a list of valid file arguments in full depot syntax with a valid revision number. When this flag is used, the arguments are processed together by building an internal table similar to a label. This file list processing is significantly faster than having to call the internal query engine for each individual file argument. However, the file argument syntax is strict and the command will not run if an error is encountered.
`-m` ***`max`***	Sync only the first ***`max`*** files specified.
`-n`	Preview mode: Display the results of the sync without actually performing the sync.

`-N`	Preview mode: Display a summary of the expected network traffic associated with a sync, without performing the sync. This tells you how many files are to be added or updated, which is useful if you're dealing with many large files and/or are bandwidth or diskspace-limited. This option is useful for estimating network impact of a sync before attempting to perform the sync. If you've recently updated your client workspace view, it's useful to know if you have inadvertently included a folder tree that holds several gigabytes of assets *before* attempting to sync your newly-configured workspace.
`-p`	Populate a client workspace, but do not update the have list. Any file that is already synced or opened is bypassed with a warning message. This option is typically used for workspaces used in processes (such as certain build or publication environments) where there is no need to track the state of the workspace after it has first been synced. **Note** If you are at a replica, and syncing from the replica's depot to your replica workspace, the sync command will fail if the master is not available. This is because the replica needs to be able to update the `db.have` table on the master. However, in this case, you can use `p4 sync -p`
`--parallel`	Specify options for parallel file transfer. The configuration variable `net.parallel.max` must be set to a value greater than 1 to enable the `--parallel` option. ■ `threads=`*`n`* sends files concurrently using *`n`* independent network connections. The specified threads grab work in batches. There is no default value; a value must be set. ■ `batch=`*`n`* specifies the number of files in a batch. Default value is `8`. ■ `batchsize=`*`n`* specifies the number of bytes in a batch. Default value is `512K`. ■ `min=`*`n`* specifies the minimum number of files in a parallel sync. A sync that is too small does not initiate parallel file transfers. Default value is `9`. ■ `minsize=`*`n`* specifies the minimum number of bytes in a parallel sync. A sync that is too small will not initiate parallel file transfers. Default value is `576K`. See "Parallel processing" on page 578.

`-q`	Quiet operation: suppress normal output messages. Messages describing errors or exceptional conditions are not suppressed.
`-r`	Reopen files that are mapped to new locations in the depot, in the new location. By default, open workspace files remain associated with the depot files that they were originally opened as. For example, pending work can be moved to a different stream by running `p4 client` **-f -s** followed by `p4 sync -r`.
`-s`	Safe sync: Compare the content in your client workspace against what was last synced. If the file was modified outside of the control of Helix server, an error message is displayed and the file is not overwritten. If your client workspace specification has both the `allwrite` and `noclobber` options set, this check is performed by default.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`read`

- If the client workspace view has changed since the last sync, the next sync removes from the client workspace those files that are no longer visible through the workspace view (unless a revision range is used), and copies into the client workspace those depot files that were not previously visible.

  By default, any empty directories in the workspace are cleared of files, but the directories themselves are not deleted. To remove empty directories upon syncing, turn on the `rmdir` option in the `p4 client` form.

- If a user has made certain files writable by using OS commands outside of Helix server's control, `p4 sync` will not normally overwrite those files. However, if the `clobber` option in the `p4 client` form has been turned on, these files will be overwritten.

  You can prevent this behavior (at a minor cost in performance) by using the `-s` "safe sync" option. Even if the `clobber` option is set, `p4 sync -s` will not overwrite files modified outside of Helix server control.

- A progress indicator is available for `p4 sync` if you request it with `p4 -I sync -q`.

- Do NOT issue a `p4 sync` with multiple arguments referencing the same file. For example: `p4 sync depot/project/...@1000 //depot/project/file.txt@1010` will result in unpredictable, inconsistent revisions.

"Syntax" on page 19

Scripting

The `-m max` option is useful when combined with the `-n` option for efficient scripting. For example, a command like `p4 sync -n -m 1` does not sync any files, but displays only one line of output if there are any files to be synced, or a message indicating that the workspace is up to date. Without the `-m 1` option, the output could be thousands of lines long, all of which would be discarded.

The `-L` option is intended for use by scripts or automated reporting processes. File arguments must be in full depot syntax, and have a valid revision number. File specifications that do not meet these requirements are silently ignored. Using this option speeds up file list processing.

Parallel processing

Depending on the number of files being transferred, the `p4 sync` command might take a long time to execute. You can speed up processing by having this command transfer files using multiple threads. Parallel processing is most effective:

- with long-haul, high latency networks
- if the network configuration prevents making full use of the available bandwidth with a single TCP flow
- when working with large, compressed binary files, where the client must perform substantial work to decompress the file

Do one of the following:

- Enable **automatic parallel sync** by setting both "net.parallel.threads" on page 786 and "net.parallel.max" on page 781.
  - For example, if you configure `net.parallel.max=40` and `net.parallel.threads=3`, the `p4 sync` command automatically uses 3 threads.
- **Manually invoke parallel sync** by not setting "net.parallel.threads" on page 786, setting "net.parallel.max" on page 781 to a value greater than `1`, and using the `--parallel` option to the `p4 sync` command.
  - For example, if `net.parallel.threads` is unset, and `net.parallel.max=40`, the number of parallel threads can be anywhere within the range of `2` to `40`. To run 12 threads in parallel, type at the command line `p4 sync --parallel=threads=12`.

Note

- The "net.parallel.max" on page 781 configuration variable can be set to any value between `0` and `100`. A value of `0` or `1` disables parallel processing. A value greater than `1` enables parallel processing up to the specified level.
- If `net.parallel.max=40`, that value of `net.parallel.threads` should be less than or equal to `40`. If the requested number of parallel threads exceeds the value set for `net.parallel.max`, only the maximum number of allowed threads run.

"Syntax" on page 19

- The `--parallel` option allows you to specify **how** the parallel processing occurs.
  - Use the `min` and/or `minsize` suboptions to indicate that you don't want parallel processing unless the sync involves sending at least `min` number of files, or at least `minsize` number of bytes.
  - Use the `batch` and/or `batchsize` suboptions to specify how many files or bytes should be taken at a time. Setting the batch size small should result in the best use of the network, but at the risk of overloading database resources.

  You can specify the suboptions in any order.

You can also control parallel processing behavior by using:

- "net.parallel.batch" on page 780
- "net.parallel.batchsize" on page 780
- "net.parallel.min" on page 782
- "net.parallel.minsize" on page 782
- "net.parallel.threads" on page 786
- "net.parallel.sync.svrthreads" on page 785
- "client.sendq.dir" on page 733

Working with streams

If both of the following are true:

- the `Stream:` field in your client workspace is set to a valid stream
- the `StreamAtChange:` field points to a specific changelist number

the command, `p4 sync`, syncs your workspace to the revisions of files available as of that changelist.

Retrying the command

Over unreliable networks, you can specify the number of retries to attempt and the length of time beyond which the Helix server application assumes that the network has timed out. Set "net.maxwait" on page 777 in your workspace's `P4CONFIG` file or on a one-command basis from the command line, and specify the number of retries with `-r n`, where *n* is the number of times to attempt reconnection.

The command below attempts to sync the user's workspace, making up to three attempts to resume the sync if interrupted. The command fails after the third 60-second timeout.

```
$ p4 -r3 -vnet.maxwait=60 sync
```

Because the format of the output of a command that times out and is restarted cannot be guaranteed (for example, if network connectivity is broken in the middle of a line of output), avoid the use of `-r` on any command that reads from standard input.

Examples

`p4 sync`	Copy the latest revision of all files from the depot to the client workspace, as mapped through the client view. If the file is already open in the client workspace, or if the latest revision of the file exists in the client workspace, it is not copied.
`p4 sync file.c#4`	Copy the fourth revision of `file.c` to the client workspace, with the same exceptions as in the example above.
`p4 sync //depot/proj1/...@21`	Copy all the files under the `//depot/proj1` directory from the depot to the client workspace, as mapped through the client view. Don’t copy the latest revision; use the revision of the file in the depot after changelist 21 was submitted.
`p4 sync @`***`labelname`***	If ***`labelname`*** is a label created with `p4 label`, and populated with `p4 labelsync`, bring the workspace into sync with the files and revision levels specified in ***`labelname`***. Files listed in ***`labelname`***, but not in the workspace view, are not copied into the workspace. Files *not* listed in ***`labelname`*** are deleted from the workspace. (That is, `@`***`labelname`*** is assumed to apply to all revisions up to, and including, the revisions specified in ***`labelname`***. This includes the nonexistent revision of the unlisted files.)
`p4 sync @`***`labelname`***`,@`***`labelname`***	Bring the workspace into sync with a label as with `p4 sync @labelname`, but preserve non-labeled files in the workspace. (The revision range `@`***`labelname`***`,@`***`labelname`*** applies only to the revisions specified in the label name itself, and excludes the nonexistent revision of the unlisted files.)
`p4 sync @2018/06/24`	Bring the workspace into sync with the depot as of midnight, June 24, 2018. (That is, include all changes made during June 23.)
`p4 sync status%40june1st.txt`	Sync a filename containing a Helix server wildcard by using the ASCII expression of the character’s hexadecimal value. In this case, the file in the client workspace is `status@june1st.txt`. For details, see "Limitations on characters in filenames and entities" on page 699.

`p4 sync file.c#none`
or
`p4 sync file.c#0`
	Sync to the nonexistent revision of `file.c` so that the file is deleted from the workspace.

`p4 sync ...#none`
or
`p4 sync ...#0`
	Sync to the nonexistent revision of all files so that all files in the workspace (that are under Helix server control) are removed.

Related Commands

To open a file in a client workspace and list it in a changelist	`p4 add`
	`p4 edit`
	`p4 delete`
	`p4 integrate`

To copy changes to files in the client workspace to the depot	**`p4 submit`**
To view a list of files and revisions that have been synced to the client workspace	**`p4 have`**

p4 sync (graph)

Synchronize the client workspace with its view of the repo.

"Syntax" on page 19

```
p4 sync [-f -n -q -k] [file[commit-sha1 | reference] ...]
```

Description

Sync updates the client workspace to reflect the current contents of the repo (if it has changed).

If a filepath is specified and it matches paths that exist in one or more repos mapped to the client, those repos will be synced. For example,

- If both `repo1` and `repo2` have a branch named `dev`,
  - the command `p4 sync -f dev` will sync the `dev` branch in both repos
  - the command `p4 sync -f //repo/repo1/...@dev` will sync the `dev` branch in `repo1` and will not sync the `dev` branch from `repo2`
- If only `repo1` has a branch named `dev`,

"Syntax" on page 19

- the command `p4 sync -f dev` will sync the `dev` branch in `repo1`
- the command `p4 sync -f //repo/repo1/...@dev` will sync the `dev` branch in `repo1`

Note

- By default, the entire portion of the repo that is mapped to the client will be synced, not just the files in the specified filepath. If a sha or reference is specified, the client will be syned to that commit.
- If a branch reference is specified, the branch with which the client is currently associated will also be updated.

If the client has open files in the specified repo, syncing will only be permitted if the commit that is to be synced is a child of the current commit (a fast-forward).

Sync adds files that have not been retrieved before, deletes previously retrieved files that have been deleted from the repo, and updates files that have been updated in the repo.

Normally, sync does not overwrite workspace files that the user has manually made writable. Setting the `clobber` option in the client specification disables this safety check.

Note

You can sync a client that has a view spec that maps files in a repo. If you do not specify a branch in the repo, `p4 sync` defaults to the `master` branch of the repo. If the client spec maps both a classic depot and a depot of type `graph`, `p4 sync` updates the client workspace for both types of depots. To sync a repo branch or a SHA-1, see the "Examples" on the next page.

For more information about depots of type `graph`, see:

- "Including Graph Depot repos in your client" on page 113 in `p4 client`
- "Working with depots of type graph" on page 153 in `p4 depot`
- "Stream and graph depot - .git suffix and repo path" on page 541 in "p4 stream" on page 539

Options

`-f`	Force the sync. Helix server performs the sync even if the client workspace already has the file at the specified revision. If the file is writable, it is overwritten. This option does not affect open files, but it *does* override the `noclobber` client option (see "p4 client" on page 100).

"Syntax" on page 19

`-k`	Updates server metadata without syncing files. This enables you to ensure that the server reflects the state of files in the workspace while avoiding a large data transfer. **Warning** An erroneous update can cause the server to incorrectly reflect the state of the workspace.
`-n`	Preview mode: Display the results of the sync without actually performing the sync.
`-q`	Quiet operation: suppress normal output messages. Messages describing errors or exceptional conditions are not suppressed.

Usage Notes

Working with a depot of type graph

Examples

`p4 sync dev`
 	
`p4 sync refs/heads/master`
	For a repo in a depot of type **`graph`**, sync the branch named **`dev`** or **`master`**. Both syntax styles are supported.

`p4 sync c2ae39f`	For a repo in a depot of type **`graph`**, sync the Git commit associated with the SHA-1 hash key **`c2ae39f4`**.
`p4 sync refs/tags/`*`v1.8.5-rc3`*	For a repo in a depot of type **`graph`**, sync the repo associated with a tag, such as **`v1.8.5-rc3`**.

p4 tag

Tag files with a label.

"Syntax" on page 19

```
p4 [g-opts] tag [-d -g -n -U] -l labelnameFileSpec[revSpec]
```

Description

Use `p4 tag` to tag specified file revisions with a label. A *labelname* is required. If a label named ***`labelname`*** does not exist, it is created automatically. If the label already exists, you must be the `Owner:` of the label and the label must be `unlocked` in order for you to tag or untag files with the label. (Use `p4 label` to change label ownership or lock status.)

If the ***`file`*** argument does not include a revision specification, the head revision is tagged with the label. If the file argument includes a revision range specification, only files with revisions in that range are tagged. (If more than one revision of the file exists in the specified range, the highest revision in the specified range is tagged.)

Options

`-d`	Delete the label tag from the named files.
`-g`	In multi-server environments, use the `-g` option to specify whether the label being applied is: ■ local to an edge server, or ■ globally available from the commit server.
`-l` ***`labelname`***	Specify the label to be applied to file revisions
`-n`	Display what `p4 tag` would do without actually performing the operation.
`-U`	If tagging files with a new label, set the `autoreload` option of the newly-created label. This option has no effect when used with an existing label.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`list`

- By default, `p4 tag` operates on the head revision of files in the depot. To preserve the state of a client workspace, use `p4 labelsync`, which operates on the revision of files last synced to your workspace.
- With a multi-server Perforce service, `p4 tag` works with a label local to the edge server (to which you are sending a request). The `-g` option can be used to apply a global label, but only with an unbound (global) client workspace.

Local Tag	Global Tag
"rpl.labels.global" on page 800 is unset, which is `0`	"rpl.labels.global" on page 800 set to `1`
by default, labels are **local** to your edge server	by default, labels are **global**
`-g` option provides access to **global** labels on the commit server	`-g` option allows the updating of **local** labels If a label exists on an edge server before `rpl.labels.global` is set, and you want that label to be available on both the edge server and the commit server, unloaded the label from the edge server and reload it on the commit server

Examples

`p4 tag -l rel1 //depot/1.0/...`	Tag the head revisions of files in `//depot/1.0/...` with label `rel1`. If the label `rel1` does not exist, create it.
`p4 tag -l build //depot/1.0/...@1234`	Tag the most recent revisions as of the submission of changelist `1234` of files in `//depot/1.0/...` with label `build`. If the label `build` does not exist, create it.
`p4 files @labelname`	List the file revisions tagged by ***`labelname`***.

"Syntax" on page 19

Related Commands

To create or edit a label	`p4 label`
To list all labels known to the system	`p4 labels`
To tag revisions in your client workspace with a label	`p4 labelsync`

p4 tickets

Display all tickets granted to a user by `p4 login`.

"Syntax" on page 19

```
p4 [g-opts] tickets
```

Description

The `p4 tickets` command lists all tickets stored in the user’s ticket file, which is specified by the "P4TICKETS" on page 685 environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.

Options

g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	none

Examples

`p4 tickets`	Display all tickets stored in a user’s local ticket file.

Related Commands

To start a login session (to obtain a ticket)	`p4 login`
To end a login session (to delete a ticket)	`p4 logout`

p4 triggers

Create or display a list of scripts to be run conditionally whenever changelists are submitted, forms are updated, when integrating Perforce with external authentication or archive mechanisms, when rotating journals, or when pushing or fetching content to and from a remote depot.

"Syntax" on page 19

```
p4 [g-opts] triggers
p4 [g-opts] triggers -o
p4 [g-opts] triggers -i
```

Description

Helix server *triggers* are user-written scripts or programs that are called by a Helix server whenever certain operations (such as changelist submission or changes to forms) are performed. If the script returns a value of `0`, the operation continues; if the script returns any other value, the operation fails.

The **`p4 triggers`** command includes three variants:

- With no options specified, the command invokes the default editor to allow the user to specify one or more trigger definitions.
- The `-i` option specifies that the user use standard input to specify one or more trigger definitions.
- The `-o` option displays the trigger definitions currently stored in the trigger table.

A trigger definition contains four fields that specify the name of the trigger, the type of event that should trigger the execution of the script, the location of the script, and other trigger type-dependent information. When the condition specified in a trigger definition is satisfied, the associated script or program is executed.

For detailed information about writing triggers and trigger definitions, see "Using triggers to customize behavior" in the *Helix Core Server Administrator Guide*.

Options

`-i`	Read the trigger table from standard input without invoking the editor.
`-o`	Write the trigger table to standard output without invoking the editor.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super`

Related Commands

To obtain information about the changelist being submitted	`p4 describe`
	`p4 opened`

To aid daemon creation	`p4 review`
	`p4 reviews`
	`p4 counter`
	`p4 counters`
	`p4 user`

p4 trust

Establish trust of an SSL connection to a Perforce service.

Use the command **`p4 trust -h`** to get help for the server (if you have not yet trusted your server). You must do this because the command is implemented on the client rather than the server. When a command begins with **`p4 help`**, the client forwards it to the server to construct the text of what is displayed on the client machine. In the case of the trust command, the client might not trust the server to send any commands to it, so the help must begin with **`p4 trust`** to get it serviced locally by the client rather than have it forwarded to the server.

Syntax

```
p4 [g-opts] trust [-l -y -n -d -f -r] [-i fingerprint]
```

Description

Use **`p4 trust`** to manage the **`P4TRUST`** file (by default, **`.p4trust`** in your home directory) to establish (or manage) the trust of an SSL connection.

The trust file contains the fingerprints of the keys received for SSL connections. When you first connect to a Perforce service, you are prompted with its fingerprint; if the fingerprint is correct, you can use **`p4 trust`** to add the service's fingerprint to your trust file. If the fingerprint changes (or expires), subsequent attempts to connect to that service will result in warning or error messages.

Your system administrator can help you confirm the accuracy of any fingerprint (or change to a fingerprint) provided to you by a Perforce service.

Only after you have added an SSL-enabled Perforce service to your **`P4TRUST`** file can you connect to it by setting **`P4PORT`** to **`ssl:`*`hostname`*`:`*`port`***.

Options

`-d`	Delete an existing trusted fingerprint.
`-f`	Force the replacement of a mismatched fingerprint.
`-i` ***`fingerprint`***	Install the specified ***`fingerprint`***.
`-l`	List all known fingerprints on this client workstation.
`-n`	Automatically refuse any prompts.

`-r`	List, install, or delete a replacement fingerprint. If a replacement fingerprint exists for the connection, and the primary fingerprint does not match (but the replacement fingerprint does), the replacement fingerprint replaces the primary. This option may be combined with the `-l`, `-i`, or `-d` options.
`-y`	Automatically accept any prompts.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`none`

p4 typemap

Modify the file name-to-type mapping table.

"Syntax" on page 19

```
p4 [g-opts] typemap
p4 [g-opts] typemap -o
p4 [g-opts] typemap -i
```

Description

The `p4 typemap` command allows Helix server administrators to set up a table linking Helix server file types to file name specifications. If a filename matches an entry in the typemap table, it overrides the file type that would otherwise have been assigned by Helix server.

By default, Helix server automatically determines if a file is of type `text` or `binary` based on an analysis of the first 65,536 bytes of a file. If the high bit is clear in each of the first 65,536 bytes, Helix server assumes it to be `text`; otherwise, it's `binary`. Files compressed in the `.zip` format (including `.jar` files) are also automatically detected and assigned the type `ubinary`.

Although this default behavior can be overridden by the use of the `-t` ***`filetype`*** option, it's easy to overlook this, particularly in cases where files' types were usually (but not always) detected correctly. This situation occasionally appears with PDF files (which sometimes begin with over 65,536 bytes of ASCII comments) and RTF files, which usually contain embedded formatting codes.

The `p4 typemap` command provides a more complete solution, allowing administrators to bypass the default type detection mechanism, ensuring that certain files (for example, those ending in `.pdf` or `.rtf`) will always be assigned the desired Perforce filetype upon addition to the depot. See "p4 add" on page 53.

Users can override any file type mapping defined in the typemap table by explicitly specifying the file type on the Helix server command line.

Form Fields

The `p4 typemap` form contains a single `TypeMap:` field. Each indented line under the `TypeMap:` field consists of a pair of values linking file types to file patterns specified in depot syntax:

Column	Description
`filetype`	Any valid Helix server file type. For a list of valid file types, see "File types" on page 707.
`pattern`	A file pattern in depot syntax. When a user adds a file matching this pattern, its default file type is the file type specified in the table. To exclude files from the typemap, use exclusionary (`-pattern`) mappings.

Options

`-i`	Reads the typemap table from standard input without invoking the editor.
`-o`	Writes the typemap table to standard output without invoking the editor.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`admin`, or `list` to use the `-o` option

- To specify all files with a given extension at or below a desired subdirectory, use four periods after the directory name, followed by the extension. (for instance, `//path/....ext`) The first three periods specify "all files below this level". The fourth period and accompanying file extension are parsed as "ending in these characters".
- File type modifiers can be used in the typemap table. Useful applications include forcing keyword expansion on or off across directory trees, enforcing the preservation of original file modification times (the `+m` file type modifier) in directories of third-party DLLs, or implementing pessimistic locking policies.
- Specify multiple file type modifiers consecutively. For example, `binary+lFS10` refers to a `binary` file with exclusive-open (`l`), stored in full (`F`) rather than compressed, and for which only the most recent ten revisions are stored (`S10`). For more information on syntax, see "File types" on page 707.
- If you use the `-t` option and file type modifiers to specify a file type on the command line, and the file to which you are referring falls under a `p4 typemap` mapping, the file type specified on the command line overrides the file type specified by the typemap table.

Examples

To tell the Perforce service to regard all PDF and RTF files as **binary**, use **p4 typemap** to modify the typemap table as follows:

```
Typemap:
        binary //....pdf
        binary //....rtf
```

The first three periods ("**...**") in the specification are a Helix server wildcard specifying that all files beneath the root directory are included as part of the mapping. The fourth period and the file extension specify that the specification applies to files ending in **.pdf** (or **.rtf**).

A more complicated situation might arise in a site where users in one area of the depot use the extension **.doc** for plain ASCII text files containing documentation, and users working in another area use **.doc** to refer to files in a binary file format used by a popular word processor. A useful typemap table in this situation might be:

```
Typemap:
        text   //depot/dev_projects/....doc
        binary //depot/corporate/annual_reports/....doc
```

To enable keyword expansion for all **.c** and **.h** files, but disable it for your **.txt** files, do the following:

```
Typemap:
        text+k //depot/dev_projects/main/src/....c
        text+k //depot/dev_projects/main/src/....h
        text   //depot/dev_projects/main/src/....txt
```

To ensure that files in a specific directory have their original file modification times preserved (regardless of submission date), use the following:

```
Typemap:
        binary   //depot/dev_projects/main/bin/...
        binary+m //depot/dev_projects/main/bin/thirdpartydll/...
```

All files at or below the **bin** directory are assigned type **binary**. Because later mappings override earlier mappings, files in the **bin/thirdpartydll** subdirectory are assigned type **binary+m** instead. For more information about the **+m** (modtime) file type modifier, see "File types" on page 707.

By default, Helix server supports concurrent development, but environments in which only one person is expected to have a file for edit at a time can implement pessimistic locking by using the **+l** (exclusive open) modifier as a partial filetype. If you use the following typemap, the **+l** modifier is automatically applied to all newly-added files in the depot:

```
Typemap:
        +l //depot/...
```

"Syntax" on page 19

Tip
Indent each line under `Typemap:`

Related Commands

To add a new file with a specific type, overriding the typemap table	`p4 add -t` ***`typefile`***
To change the filetype of an opened file, overriding any settings in the typemap table	**`p4 reopen-t`** ***`type file`***

p4 undo

Undo a range of revisions.

"Syntax" on page 19

```
p4 [g-opts] undo [-n] [-c change] [[FileSpec][revSpec]]
```

Description

The **`p4 undo`** command opens files to undo a set of previously submitted changes. The undone changes remain a part of the file history, but the new revisions submitted after **`p4 undo`** reverse their effect.

If a single revision is specified, the specified revision is undone. If a revision range is specified, the entire range is undone.

The workspace files opened by **`p4 undo`** are synced to the revision prior to those of the range and opened at the most recent **`undone`** revision.

Files that are opened at a revision prior to the head must be resolved prior to submission. To accomplish this, run `" p4 sync" on page 574` followed by `"p4 resolve" on page 458`.

The **`-n`** option previews the operation without changing any files.

If **`-c`** ***`change`*** is included, files are opened in the specified pending changelist instead of the default changelist.

See the "Example" on the next page below.

Options

`-n`	Preview the operation without changing any files.
`-c` ***`change`***	Open files in the specified pending changelist rather than the default changelist.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	**`open`**

To undo the effects of a particular change, issue this command:

```
$ p4 undo @change
```

To undo all changes made on April 1, 2018, issue this command:

```
$ p4 undo @2018/04/01,@2016/04/02
```

The above examples open files in the default changelist. To use a numbered changelist, specify it with the `-c` option.

Example

User Command	Result

The user wants the content of revision #1 to become the latest revision.
`p4 undo //depot2/dirE/readme.txt#2`
	An `undid` action against that revision #2 is pending:
	`//depot2/dirE/readme.txt#2 - opened for integrate`
	`... undid //depot2/dirE/readme.txt#2`

The user submits the file.
`p4 submit -d "undoing edit made in #2"`
	The content and history of revision #1 is now associated with the newly created revision #3 because an `integrate` action has completed.
	`Submitting change 635.`
	`Locking 1 files ...`
	`integrate //depot2/dirE/readme.txt#3`
	
	`Change 635 submitted.`

The user decides to review the history of the file.
`p4 filelog //depot2/dirE/readme.txt`
	The "p4 filelog" on page 199 command returns the history, indicating that revision #3 has `undone` revision #2.
	`//depot2/dirE/readme.txt`
	`... #3 change 635 integrate on 2018/10/26 by bruno@7ws (text) 'undoing edit made in #2'`
	`... ... undid //depot2/dirE/readme.txt#2`
	`... #2 change 634 edit on 2018/10/26 by bruno@7ws (text) 'making #2 in depot2/dirE '`
	`... ... undone by //depot2/dirE/readme.txt#3`
	`... #1 change 234 add on 2018/05/29 by bruno@7ws (text) 'added the initial file'`

Related Commands

Show integrations that have been submitted.	"p4 integrated" on page 271

p4 unload

Unloads a workspace, label, or task stream to the unload depot or to a flat file.

"Syntax" on page 19

```
p4 [g-opts] unload [-f -L -p -z] [-c client | -l label | -s
stream]
p4 [g-opts] unload [-f -L -z] [-a | -al | -ac] [-d date | -u
user]
```

Description

Two uses for the **p4 unload** command are:

To transfer infrequently-used metadata from Helix Core server db.* files to a set of flat files in the unload depot	To unload a client, label, or task stream to a flat file on the client
Helix server reporting commands often retrieve a superset of the desired data, and then users take advantage of automated or manual post-processing to discard the irrelevant lines of output. For example, the `p4 clients` command (when called without arguments) returns the name of every client workspace ever created by every current and former employee of your organization, even those who left years ago. Unloading metadata reduces the size of the working set required by Helix server. On large sites with many years of historical metadata, unloading can offer significant performance improvements, reduce the output of the command-line queries and the amount of information displayed in applications like P4V Helix server commands such as `p4 clients`, `p4 labels`, `p4 files`, `p4 sizes`, and `p4 fstat` ignore unloaded metadata. To view metadata that has been unloaded, use the `-U` option with these commands.	This is useful for seeding a client into another database, or for creating a private backup of the client. The flat file uses the standard journal format. The client, label, or task stream remains fully loaded after the command is run. Requires the `-o` Option.

"Syntax" on page 19

Use the `-c` and `-l` options to unload a specific client workspace or label. Users can only unload their own workspaces or labels. Administrators can use the `-f` option to unload workspaces and labels owned by other users.

You do not need to unload a workspace in preparation for moving it from one edge server to another because running the `p4 reload` command automatically unloads the specified workspace before reloading it into a new edge server.

Use the `-a`, `-al`, or `-ac` options to indicate that all specified labels and/or client workspaces are to be unloaded. You cannot use these options if you are also using the `-o` option.

Use the `-d` ***`date`*** and/or `-u` ***`user`*** to restrict the unloading operation to labels and/or workspaces older than a specific ***`date`***, owned by a specific ***`user`***, or both.

Use the `-L` option to unload locked workspaces and/or labels. By default, only unlocked labels or workspaces are unloaded.

The access date for a workspace is updated when:

- the workspace is used by a command that directly references the workspace. These commands include: "p4 add" on page 53, "p4 change" on page 83, "p4 delete" on page 143, "p4 diff" on page 162, "p4 edit" on page 180, "p4 have" on page 247, "p4 integrate" on page 265, "p4 labelsync" on page 319, "p4 lock" on page 341, "p4 move" on page 377, "p4 opened" on page 383, "p4 reconcile" on page 431, "p4 reopen" on page 447, "p4 resolve" on page 458, "p4 revert" on page 481, "p4 shelve" on page 517, "p4 submit" on page 558, " p4 sync" on page 574, "p4 unshelve" on page 607, and "p4 where" on page 631
- the workspace is used in a revision specifier of the form `@`***`workspace`***.

Note
Running "p4 client" on page 100 does NOT update the access time, only the client Update time if the client spec is edited and saved.

The access date for a label is updated when:

- the label is used by a command that directly references that label.
- the label is used in a revision specifier of the form `@`***`labelname`***.

By default, data in the unload depot is uncompressed. Use `-z` to store it in compressed form. Unloaded metadata is often highly compressible, particularly in continuous build environments characterized by millions of build-associated workspaces labels that are used to perform a single build and then rarely, if ever, accessed again.

Options

`-a`	Unload all applicable client workspaces and labels; requires `-d`, `-u`, or both `-d` and `-u` options. This option does not affect task streams.

`-ac`	Unload client workspaces; requires `-d`, `-u`, or both `-d` and `-u` options.
`-al`	Unload labels; requires `-d`, `-u`, or both `-d` and `-u` options.
`-c` *`client`*	Unload the specified client workspace's metadata from `db.have` (and related tables) and store it in the unload depot.
`-d` *`date`*	Unload metadata older than the specified date.
`-f`	Force option; administrators can unload workspaces, labels, and task streams owned by other users.
`-l` *`label`*	Unload the specified label from `db.label` (and related tables) and store it in the unload depot.
`-L`	Unload a `locked` workspace, label, or task stream.
`-p`	Promote any non-promoted shelves belonging to the specified client that is being unloaded. The shelf is promoted to the commit server where it can be accessed by other edge servers.
`-s` *`stream`*	Unload the specified task stream. Note that the *`stream`* must be of type `task`.
`-u` *`user`*	Unload metadata owned by the specified user.
`-z`	Store the unloaded workspace, label, or task stream in compressed format.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`write` `admin`

- To unload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough (in the `p4 group` form) that users do not need to ask for assistance with `p4 unload` or `p4 reload` operations.

Related Commands

To reload data from the unload depot.	`p4 reload`

p4 unlock

Release the lock on one or more files.

"Syntax" on page 19

```
p4 [g-opts] unlock [-c change | -s shelvedchange | -x] [-f] [file
...]
p4 [g-opts] -c client unlock [-f ] -r
```

Description

Files might be locked on a server from a failed submit from a client workspace. To unlock those files:

```
p4 [g-opts] unlock [-c change | -s shelvedchange | -x] [-f]
[file ...]
```

This releases locks that were created explicitly using the `p4 lock` command, or implicitly during the course of a submit operation.

If the file is open in a pending changelist other than `default`, use this command's `-c` option to specify the pending changelist.

If no changelist is specified, `p4 unlock` unlocks files in the default changelist.

If no file name is given, all files in the designated changelist are unlocked.

By default, files can be unlocked only by the changelist owner, who must also be the user who has the files locked. However, administrators on the commit server can use the `-fx` option. For example:

```
p4 unlock -fx readme.html
```

Note
Consider that when the administrator uses the `-fx` option, the administrator undoes a feature of the **`+l`file type** described at "File type modifiers" on page 709. The exclusive lock for editing of the **`+l`file type** is meant to prevent other users from being able to affect the file.

For an alternative approach to managing the files of an absent user, see the Support Knowledgebase article, "Reverting Another User's Files".

Commit-edge architecture: Files might be locked on a commit server from a failed `p4 submit` or a failed `p4 unlock` from an edge server.
To unlock those files on the commit server, either the user who issued the failing command unlocks them:
`p4 -c workspace-name unlock -r`
or an administrator forces unlocking with the `-f` option:
`p4 -c workspace-name unlock -f -r`
specifying the name of the workspace the files are locked in as the `-c` global flag to p4.

DVCS: Files might be locked on a remote sever from a failed `p4 push`.
To unlock those files on the remote server, either the user who issued the failed push command unlocks them:
`p4 -c workspace-name unlock -r`
or an administrator forces unlocking with the `-f` option:
`p4 -c workspace-name unlock -f -r`
specifying the name of the workspace the files are locked in as the `-c` global flag to p4.

Note
If **p4 unlock** is called from an Edge Server, any corresponding files locked globally via "p4 lock" on page 341 `-g` by that client will be unlocked on the Commit Server.

Options

`-c` ***`changelist`***	Unlock files in pending changelist ***`changelist`***. This option applies to opened files in a pending changelist that were locked by `p4 lock` or a failed submit operation of an unshelved changelist. **Tip** The **p4 unlock** `-c` option is different from the global `-c` option.
`-f`	Superuser or administrator force option that allows unlocking of files opened by other users.
`-r`	Commit/Edge: Unlock the files associated with the specified client that were locked on the commit server due to a failed `p4 submit` or a failed `p4 unlock` command from the edge server. DVCS: Unlock the files associated with the specified client that were locked on the remote server due to a failed `p4 push` command.

`-s` *`shelvedchange`*	If a file is locked in a pending shelved changelist, unlock it and keep it within the *`shelvedchange`*. This can typically only happen if a `p4 submit -e` command is aborted.
`-x`	In multi-server environments, unlock files that have the `+l` filetype (exclusive open) but have become orphaned (this is typically only necessary in the event of an extended network outage between an edge server and the commit server).
`g-opts`	See "Global options" on page 690, and in particular the `-p` and `-c` options.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`write`

Related Commands

To lock files so other users can't submit them	`p4 lock`
To display all your open, locked files (UNIX)	`p4 opened	grep "*locked*"`

p4 unlock (graph)

Release a locked file, leaving it open.

"Syntax" on page 19

```
p4 unlock [-c changelistNumber -f file ...]
```

Description

`p4 unlock` releases locks on the specified files.

If no file specification is given and no changelist is specified, all open files are unlocked.

The changelist flag and file specification limit the files to be unlocked:

- If a changelist is specified, only those files open in that changelist are unlocked.
- If a file specification is given, only those files are unlocked.

- If both changelist and file specification are provided, only the matching files in the specified changelist are unlocked.

Options

`-c` *`changelist`*	Applies to opened files in a pending changelist locked by `p4 lock` or by a failed submit.
`-f`	Enables you to unlock files in changelists owned by other users. This option requires `super` access, granted by "p4 protect" on page 401 or `admin` permissions on the affected repos. **Note** By default, files can be unlocked only by the changelist owner, who must also be the person who has the files locked.

Example

```
$ p4 edit ...
//repo/main/src/foo.c - opened for edit
//repo/main/src/bar.c - opened for edit
//repo/main/src/main.c - opened for edit
```

```
$ p4 lock ...
//repo/main/src/foo.c - locking
//repo/main/src/bar.c - locking
//repo/main/src/main.c - locking
```

```
$ p4 opened ...
//repo/main/src/foo.c#none - edit default change (text) *locked*
//repo/main/src/bar.c#none - edit default change (text) *locked*
//repo/main/src/main.c#none - edit default change (text) *locked*
```

```
$ p4 unlock ...
//repo/main/src/foo.c - unlocking
//repo/main/src/bar.c - unlocking
//repo/main/src/main.c - unlocking
```

```
$ p4 opened ...
//repo/main/src/foo.c#none - edit default change (text)
//repo/main/src/bar.c#none - edit default change (text)
//repo/main/src/main.c#none - edit default change (text)
```

Other users can open locked files for edit, but will not be able to submit them:

```
$ p4 -u user2 submit -d edit_of_locked_files
//repo/main/src/foo.c - already locked by bruno@prc7
//repo/main/src/bar.c - already locked by bruno@prc7
//repo/main/src/main.c - already locked by bruno@prc7
File(s) couldn't be locked.
Submit failed -- fix problems above then use 'p4 submit -c 919'.
```

p4 unshelve

Restore shelved files from a pending change into a workspace.

"Syntax" on page 19

```
p4 [g-opts] unshelve -s shelvedchange [-A [-f | -s] -n] [-c
change]
                            [-b branch | -S stream [-P stream]] [file
...]
```

Description

The **`p4 unshelve`** command retrieves shelved files from the specified pending changelist, opens them in a pending changelist, and copies them to the invoking user's workspace.

Unshelving files from a pending changelist is restricted by the user's permissions on the files. Access to shelved files from a pending changelist is controlled by the user's permissions on the files.

In addition to the files being unshelved, **`p4 unshelve`** also unshelves any open stream specification. For open stream specifications, see **`p4 stream`**.

You can limit the files to be unshelved by specifying a file pattern.

Unshelving copies the shelved files into the user's workspace as they existed when they were shelved. (For example, a file open for edit when shelved will also be open for edit in the unshelving user's workspace.)

You can unshelve a promoted shelf into open files and branches on a server from where the shelf did not originate.

By default, the unshelve command acts on both the files and the stream spec.

- To unshelve only files, use **`p4 unshelve -Af`**
- To unshelve only the stream spec, use **`p4 unshelve -As`**

See also the **`p4 help streamcmds`** command-line output.

Note
Unshelving a file over an already opened file is permitted if both shelved file and opened file are opened for **`edit`**. In a multi-server environment, the shelf must either be promoted or have been created on the same edge server. After unshelving, the workspace file is flagged as unresolved, and "p4 resolve" on page 458 must be run to resolve the differences between the shelved file and the workspace file.

Unshelving a file opened for `add` when the file already exists in the depot will result in the file being opened for edit. After unshelving, the workspace file is flagged as unresolved, and "p4 resolve" on page 458 must be run to resolve the differences between the shelved file and the depot file at the head revision.

Note
As a best practice, use the `-f` option to unshelve a shelf that has added files.

`p4 obliterate myfile` does not obliterate a shelve of the file (archive or metadata). When you attempt to unshelve a file that has been obliterated, you will get an error. To recover the content of that file, print the file. To get rid of the shelve, delete the shelf.

Options

`-A`	Used with `-f` to unshelve only files, or with `-s` to unshelve only the stream spec.
`-b` *`branch`*	Specifies a branch spec through which the shelved files will be mapped prior to unshelving. This option enables you to shelve files in one branch and unshelve them in another.
`-c` *`change`*	Specify a changelist number in the user's workspace into which the files are to be unshelved. By default, `p4 unshelve` retrieves files into the default changelist.
`-f`	Force the overwriting of writable (but unopened) files during the unshelve operation.
`-n`	Preview the results of the unshelve operation without actually restoring the files to your workspace.
`-P` *`stream`*	Unshelve to the specified parent stream. Overrides the parent defined in the source stream specification.
`-s` *`shelvedchange`*	Specify the pending changelist number that contains the originally-shelved files.
`-S` *`stream`*	Specifies the use of a stream-derived branch view to map the shelved files between the specified stream and its parent stream. See also the `-P` option.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`open`

Related Commands

To store files from a pending changelist into the depot without submitting them.	`p4 shelve`

p4 unsubmit

Unsubmit a changelist, making it a shelved set of changes.

You may not issue this command directly to an edge server, but you can issue it directly to a commit server.

> **Note**
> For Helix Server for Distributed Versioning (DVCS) only.

"Syntax" on page 19

```
p4 [g-opts] unsubmit [-n] [-r remote] [[FileSpec][revSpec]]
```

Description

The **`p4 unsubmit`** command takes one or more submitted changelists and undoes the submissions, leaving the changelist as a shelved change with the same content. The changelist can then be unshelved and further updated prior to resubmitting it.

The changelist must have been submitted by the same user and workspace which are used in the **`p4 unsubmit`** command. The files in the changelist must be the head revisions of those files in the server. The files must not have been integrated into any other files in the server. The files must not be open by any pending or shelved changelists. The files must not have been archived or purged. The files must not have associated attributes.

If the command specifies multiple files or multiple revisions, all the changelists which affected the specified revisions of the specified files are unsubmitted. Each such change becomes its own separate shelf. Fix records linked to the changelist are not modified.

After unsubmitting a change which has associated jobs, you should review the job and fix status for accuracy. The shelved changelists that are created do not fire any triggers of type **`shelve-submit`** or **`shelve-commit`**.

After all the specified changelists have been unsubmitted, the **`p4 unsubmit`** command syncs the workspace to the head revision.

Options

`-n`	Performs all the correctness checks, but does not unsubmit any files.

"Syntax" on page 19

`-r` ***`remotespec`***	Specifies a remote spec. The map in the remote spec is used to limit the files affected by the **`p4 unsubmit`** command. Thus a command such as **`p4 unsubmit -r rmt @>=17`** will affect only the files specified by the remote spec, not all files in the depot.
`FileSpec`	The files whose changes will be unsubmitted.
`revSpec`	If the file argument has a revision, the specified revision is unsubmitted. If the file argument has a revision range, the revisions in that range are unsubmitted.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	**`admin`**

Examples

`p4 unsubmit //depot/foo#head`	Unsubmits the last change to **`//depot/foo`**.

Related Commands

To resubmit unsubmitted changelists	**`p4 resubmit`**

p4 unzip

Import files from a **`p4 zip`** package file.

"Syntax" on page 19

```
p4 [g-opts] unzip -i zipfile [-Ox] [-f -n -A -I -v --enable-dvcs-
triggers]
```

Description

The infrequently used combination of "p4 zip" on page 635 and **p4 unzip** is one way an administrator can bring over a set of files (and their history) from the original server to the root directory of the target server. One use case is to transfer files over an "air gap" where there is no network connection or Perforce replication between the two Helix Core servers.

Tip
This process involves the manual step of the administrator copying onto the target server the file that contains the archive. See the Support Knowledgebase article, "How to move data from one Perforce server to another Perforce server using p4 zip and p4 unzip".

Note
Typically, an organization will instead use an automated solution for data (and metadata) across multiple servers. See the "Deployment architecture" chapter of the Helix Core Server Administrator Guide.

The **p4 unzip** operation imports:

- the specified set of files (such as source code and graphics)
- the changelists that submitted those files
- the files' attributes
- any fixes association with the changelists
- all integration records that describe integrations to the files being unzipped

The value of the **`rpl.checksum.change`** configurable determines the level of verification performed for the **`p4 unzip`** command. See "Configurables" on page 715.

Triggering when unzipping

The following push trigger types may be invoked during the execution of the **`p4 unzip`** command:

- The `push-submit` trigger can customize processing during the phase of the `p4 unzip` command when metadata has been transferred, but files have not yet been transferred.
- The `push-content` trigger can customize processing during that phase of the `p4 unzip` command when files have been transferred, but their contents have not yet been committed.
- The `push-commit` trigger can do any clean up work or other post processing after changes have been committed by the `p4 unzip` command.

To enable push triggers for the `p4 unzip` command, use the `--enable-dvcs-triggers` option.

For more information, see the section "Triggering on pushes and fetches" in *Helix Core Server Administrator Guide*.

Options

`-A`	Include the archive content of the new revisions.
`--enable-dvcs-triggers`	Enable any push triggers when the specified file is processed. Push triggers are disabled by default for the `p4 unzip` command. For more information about these kinds of triggers, see the chapter "Using triggers to customize behavior" in *Helix Core Server Administrator Guide*.
`-f`	Bypasses the correctness checks.
`-i` ***`zipfile`***	Specifies the zip file name.
`-I`	Excludes integration records for the new revisions.
`-n`	Performs all the correctness checks, but does not push any files or changelists to the remote server.
`-Oc`	When set, the `p4 unzip` command outputs information about every changelist. The `-v` option must be set for this to take effect.
`-Of`	When set, the `p4 unzip` command outputs information about every file in every changelist. The `-v` option must be set for this to take effect.
`-Oi`	When set, the `p4 unzip` command outputs information about every integration in every file in every changelist. The `-v` option must be set for this to take effect.

`-v`	Specifies verbose mode, which provides diagnostics for debugging. You must opt in to verbose mode. With verbose mode turned on, you can refine and control the precise level of verbosity. Specifically, you can indicate whether you want information about: ■ every changelist unzipped ■ every file in every changelist unzipped ■ every integration of every file in every changelist unzipped You can specify any combinations of these options, but must always include the `-O`. The default is to display information about every changelist.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`admin`

Related Commands

Package a set of files and their history for use by **`p4 unzip`**	`p4 zip`

p4 update

Update a client workspace without overwriting files that have changed since last sync.

"Syntax" on page 19

```
p4 [g-opts] update [-L -n -q]  [[FileSpec][revSpec]]
```

Description

p4 update is an alias for a **p4 sync -s**.

Options

-L	For scripting purposes, perform the update on a list of valid file arguments in full depot syntax with a valid revision number.
-n	Display the results of the update without actually performing the update. This lets you make sure that the update does what you think it will do before you do it.
-q	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	**read**

p4 update is an alias for **p4 sync -s**.

Related Commands

p4 update is an alias for **p4 sync -s**	**p4 sync** -s
To copy files from the depot to the client workspace	**p4 sync**

p4 user

Create, edit, or delete Helix server user specifications and preferences.

"Syntax" on page 19

```
p4 [g-opts] user [-f] [username]
p4 [g-opts] user -d [-f | -F] username
p4 [g-opts] user -D [-f | -y] username
p4 [g-opts] user -o [username]
p4 [g-opts] user -i [-f]
```

Description

Use the **`p4 user`** command to edit these settings or to create new user records. (By default, new users are created automatically. After installing Helix server, a Helix server superuser can control this behavior with the **`p4 configure`** command.)

Types of Helix server users:

standard users	■ standard users are the default ■ each standard user consumes one Helix server license
operator users	■ intended for system administrators ■ can run only a limited subset of Helix server commands ■ does not consume a license
service users	■ intended for inter-server communication in replicated and multi-server environments ■ restricted to an even smaller subset of Helix server commands than operator users ■ does not consume a license

When called without a ***`username`***, **`p4 user`** edits the specification of the current user. When called with a ***`username`***, the user specification is displayed, but cannot be changed. The form appears in the editor defined by the **`P4EDITOR`** environment variable.

Helix server superusers can create new users or edit existing users' specifications with the `-f` (force) option: **`p4 user -f`** ***`username`***.

Note that both the `-f` and `-F` options can be used to delete users, but the `-F` option has additional effects on protections and groups. See "Options" on page 619 below.

The user who gives a Helix server command is not necessarily the user under whose name the command runs. The user for any particular command is determined by the following:

- If the user running the command is a Helix server superuser, and uses the syntax **`p4 user -f`** ***`username`***, user ***`username`*** is edited.
- If the `-u` ***`username`*** option is used on the command line (for instance, **`p4 -u joe submit`**), the command runs as that user named "joe" (a password might be required).
- If the above has not been done, but the file pointed to by the `P4CONFIG` environment variable contains a setting for `P4USER`, the command runs as that user.
- If neither of the above has been done, but the `P4USER` environment variable has been set, the command runs as that user.
- If none of the above apply, then the username is taken from the OS level `USER` or `USERNAME` environment variable.

Note
The `-D` option can be convenient for the administrator if a user leaves the organization. This option not only deletes the specified user, it also deletes all the client workspaces that belong to the absent user.

See "Options" on page 619 and "Examples" on page 622.

Form Fields

Field Name	Type	Description
`User:`	Read-only	The Helix server username under which **`p4 user`** was invoked. By default, this is the user's system username. Be aware of the "Limitations on characters in filenames and entities" on page 699.
`Type:`	Read-only	Type of user: `standard`, `operator`, or `service`. **Important** The type cannot be changed after the user is created.

Field Name	Type	Description
`AuthMethod:`	Writable	One of the following: `perforce` or `ldap`. This field can only be changed when the `-f` option is specified for the `p4 user` command. ■ Specifying `perforce` enables authentication using Helix server's internal `db.user` table or by way of an authentication trigger. This is the default unless it is overridden with the `auth.default.method` configurable. ■ Specifying `ldap` enables authentication against AD/LDAP servers specified by the currently active LDAP configurations.
`Email:`	Writable	The user's email address. By default, this is ***`user`***`@`***`client`***.
`Update:`	Read-only	The date and time this specification was last updated.
`Access:`	Read-only	The date and time this user last ran a Helix server command.
`FullName:`	Writable	The user's full name.
`JobView:`	Writable	A description of the jobs to appear automatically on all new changelists (see "Usage Notes" on page 620).
`Password:`	Writable	The user's password (see "Usage Notes" on page 620).
`PasswordChange:`	Read-only	The date and time of the user's last password change. If the user has no password, this field is blank.
`Reviews:`	Writable List	A list of files the user would like to review (see "Usage Notes" on page 620). This field can include exclusionary mappings.

Tip
If there is a line under a field, indent that line. For example,

```
Reviews:
    //depot/path/to/directory1/...
//maria/depot/path/to/directory1/...
    //depot/path/to/directory2/...
//maria/depot/path/to/directory2/...
```

Options

`-d` ***`username`***	Deletes the specified user. Only user ***`username`***, or a Helix server superuser, can run this command. If you have set `P4AUTH`, no warning will be given if you delete a user who has an open file or client.
`-D` ***`username`***	Deletes: ▪ the specified user and removes the user from the protections table and from all groups ▪ all the client workspaces that belong to that user, thereby reverting files that the deleted user had open However: ▪ does not work on an edge server - see "Commit-edge" in *Helix Core Server Administrator Guide* ▪ does not delete any Helix Swarm client workspaces that are associated with the deleted user, so that Swarm's history remains intact **Note** Does NOT delete workspace clients that have files opened by OTHER users. To force the deletion of workspace clients that have files opened by OTHER users, combine the `-D` option with the `-f` option in preview mode. To perform the operation, add the `-y` option. See "Examples" on page 622.
`-y`	Used with `-D` to actually perform the delete operation. Without `-y`, `p4 user -D`, `p4 user -D -f`, and `p4 user -D -F` merely report what the command would do IF `-y` were included. **Note** `p4 user -D -y`, `p4 user -D -f -y`, and `p4 user -D -F -y` cannot be undone.
`-f`	Superuser force option that allows the superuser to create, modify, or delete the specified user, or to change the last modified date. (preview mode unless the `-y` option is added)

`-F`	Superuser option that is used with the `-d` or `-D` option. With the `-D` option, `-F` forces the deletion of the specified user and also removes the user from the protections table and from all groups. The command fails if removing the user from a group causes the group to be deleted. In such a case, delete the group before deleting the user. (preview mode unless the `-y` option is added)
`-i`	Read the user specification from standard input. The input must conform to the `p4 user` form's format.
`-o`	Write the user specification to standard output.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- The `-d` option can be used by non-superusers only to delete the user specification that invoked the `p4 user` command. Helix server superusers can delete any Helix server user.
- User deletion fails if the specified user has any open files. Submit or revert these files before deleting users.
- By default, user records are created without passwords, and any Helix server user can impersonate another by setting `P4USER` or by using the globally available `-u` option. To prevent another user from impersonating you, set a password with the `p4 passwd` command.

  Passwords can be created, edited, or changed in the `p4 user` form or by using the `p4 passwd` command. Setting your password in the `p4 user` form is only supported at security levels 0 or 1. You can `p4 passwd` to set passwords at any server security level, and you *must* use `p4 passwd` to set passwords at higher security levels. For more about how the various security levels work, see the *Helix Core Server Administrator Guide*.

  Tip
  If you edit the `Password:` field in the `p4 user` form, do not use the comment character `#` within the password. Helix server interprets everything following that character on the same line as a comment, and does not store it as part of the password.

  If the `dm.user.resetpassword` configurable has been set, all users created with passwords are required to reset their passwords before they can issue commands.
- Passwords are displayed as six asterisks in the `p4 user` form regardless of their length.
- If you are using ticket-based authentication (see `p4 login` for details), changing your password automatically invalidates all of your outstanding tickets.

- The collected values of the `Email:` fields can be listed for each user with the `p4 users` command, and can used for any purpose.
- The `p4 reviews` command, which is used by the Helix server change review daemon, uses the values in the `Reviews:` field; when activated, it will send email to users whenever files they've subscribed to in the `Reviews:` field have changed. Files listed in this field must be specified in depot syntax.
  For example, if user `joe` has a `Reviews:` field value of

  ```
  //depot/main/...
  //depot/.../README
  ```

  the change review daemon sends `joe` email whenever any `README` file has been submitted, and whenever any file under `//depot/main` has been submitted.

  Another example for the `Reviews:` field is:

  `//depot/*/relnotes.txt`

  to send notification for changes to the `relnotes.txt` file on *all* the branches in the depot.
- There is a special setting for job review when used with the Helix server change review daemon. If you include the value:

  ```
  //depot/jobs
  ```

  in your `Reviews:` field, you will receive email when jobs are changed.
- If you set the `Jobview:` field to any valid jobview, jobs matching the jobview appear on any changelists created by this user. Jobs that are fixed by the changelist should be left in the changelist when it's submitted with `p4 submit`; other jobs should be deleted from the form before submission.

  For example, suppose the jobs at your site have a field called `Owned-By:`. If you set the `Jobview:` field on your `p4 user` form to `Owned-By=`*`yourname`*`&status=open`, all open jobs owned by you appear on all changelists you create. See `p4 jobs` for a full description of jobview usage and syntax.
- Operators are intended for system administrators who, even though they have super or admin privileges, are responsible for the maintenance of the Perforce service, rather than the development of software or other assets versioned by the service. Operators can run only the following commands:

`p4 admin stop`	`p4 admin restart`	`p4 admin checkpoint`
`p4 admin journal`	`p4 counter`	`p4 counters`
`p4 dbstat`	`p4 dbverify`	`p4 diskspace`

`p4 configure`	`p4 counter` (including `-f`)	`p4 counters`
`p4 journaldbchecksums`	`p4 jobs` (including `-R`)	`p4 login`
`p4 logout`	`p4 logappend`	`p4 logparse`
`p4 logrotate`	`p4 logschema`	`p4 logstat`
`p4 logtail`	`p4 lockstat`	`p4 monitor`
`p4 passwd`	`p4 ping`	`p4 verify`
`p4 user`		

- Service users are used in replication environments, and can run only the following commands:

`p4 dbschema`	`p4 export`	`p4 login`
`p4 logout`	`p4 passwd`	`p4 info`
`p4 user`		

Examples

`p4 user joe`	View the user specification of Helix server user `joe`.
`p4 user`	Edit the user specification for the current Helix server user.
`p4 user -d sammy`	Delete the user specification for the Helix server user `sammy`, but has no effect on `sammy`'s workspace clients.
`p4 user -D sammy`	**Previews** the deletion of: • user `sammy` • all of `sammy`'s workspace clients, **except** those where a user other than `sammy` has files opened

"Syntax" on page 19

`p4 user -D -y sammy`	**Performs** the deletion of: • user `sammy` • all of `sammy`'s workspace clients, **except** those where a user other than `sammy` has files opened **Note** Cannot be undone.
`p4 user -D -f sammy`	**Previews** the deletion of: • user `sammy` • all of `sammy`'s workspace clients, **including** those where a user other than `sammy` has files opened
`p4 user -D -f -y sammy`	**Performs** the deletion of: • user `sammy` • all of `sammy`'s workspace clients, **including** those where a user other than `sammy` has files opened **Note** Cannot be undone.
`p4 user -D -F -y sammy`	**Performs** the deletion of: • user `sammy` • all of `sammy`'s workspace clients, **including** those where a user other than `sammy` has files opened • `sammy` from the protections table and groups **Note** Cannot be undone.

`p4 -u joe -P hey submit`	Run `p4 submit` as user `joe`, whose password is **`hey`**. This command does not work at higher security levels.
`p4 user -f joe2`	Create a new Helix server user named `joe2` if the caller is a Helix server superuser, and `joe2` does not already exist as a Helix server user. If user `joe2` already exists, allow a Helix server superuser to modify the user's settings.

Related Commands

To view a list of all Helix server users	**`p4 users`**
To change a user's password	**`p4 passwd`**
To view a list of users who have subscribed to review particular files	**`p4 reviews`**
To control how new users are created by changing the `dm.user.noautocreate` configurable	**`p4 configure`**

p4 users

Print a list of all known users of the current Perforce service.

"Syntax" on page 19

```
p4 [g-opts] users [-l -a -r -c] [-m max] [user ...]
```

Description

`p4 users` displays a list of all the users known to the current Perforce service. For each user, the information displayed includes their Helix server user name, their email address, their real name, and the date and time the user last accessed the service.

If a ***`user`*** argument is provided, only information pertaining to that user is displayed. The ***`user`*** argument can contain the `*` wildcard; in this case, all users matching the given pattern are reported on. (If you use a wildcard, be sure to quote the user argument, because the OS will likely attempt to expand the wildcard to match file names in the current directory).

Use the `-m` ***`max`*** option to limit the output to the first ***`max`*** users.

Options

`-a`	Include service users in list.
`-c`	On replica servers, only user information from the master server are reported.
`-l`	Login information: includes time of last password change and login ticket expiry, if applicable. You must be a Helix server superuser to use this option.
`-m` ***`max`***	List only the first ***`max`*** users.
`-r`	On replica servers, only users who have used this replica server are reported.
`g-opts`	See "Global options" on page 690.

"Syntax" on page 19

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list`

- You must be connected to a replica to use `-c` or `-r`, and the `-c` and `-r` options are mutually exclusive.
- If the "run.users.authorize" on page 802 configurable has been set to `1`, users must authenticate themselves to the Perforce service before running the **`p4 users`** command.

Related Commands

To add or edit information about a particular user	`p4 user`
To edit information about the current client workspace	`p4 client`

p4 verify

Verify that the Helix server archives (depot files) are intact (complete and without corruption).

"Syntax" on page 19

```
p4 [g-opts] verify [-u | -v | -z] [-m max -q -s -X -b batchsize]
FileSpec[revRange] ...
p4 [g-opts] verify -t [-U | -A ][-z -m max -q -s -X -b batchsize]
FileSpec[revRange] ...
p4 [g-opts] verify -S [-t -m max -q -X -b batchsize] FileSpec...
p4 [g-opts] verify -U [-u | -v | -z] [-m max -q -s -X -b
batchsize] unloadFileSpec ...
p4 [g-opts] verify -A [-u | -v | -z] [-m max -q -s -X -b
batchsize] archiveFileSpec ...
p4 verify -Z [ -q ] file ...
```

Description

For each revision of the specified depot files, **`p4 verify`** reports the revision-specific information and an MD5 digest (fingerprint) of the revision's contents.

The verification process involves opening the depot file (sometimes called the depot archive file), reading it, and verifying that its digest matches the expected value.

- If the file can't be opened, **`p4 verify`** reports **`MISSING`**.
- If the file can be opened, but does not have the expected contents, **`p4 verify`** reports **`BAD`**.

If invoked without arguments, **`p4 verify`** computes and displays the MD5 digest of each revision.

Syntax variants offer the following choices:

- Verify a given set of file revisions.
- Verify a given set of shelved file revisions.
- Verify file revisions in the unload depot.
- Verify file revisions in the archive depot. (To learn about archive depots for infrequently-accessed files, see "Reclaiming disk space by archiving files" in Helix Core Server Administrator Guide.)

We recommend that you regularly verify the integrity of your depot files. See backup-recovery-ensuring-integrity.html in *Helix Core Server Administrator Guide*.

Verifying shelved files

The verification of shelved files lets you know whether your shelved archives have been lost or damaged.

"Syntax" on page 19

If a shelf is local to a specific edge server, you must issue the **`p4 verify -S`** command on the edge server where the shelf was created. If the shelf was promoted, run the **`p4 verify -S`** on the commit server.

You may also run the **`p4 verify -S -t`** command on a replica to request re-transfer of a shelved archive that is missing or bad. Re-transferring a shelved archive from the master only works for shelved archives that are present on the master, that is, a shelf that was originally created on the master or promoted from an edge server.

Verifying archived files

The verification of archived files lets you know whether your archived files have been damaged. It is a good practice to run a command like the following before you restore files with the **`p4 restore`** command.

```
$ p4 verify -A //archive/depot/mysource/...
```

Note
If **`p4 verify`** returns errors, contact Perforce Technical Support.

Options

`-A archivefiles`	It is possible for files in the archive depot to become corrupted over time. This option allows you to verify these files before you restore them with the **`p4 restore`** command. **`archivefiles`** specifies a file, a list of files, or set of files to verify. For example: `//archive/depot/..../source`
`-b batchsize`	By default, **`p4 verify`** processes files in batches of 10,000 files at a time. You can change this batch size with the **`-b batchsize`** option, where *batchsize* represents an integer. To disable batching, specify **`-b 0`**. If the **`-z`** option is specified, the **`-b`** option is ignored and all options are processed in a single batch.

"Syntax" on page 19

`-m` ***`max`***	Limit **`p4 verify`** to ***`max`*** number of revisions. Use this option with the **`-u`** option to compute and save digests for a limited number of revisions in each invocation of the **`p4 verify`** command. **Tip** Depending on the ***`FileSpec`***, **`p4 verify -m 10`** ***`FileSpec`*** might list 10 files, 10 versions of a single file, or a combination of files and file versions.
`-q`	Run quietly and report only errors from mismatched digests or unreproducible revisions.
`-s`	Verify file size as well as digest. The **`-v`** implies the **`-s`** option.
`-S`	Verify shelved files. If you specify this option, the only valid revision specifier is **`@=change`**, specifying a single shelf. Use file patterns to restrict verification to a specific set of shelved files. In a multi-server installation, this command should be run on the edge server where the shelf was created. If the shelf has been promoted, this command may also be run on the commit server.
`-t`	For use on replicas only: **`p4 verify -t`** causes the replica to schedule a transfer of the contents of any damaged or missing revisions. (This option also works on a replica with **`lbr.replication=cache`**.) The **`-t`** option cannot be used with the **`-v`** or **`-u`** options. In replicated environments, **`p4 verify -t`** reports **`BAD!`** or **`MISSING!`** files with **`(transfer scheduled)`** at the end of the line.
`-u`	Store the filesize and MD5 digest of each file in the Helix server database if no filesize and/or digest has been previously stored. Subsequent uses of **`p4 verify`** will compare the computed version against this stored version.
`-U` ***`unloadfile`***	Verify files in the unload depot. See **`p4 unload`** for details.
`-v`	Store the MD5 digest of each file in the Helix server database, even if there's already a digest stored for that file, overwriting the existing digest. (The **`-v`** option is used only to update the saved digests of archive files that have been deliberately altered outside of Helix server control by a Helix server system administrator.) The **`-v`** and **`-u`** options are mutually exclusive.
`-X`	Skip files of filetype **`+X`** (for which the service runs an **`archive`** trigger.)

`-z`	Optimizes performance by skipping revisions that have already been computed in the current pass. This option speeds verifications for files that exist via lazy copies. The resulting output might report a lazy copy revision if it is the first revision in the sort order to access a common archive file. This option cannot be used with the `-v` or `-u` options.
`-Z`	Specifies that the digest and file length check use the storage table as opposed to the revision table. This is generally faster because the lazy copied files are only checked one time. The output displays the reference count instead of the action and change fields. The *file* argument is different because a lazy copied depot file is not be selectable. For example, consider file `//x/x` with a lazy copy at the path `//y/y`. Calling `p4 verify -Z //y/...` does not verify any files because there is no storage record that matches the `//y` path. However, without the `-Z` option, `p4 verify //y/...` uses the revision table to find and check the `//x/x` file. The `-Z` option can be combined solely with the `-q` option.
`g-opts`	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	`admin`

Examples

`p4 verify -S //depot/main/p4/...`	Verifies all shelved files matching the specified path.
`p4 verify -q -S //...@=1023548`	Verifies the shelved files in shelf 1023548 only.
`p4 verify -q //...#head,#head`	Verifies only the head revision of all files in the depot, reporting only on files with problems.

Related Commands

To get information about how you can specify revisions.	`p4 help revisions`

p4 where

Show where a particular file is located, as determined by the client view.

"Syntax" on page 19

```
p4 [g-opts] where [file ...]
```

Description

p4 where uses the client view and root (as set in **p4 client**) to print files' locations relative to the top of the depot, relative to the top of the client workspace, and relative to the top of the local OS directory tree. The command does not check to see if the file exists; it merely reports where the file *would be* located if it *did* exist.

For each file provided as a parameter, a set of mappings is output. Each set of mappings is composed of lines consisting of three parts: the first part is the filename expressed in depot syntax, the second part is the filename expressed in client syntax, and the third is the local OS path of the file.

Options

g-opts	See "Global options" on page 690.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`

- The mappings are derived from the client view: a simple view, mapping the depot to one directory in the client workspace, produces one line of output.

  More complex client views produce multiple lines of output, possibly including exclusionary mappings. For instance, given the client view:

  ```
  View: //a/... //client/a/...
        //a/b/... //client/b/...
  ```

  Running:

"Syntax" on page 19

```
$ p4 where //a/b/file.txt
```

gives:

```
-//a/b/file.txt //client/a/b/file.txt //home/_user_/root/a/b/file.txt
//a/b/file.txt //client/b/file.txt /home/_user_/root/b/file.txt
```

This can be interpreted as saying that the first line of the client view would have caused the file to appear in `/home/user/root/a/b/file.txt`, except that it was overridden by the second mapping in the view. An exclusionary mapping was applied to perform the override, and the second mapping applies, sending the file to `/home/user/root/b/file.txt`.

- The simplest case (one line of output per file, showing each filename in depot, client, and local syntax) is by far the most common.

Examples

`p4 where file.c`	Show depot, client workspace, and local filesystem locations of `file.c` (or where `file.c` would appear if it existed in the depot.)
`p4 where 100%25.txt`	Use ASCII expansion of "`%`" character to locations for file `100%.txt`. ASCII expansion is supported for the following four special characters: `@` (`%40`), `#` (`%23`), `*` (`%2A`), and `%` (`%25`).

Related Commands

To list the revisions of files as synced from the depot	`p4 have`

p4 workspace

Create or edit a client workspace specification and its view.

Syntax

```
p4 [g-opts] workspace [-f] [-t template] [workspace]
p4 [g-opts] workspace -d [-f [-Fs]] workspace
p4 [g-opts] workspace -o [-t template] [workspace]
p4 [g-opts] workspace -S stream [[-c change] -o] [workspace]
p4 [g-opts] workspace -s [-f] -S stream [workspace]
p4 [g-opts] workspace -s [-f] -t template [workspace]
p4 [g-opts] workspace -i [-f]
```

Description

The command `p4 workspace` is an alias for `p4 client`.

Warning
A branch, depot, label, and workspace may not share the same name.

p4 workspaces

List all client workspaces currently known to the system.

Syntax

```
p4 [g-opts] workspaces [-t] [-u user] [[-e|-E] filter] [-m max]
[-S stream]  [-a | -s serverID]
```

```
p4 [g-opts] workspaces -U
```

Description

The command **`p4 workspaces`** is an alias for **`p4 clients`**.

p4 zip

(DVCS) Package a set of files and their history for use by **`p4 unzip`**.

"Syntax" on page 19

```
p4 zip -o file [-r remote -A -I] [filespec | -c change]
```

```
p4 zip -o file [-r remote -A -I] -s shelf
```

Description

Writes the following to the specified zip file:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes association with the changelists
- all integration records that describe integrations to the files being zipped

The content of the zip file can be specified either by providing a filespec, which selects a set of revisions, or by providing one or more changelist numbers using the **`-c`** option, which selects all the revisions modified by those changelists.

The second form of the command writes a single shelved changelist to the specified zip file. When you use the **`-s`** option, you must also use the **`-A`** option.

The zip appears either in the server installation directory ("P4ROOT" on page 680) or the path that you specify in the ***`file`*** option.

Tip
For a walkthrough, see the Support Knowledgebase article, "How to move data from one Perforce server to another Perforce server using p4 zip and p4 unzip".

Options

`-A`	Include the archive content of the new revisions.
`-c` ***`change`***	Zips up just the files covered by the specified changelist.

"Syntax" on page 19

`-I`	Excludes integration records for the new revisions.
`-o` *`filename`*	Specifies the zip file name.
`-r` *`remotespec`*	Specifies the remote spec to be used to re-map the files when they are written to the zip file.
`-s`	Specifies a shelved changelist to be zipped, instead of one or more submitted changelists. For more information, see the section "Fetch and push a shelved changelist" in *Using Helix Core Server for Distributed Versioning*.
`filespec`	Specifies the filepath of the files to zip up.

Examples

`p4 zip -o foo //...`	Creates a zip file named `foo` with all changes and revisions.

Related Commands

Import files from a **`p4 zip`** package file	`p4 unzip`

Environment and registry variables

Each operating system and shell has its own syntax for setting environment variables. The following table shows how to set the `P4CLIENT` environment variable on various systems:

OS or Shell	Environment Variable Example
UNIX: `bash`, `ksh`, `sh`,	`P4CLIENT=`*`value`*` ; export P4CLIENT`
UNIX: `csh`	`setenv P4CLIENT `*`value`*
Mac OS X (`bash`)	`P4CLIENT=`*`value`*` ; export P4CLIENT`
Windows	`p4 set P4CLIENT=`*`value`* Note how Windows administrators running Helix server as a service set the value of a variable: For more details on setting Helix server variables in Windows and OS X, see the `p4 set` command.

For the specified service	For all users on the local machine

`p4 set -S `*`servicename`*` `*`var`*`=`*`value`*
For example,
`p4 set -S Perforce "P4DEBUG" on page 652 =""net.keepalive.idle" on page 774=2700"`
where Perforce is the service name
	`p4 set -s `*`var`*`=`*`value`*
	For example,
	`p4 set "P4PORT" on page 677 =ssl:myhost:1667`
	where users connect on port 1667

Note
You may use `$home` to set environment variables. For example:

```
P4IGNORE=$home/myignorefile
```

`$home` is expanded to the path of the user's home directory. The user's home directory is taken to be the value of the `HOME` environment variable or of `USERPROFILE` on Windows.

Helix server's environment variables can be grouped into the following four categories:

Type	Description	Name

crucial	must be set on the client and default values are rarely sufficient	`P4CLIENT`
		`P4PORT`
		`P4PASSWD`
		`P4USER`

useful	can provide additional functionality to the user	`P4CONFIG`
		`P4DIFF`
		`P4EDITOR`
		`P4MERGE`
		`P4CHARSET`
		`P4TRUST`

server	set by the administrator for users or applications	`P4AUDIT`
		`P4JOURNAL`
		`P4LOG`
		`P4PORT`
		`P4ROOT`
		`P4DEBUG`
		`P4NAME`
		`P4SSLDIR`

esoteric	default value is sufficient	`P4PAGER`
		`PWD`
		`TMP`, `TEMP`
		`P4TICKETS`
		`P4LANGUAGE`
		`P4LOGINSSO`
		`P4COMMANDCHARSET`
		`P4DIFFUNICODE`
		`P4MERGEUNICODE`
		`P4CLIENTPATH`

Note
Although these environment variables are not listed in the "Configurables" on page 715 chapter, these environment variables are also "configurables" in the sense that they can be set by using the "p4 configure" on page 122 command. The two exceptions are **`P4JOURNAL`** and **`P4ROOT`**.

P4ALIASES

The location of the directory that holds your **`.p4aliases`** file.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

If **`P4ALIASES`** is not defined, the system looks in the **`$HOME`** directory on Unix and Mac systems, and in your **`$USERPROFILE`** directory on Windows.

For more information on custom command aliases, see "Command aliases" on page 34.

Examples

```
/usr/bin
/customize
```

P4AUDIT

Location of the audit log file.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	**`p4d -A`** ***`auditlog`***	N/A

Value if not Explicitly Set

Operating System	Value
All	None. If no log file is specified, auditing is disabled.

Notes

`P4AUDIT` specifies the location of the audit log file.

When auditing is enabled, Helix server adds a line to the audit log file every time file content is transferred from the shared versioning service to any user. On an active installation, the audit log file will grow very quickly.

Lines in the audit log appear in the form:

```
datetimeuser@clientclientIPcommandfile#rev
```

For example:

```
2011/05/09 09:52:45 karl@nail 192.168.0.12 diff //depot/src/x.c#1
2011/05/09 09:54:13 jim@stone 127.0.0.1 sync //depot/inc/file.h#1
```

If a command is run on the same physical machine that hosts the Perforce service, the ***`clientIP`*** is shown as **`127.0.0.1`**.

For commands that arrive through a Helix Proxy, the IP address is reported in the form ***`proxyIP`*****`/`*****`clientIP`***, and the command is reported as ***`command`*****`-proxy`**.

In order to ensure that user activity on replica and edge servers (specifically in environments involving build farm replicas, forwarding replicas, and/or edge servers) is tracked, each replica or edge server must have **`P4AUDIT`** configured.

See Auditing user file access in the *Helix Core Server Administrator Guide*.

P4AUTH

A hostname and port number of an optional Helix Core centralized authorization server.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	N/A	N/A

Value if not Explicitly Set

Program	Value
Perforce Servers	`null`

Examples

Helix server server examples
`perforce.example.com:1818`
`192.168.0.123:1818`

Notes

The format of **`P4AUTH`** is ***`host:port`***, or ***`port`*** by itself if both the Helix Core server and the authorization server are running on the same host. All servers must be at the same release level.

Port numbers must be in the range `1024` through `32767`.

Warning
If you have set **`P4AUTH`**, no warning will be given if you delete a user who has an open file or client.

See Centralized authorization server (P4AUTH) in the *Helix Core Server Administrator Guide*.

Tip
We recommend that all the servers run the same operating system. See the Knowledgebase article on "CR/LF Issues and Text Line-endings" between Windows and non-Windows operating systems.

P4BROKEROPTIONS

Set Helix Broker options for a Windows service.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	**`p4broker %P4BROKEROPTIONS%`**	N/A

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

For example, if you normally run the Broker with the command:

```
p4broker -c c:\p4broker\broker.conf
```

you can set the **`P4BROKEROPTIONS`** variable for the Windows service to run with:

```
p4 set -S "Broker" P4BROKEROPTIONS= "-c c:\p4broker\broker.conf"
```

When you run P4Broker under the **`"Broker"`** service, the Broker will configure itself using the specified **`broker.conf`** file. Use **`P4BROKEROPTIONS`** when you need to call **`p4broker`** with options for which there are no corresponding environment variables, or when you are doing so within the context of a Windows service.

See Helix Broker in the *Helix Core Server Administrator Guide*.

P4CHANGE

The hostname and port number of the optional centralized changelist server from which this Helix server is being configured to get the most recent unused changelist number.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	N/A	N/A

Value if not Explicitly Set

Operating System	Value
All	None. If the Perforce service is operating in unicode mode and **`P4CHARSET`** is unset, Helix server applications return an error message. See Centralized changelist server (P4CHANGE) in *Helix Core Server Administrator Guide*.

Examples

Helix server server examples
`perforce.example.com:1818`
`192.168.0.123:1818`

Notes

The format of `P4CHANGE` is ***`host:port`***, or *port* by itself if both the Helix server and the changelist server are running on the same host. All servers must be at the same release level.

Port numbers must be in the range `1024` through `32767`.

See Centralized changelist server (P4CHANGE) in the *Helix Core Server Administrator Guide*.

P4CHARSET

Character set used for translation of unicode files.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -C` ***`charsetcmd`***	Yes

Value if not Explicitly Set

Operating System	Value
All	None. If the Perforce service is operating in unicode mode and **`P4CHARSET`** is unset, Helix server applications return an error message. See Centralized changelist server (P4CHANGE) in *Helix Core Server Administrator Guide*.

Notes

If the server is set to Unicode-mode, the client sets **`P4CHARSET`** to **`auto`** and examines the client's environment to determine the character set to use in converting files of type **`unicode`**. Thus, the only time you need to set **`P4CHARSET`** to a specific type is if the client's choice of charset results in a faulty conversion or if you have other special needs. For example, the application that uses the checked out files expects a specific character set.

`P4CHARSET` only affects files of type **`unicode`** and **`utf16`** and non-unicode files are never translated.

For Perforce services operating in the default (non-Unicode mode), **`P4CHARSET`** must be left unset (or set to **`none`**) on user workstations. If **`P4CHARSET`** is set, but the service is not operating in internationalized mode, the service returns the following error message:

```
Unicode clients require a unicode enabled server.
```

For Perforce services operating in Unicode mode, **`P4CHARSET`** must either be set to **`auto`** or be set to some value (other than **`none`**) on user machines. If **`P4CHARSET`** is unset, but the service is operating in Unicode mode, Helix server applications return the following error message:

```
Unicode server permits only unicode enabled clients.
```

For more about Unicode mode, including settings of `P4CHARSET` for various UTF-8, UTF-16, and UTF-32 character sets, with and without byte-order marks, see the *Internationalization Notes*.

In addition to affecting the client, Unicode settings also affect trigger scripts that communicate with the server. You should check your trigger's use of file names, Helix server identifiers, and files containing unicode characters, and make sure that these are consistent with the character set used by the server.

For a complete list of valid `P4CHARSET` values, issue the **`p4 help charset`** command.

P4_port_CHARSET

Specifies whether the server is in Unicode mode.

"Syntax" on page 19

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`-C` option	Yes

Value if not Explicitly Set

Operating System	Value
All	None, which specifies that the server is not in Unicode mode.

Notes

When a client connects to the server, it attempts to discover the server's Unicode mode setting, and it sets the **`P4_port_CHARSET`** variable to specify that setting: for non-Unicode, the variable is set to **`none`**; for Unicode, the variable is set to **`auto`**. If **`P4_port_CHARSET`** is set to **`auto`**, the client sets the **`P4CHARSET`** to **`auto`**. The client then examines its own environment to determine what character set it needs to use in appropriately rendering unicode files from the server.

The *port* part of this environment variable specifies the *host:port* of the server to which the client is connected.

For more information about using servers in Unicode mode, see "Setting up and managing Unicode installations" in the *Helix Core Server Administrator Guide*.

P4CLIENT

Name of current client workspace.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -c` *`clientnamecmd`*	Yes

Value if not Explicitly Set

Operating System	Value
Windows	Value of `COMPUTERNAME` environment variable
All others	Name of host machine

Examples

```
"C:\Users\Joe Coder"
/usr/team/joe/workspace/buildfarm/joe
```

P4CLIENTPATH

A list of directories to which Helix server applications are permitted to write.

Any attempt by a Helix server application to access or modify files outside these areas of the filesystem will result in an error message.

To specify more than one directory, separate the directories with semicolons.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	N/A	Yes

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
"C:\Users\Joe Coder"
/usr/team/joe/workspace/buildfarm/joe
```

P4COMMANDCHARSET

Used to support UTF-16 and UTF-32 character sets from the Command-line Client.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -Q` ***`commandcharsetcmd`***	Yes

Value if not Explicitly Set

Operating System	Value
All	None

Notes

If you have set **`P4CHARSET`** to a UTF-16 or UTF-32 value, you must set **`P4COMMANDCHARSET`** to a non-UTF-16 or -32 value in order to use the **`p4`** Command-line Client. For details, see the *Internationalization Notes*.

For a complete list of valid **`P4COMMANDCHARSET`** values, issue the command **`p4 help charset`**.

P4CONFIG

Contains a file name without a path. The specified file is used to store other Helix server environment variables.

The current working directory (returned by **PWD**) and its parents are searched for the file. If a file is found, the variable settings within the file are used. If additional files are found in parent directories, and they contain variable settings not already found in other files, those variable settings are also used.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	N/A

Value if not Explicitly Set

Operating System	Value
All	If not set, this variable is not used.

Examples

A sample **P4CONFIG** file might contain the following lines:

```
P4CLIENT=joes_client
P4USER=joe
P4PORT=ssl:ida:3548
```

Notes

P4CONFIG makes it trivial to switch Helix server settings when switching between different projects. If you place a configuration file in each of your client workspaces and set **P4CONFIG** to point to that file, your Helix server settings will change to the settings in the configuration files automatically as you move from directories in one workspace to another.

Common settings may be placed into a configuration file in a parent directory. These settings may be overridden by setting them in configuration files in child directories.

The file defined by **P4ENVIRO** contains the same kind of information as the **P4CONFIG** file. However,

- the **P4CONFIG** variable contains only the file name of a configuration file, and the system searches through successive parent directories
- the **P4ENVIRO** variable contains the exact location and name of a configuration file if it is not at its default location

Each line in the configuration file defines one variable. The definition takes the form ***variable=value***.

You can use both **P4CONFIG** and **P4ENVIRO** files to define environment variables: use the **P4CONFIG** file for those variables that have different values for different workspaces and the **P4ENVIRO** file for those variables that remain constant for all projects. Values set in a **P4CONFIG** file override those set in a **P4ENVIRO** file.

Common variables to set within a **P4CONFIG** file include the following:

- **P4CLIENT**
- **P4DIFF**
- **P4EDITOR**
- **P4HOST**
- **P4LANGUAGE**
- **P4MERGE**
- **P4PASSWD**
- **P4PORT**
- **P4TICKETS**
- **P4USER**

P4DEBUG

Set Helix server or proxy trace options.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	None	No

Value if not Explicitly Set

Operating System	Value
All	As of Release 2011.1, `server=1` is the default setting.

Examples

See the "Helix Server Trace Flags" article.

Notes

To disable logging, set `P4DEBUG` to `server=0`.

Higher settings for the Helix server trace options are useful only to administrators working with Perforce Technical Support to diagnose or investigate a problem. The proxy does not set debugging by default.

The preferred way to set trace options for the Helix server (or proxy) is to set them on the `p4d` (or `p4p`) command line. For technical reasons, this does not work for sites running Helix server or proxies as services under Windows. Administrators at such sites can use `p4 set` to set the trace options within `P4DEBUG`, allowing the service to run with the options enabled.

Setting server debug levels on a Helix server (`p4d`) has no effect on the debug level of a Helix Proxy (`p4p`) process, nor on downstream replicas or edge servers.

P4DESCRIPTION

An optional description for a Helix server.

`P4DESCRIPTION` is used by `p4 server` as a means of identifying servers.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4d -Id` ***`description`***	N/A

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
"Commit server"
"Replica server"
"Build farm"
```

P4DIFF

The name and location of the diff program used by `p4 resolve` and `p4 diff`.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
Windows	If the environment variable **`DIFF`** has been set, then the value of **`DIFF`**; otherwise, if the environment variable **`SHELL`** has been set to *any* value, then the program diff is used; otherwise, **`p4diff.exe`**.
All Others	If the environment variable **`DIFF`** has been set, then the value of **`DIFF`**; otherwise, Helix server's internal diff routine is used.

Examples

```
diff
diff -b windiff.exe
```

Notes

The value of `P4DIFF` can contain options to the called program, for example, **`diff -u`**.

The commands `p4 describe`, `p4 diff2`, and `p4 submit` all use a diff program built into **`p4d`**. This cannot be changed.

P4DIFFUNICODE

Used to support UTF-16 and UTF-32 character sets from the Command-line Client.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
Windows	None

Notes

This environment variable is used in place of "P4DIFF" on page 654 if the file being diffed is of type unicode or utf16, and the character set is passed as the first argument to the command.

P4EDITOR

The editor invoked by Helix server commands that use forms.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
Linux, UNIX, OS X	If `EDITOR` is set to any value, then the value of `EDITOR`, otherwise, `vi`.
Windows	If `P4EDITOR` is set to any value, then the value of `P4EDITOR`, otherwise, `Notepad`.

Examples

On Linux, UNIX, OS X:

```
export EDITOR=/usr/bin/vi
```

On Windows:

```
p4 set P4EDITOR="C:\Program Files\TextPad 8\TextPad.exe"
```

Notes

The regular Helix server commands that use forms (and therefore, use this variable), are `p4 branch`, `p4 change`, `p4 client`, `p4 job`, `p4 label`, `p4 submit`, and `p4 user`.

The superuser commands that use forms are `p4 depot`, `p4 group`, `p4 jobspec`, `p4 protect`, `p4 triggers`, and `p4 typemap`.

P4ENVIRO

Contains the non-default path and name of a configuration file that stores Helix server environment variables.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`none`	No

Value if not Explicitly Set

Operating System	Value
Windows, Mac OS X	None
POSIX/Unix	`$HOME/.p4enviro`

Notes

The file specified by `P4ENVIRO` contains the same kind of information as the file specified by `P4CONFIG`. The difference is that the `P4CONFIG` variable contains just the file name of a configuration file for which the system searches through successive parent directories; the `P4ENVIRO` variable contains the exact location of a configuration file if it is not at its default location. For Windows and Mac OS X platforms, the `P4ENVIRO` variable must be explicitly set if you have values stored in a configuration file you mean to use across projects.

Each line in the `P4ENVIRO` file is used to define one variable; the definition takes the form *`variable=value`*.

You can use both `P4ENVIRO` and `P4CONFIG` files to define environment variables: use the `P4CONFIG` file for those variables that have different values for different workspaces and the `P4ENVIRO` file for those variables that remain constant for all projects. Values set in a `P4CONFIG` file override those set in a `P4ENVIRO` file.

Note
Setting `P4ENVIRO` on Windows will cause `p4 set` to store values in the specified environment file rather than in the Windows registry.

Examples

A sample **`P4ENVIRO`** file might contain the following line:

```
P4_myServer:1667_CHARSET=auto
```

P4HOST

Name of host computer to impersonate.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -H hostname command`	Yes

Value if not Explicitly Set

Operating System	Value
All	The value of the client hostname as returned by `p4 info`.

Examples

```
workstation123.perforce.com
```

Notes

Helix server users can use the `Host:` field of the `p4 client` form to specify that a particular client workspace can be used only from a particular host machine. When this field has been set, the `P4HOST` variable can be used to fool the service into thinking that the user is on the specified host machine regardless of the machine being used by the user. As this is a very esoteric need, there's usually no reason to set this variable.

The hostname must be provided exactly as it appears in the output of `p4 info` when run from that host.

P4IGNORE

Specify a list of files that contain lists of rules for ignoring files when adding files to the depot and reconciling workspaces.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
All	none

Examples

```
.p4ignore
.p4ignore;$home/.myp4ignore
```

Notes

Note
You may use **$home** to set environment variables. For example

```
P4IGNORE=$home/myignorefile
```

The syntax for the contents of a **P4IGNORE** file is *not* the same as Helix server syntax. Instead, it is similar to that used by other versioning systems:

- Rules are specified using local filepath syntax. Unix style paths will work on Windows for cross platform file support.
- A # character at the beginning of a line denotes a comment.

  You can escape the # comment character with a backslash (\). This allows filenames beginning with # to be ignored.

- A `!` character at the beginning of a line line excludes the file specification. These exclusions override rules defined above it in the **`P4IGNORE`** file, but may be overridden by later rules.
- A `/` (or `\` on Windows) character at the beginning of a line causes the file specification to be considered relative to the **`P4IGNORE`** file. This is useful when the rule must apply to files at particular depots of the directory tree.
- A `/` (or `\` on Windows) character at the end of a line causes the file specification to only match directories, and not files of the same name.
- The `*` wildcard matches substrings. Like the Helix server wildcard equivalent, it does not match path separators; however, if it is not used as part of a path, the directory scanning nature of the rule may make it appear to perform like the Perforce "`...`" wildcard.
- The `**` wildcard matches substrings including path separators. It is equivalent to the Helix server "`...`" wildcard, which is not permitted.

For example:

```
# Ignore .p4ignore files
.p4ignore

# Ignore object files, shared libraries, executables
*.dll
*.so
*.exe
*.o

# Ignore all HTML files except the readme file
*.html
!readme.html

# Ignore the bin directory
bin/

# Ignore the build.properties file in this directory
/build.properties

# Ignore all text files in test directories
test/**.txt
```

The first match is used when figuring out what to ignore.

Use the `p4 ignores` command to get information about why a file is being ignored during add and reconcile operations. For example, the following command, lets you know which line of the `P4IGNORE` file is being used to ignore a file.

```
$ p4 ignores -v -i mypath
```

P4JOURNAL

A file that holds the Helix server database’s journal data.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4d -J` ***`file`***	N/A

Value if not Explicitly Set

Operating System	Value
All	`P4ROOT/journal`

Examples

```
journal
off
/disk2/perforce/journal
```

Notes

If a relative path is provided, it should be specified relative to the Helix server root.

Setting `P4JOURNAL` to `off` disables journaling. This is not recommended.

For further information, see Setting protections with p4 protect, Journal files and Helix Core server (p4d) Reference in the the *Helix Core Server Administrator Guide*.

P4LANGUAGE

This environment variable is reserved for system integrators. See also *Internationalization Notes*.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -L` ***`language cmd`***	Yes

Value if not Explicitly Set

Operating System	Value
All	N/A

P4LOG

Name and path of the file to which Helix server errors are written.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?

No	Yes	`p4d -L` *`file`*
		`p4p -L` *`file`*
			N/A

Value if not Explicitly Set

Operating System	Value
All	Standard error

Examples

```
log
/disk2/perforce/log
```

Notes

If a relative path is provided, it is specified relative to the Helix server root.

See Logging commands in the *Helix Core Server Administrator Guide*.

P4LOGINSSO

Client-side single sign-on (SSO) script.

Triggers of type `auth-check-sso` fire when standard users run the `p4 login` command.

Two scripts are run: a **client-side script** is run on the user's workstation, and its output is passed (in plaintext) to the Helix server, where the **server-side script** runs.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	N/A	N/A

Value if not Explicitly Set

Operating System	Value
All	N/A

Examples

```
/Users/joe/bin/runsso
```

Notes

If this value is not set on a server that is configured to use SSO-based login, the user will either have to use password authentication or the user will be denied the ability to log in. See Single signon and auth-check-sso triggers in the the *Helix Core Server Administrator Guide*.

P4MERGE

A third-party merge program to be used by `p4 resolve`'s merge option.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
All	If the `MERGE` environment variable is set, then its value; otherwise, nothing.

Examples

```
c:\Perforce\p4merge.exe
c:\progra~1\Perforce\p4merge.exe
```

Notes

The program represented by the program name stored in this variable is used only by `p4 resolve`'s merge option. When `p4 resolve` calls this program, it passes four arguments, representing (in order) *base*, *theirs*, and *yours*, with the fourth argument holding the resulting *merge* file.

If the program you use takes its arguments in a different order, set `P4MERGE` to a shell script or batch file that reorders the arguments and calls the proper merge program with the arguments in the correct order.

If you are running under Windows, you must call a batch file, even if your third-party merge program already accepts arguments in the order provided by Helix server. This is due to a limitation within Windows. For instance, if you want to use a program called `MERGE.EXE` under Windows, your batch file might look something like this:

```
SET base=%1
SET theirs=%2
SET yours=%3
SET merge=%4
C:\FULL\PATH\TO\MERGE.EXE %base% %theirs% %yours% %merge%
```

P4MERGEUNICODE

Used to support UTF-16 and UTF-32 character sets from the Command-line Client.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
All	None

Notes

This environment variable is used in place of `P4MERGE` if the file being resolved is of type `unicode` or `utf16`, and the character set is passed as the first argument to the command.

P4NAME

Important
We recommend that you use "p4 serverid" on page 505 instead of `P4NAME`.

Important
To avoid configuration problems, the value of `serverID` should always match the value of P4NAME if both are set. We recommend setting `serverID`, but support `P4NAME` for backward compatibility.

A unique identifiable name for a Helix server. This configurable cannot be set globally. You must specify a server id.

`P4NAME` is used by `p4 configure` as a means of identifying servers. Unless a `P4NAME` value has been specified for the server, the server uses the `serverid` to determine the appropriate configuration settings. See `p4 serverid`.

Warning
On Windows if there is no `P4NAME` defined in the registry for a service, it is picked up from the name of the service itself.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4d -In` ***name***	N/A

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
masterserver
failoverserver
buildserver
```

P4PAGER

The program used to page output from `p4 resolve`'s diff option.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	No

Value if not Explicitly Set

Operating System	Value
All	If the variable `PAGER` is set, then the value of `PAGER`; otherwise, none.

Examples

`/bin/more` (UNIX)

Notes

The value of this variable is used *only* to display the output for `p4 resolve`'s diff routine. If the variable is not set, the output is not paged.

P4PASSWD

Supplies the current Helix server user's password for any Helix server command.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -P passwd command`	Yes

Value if not Explicitly Set

Operating System	Value
All	None

Notes

Helix server passwords are set via `p4 passwd`, or in the form invoked by `p4 user`. The setting of `P4PASSWD` is used to verify the user's identity. If a password has not been set, the value `P4PASSWD` is not used, even if set.

While it is possible to manually set the `P4PASSWD` environment variable to your plaintext password, the more secure way is to use the `p4 passwd` command. On UNIX, this will invoke a challenge/response mechanism which securely verifies your password. On Windows, this sets `P4PASSWD` to the encrypted MD5 hash of your password.

On Windows platforms, if you set a password in P4V, the value of the registry variable `P4PASSWD` is set for you. Setting the password in P4V is like using `p4 passwd` (or `p4 set P4PASSWD`) from the MS-DOS command line, setting the registry variable to the encrypted MD5 hash of the password. The unencrypted password itself is never stored in the registry.

If you are using ticket-based authentication, but have a script that relies on a `P4PASSWD` setting, use `p4 login` **-p** to display the value of a ticket that can be passed to Helix server commands as though it were a password (that is, either from the command line, or by setting `P4PASSWD` to the value of the valid ticket).

P4PCACHE

For the Helix Proxy, the directory in which the proxy stores its files and subdirectories.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4p -r directory`	N/A

Value if not Explicitly Set

Operating System	Value
All	`p4p`'s directory. Windows administrators running the Helix Proxy process as a service should use `p4 set -S svcname P4PCACHE=directory` to set the value of `P4PCACHE` for the named service.

Notes

Create this directory before starting the Helix Proxy (**p4p**).

Only the account running **p4p** needs to have read/write permissions in this directory.

See Helix Proxy in the *Helix Core Server Administrator Guide*.

P4PFSIZE

For the Helix Proxy, the size (in bytes) of the smallest file to be cached. All files larger than `P4PFSIZE` bytes in length are cached.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4p -e` ***`size`***	N/A

Value if not Explicitly Set

Operating System	Value
All	0; that is, cache all files

Notes

See Helix Proxy in the *Helix Core Server Administrator Guide*.

P4POPTIONS

Set Helix Proxy options for a Windows service.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4p %P4POPTIONS%`	N/A

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

For example, if you normally run the Proxy with the command

```
$ p4p -p 1999 -t mainserver:1666
```

you can set the `P4POPTIONS` variable for the Windows `proxysvc` to run with

```
$ p4 set -S "Perforce Proxy" P4POPTIONS="-p 1999 -t
mainserver:1666"
```

When you run P4P under the `"Helix Proxy"` service, the Proxy will listen to port 1999 and communicate with the Perforce service at `mainserver:1666`.

Most installations do not need to use `P4POPTIONS`, because there are already environment variables associated with most `p4p` options; in the example shown above, you can use `P4PORT` and `P4TARGET`. Use `P4POPTIONS` when you need to call `p4p` with options for which there are no corresponding environment variables, and when you are doing so within the context of a Windows service.

See Helix Proxy in the *Helix Core Server Administrator Guide*.

P4PORT

For the Perforce service (server, broker, or proxy), the port number on which it listens, and the network transport(s) to which it is to bind.

For Helix server applications, the protocol, host and port number of the Perforce service with which to communicate. The most commonly-used communications protocols are `tcp` (plaintext over TCP/IP) or `ssl` (SSL over TCP/IP).

Helix server supports connectivity over IPv6 networks as well as over IPv4 networks. You can specify whether you require (or prefer) to use IPv4 or IPv6 addresses when resolving hostnames. The protocol settings of `tcp4` and `ssl4` require IPv4 address support. Similarly, `tcp6` and `ssl6` require IPv6 support. Using `tcp64` and `ssl64` attempts first to resolve the host to an IPv6 address, but will accept an IPv4 address if IPv6 is not available. The opposite behavior is available with `tcp46` and `ssl46`; these default to the use of IPv4 if possible, and use IPv6 if IPv4 is unavailable. A configurable, `net.rfc3484`, may be set on user workstations or in `P4CONFIG` files in order to permit the operating system to automatically determine which transport to use.

Behavior and performance of networked services depend on:

- the networking capabilities of the machine that hosts the service
- the operating systems used by the end users
- your specific LAN and WAN infrastructure (and the state of IPv6 support for every router between the end user and the Helix Core server).

Suppose a user is working from home with an IPv6-based home network, but the ISP or VPN provider does not fully support IPv6. Variations of P4PORT provide flexibility and backwards compatibility for administrators and users during the transition from IPv4 to IPv6:

P4PORT protocol value	Behavior in IPv4/IPv6 or mixed networks
`<not set>`	Use `tcp4:` behavior, but if the address is numeric and contains two or more colons, assume `tcp6:` If the `net.rfc3484` configurable is set, allow the OS to determine which transport is used.
`tcp:`	Use `tcp4:` behavior, but if the address is numeric and contains two or more colons, assume `tcp6:` If the `net.rfc3484` configurable is set, allow the OS to determine which transport is used.
`tcp4:`	Listen on/connect to an IPv4 address/port only.
`tcp6:`	Listen on/connect to an IPv6 address/port only.
`tcp46:`	Attempt to listen/connect to an IPv4 address. If this fails, try IPv6.

P4PORT protocol value	Behavior in IPv4/IPv6 or mixed networks
`tcp64:`	Attempt to listen/connect to an IPv6 address. If this fails, try IPv4.
`ssl:`	Use `ssl4:` behavior, but if the address is numeric and contains two or more colons, assume `ssl6:` If the `net.rfc3484` configurable is set, allow the OS to determine which transport is used.
`ssl4:`	Listen on/connect to an IPv4 address/port only, using SSL encryption.
`ssl6:`	Listen on/connect to an IPv6 address/port only, using SSL encryption.
`ssl46:`	Listen on/connect to an IPv4 address/port. If that fails, try IPv6. After connecting, require SSL encryption.
`ssl64:`	Listen on/connect to an IPv6 address/port. If that fails, try IPv4. After connecting, require SSL encryption.

Tip
In mixed environments, it is good practice to set the net.rfc3484 configurable to 1:

```
$ p4 configure set net.rfc3484=1
```

Doing so ensures RFC3484-compliant behavior for users who do not explicitly specify the protocol value. In other words, if the client-side configurable `net.rfc3484` is set to `1`, and `P4PORT` is set to `example.com:1666`, or `tcp:example.com:1666`, or `ssl:example.com:1666`, the user's operating system determines whether to use IPv4 or IPv6 transport for a given connection.

If you use SSL to connect to Helix server, the fingerprint of the Helix server must match that stored in the `P4TRUST` file. (When you connect to a new Helix server installation for the first time, the server's fingerprint is displayed. If it matches the one your administrator has assigned it, you may safely connect to the server by using the `p4 trust` command to add the server to your `P4TRUST` file.)

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	`p4 -p` *`protocol`* `:`*`host`*`:`*`portcmd`*	Yes

Value if not Explicitly Set

Program	Value
Helix server	`1666`
Helix Proxy	`1666`
Helix server application	`perforce:1666`

Examples

Helix server application	Service
`1818`	`1818`
`ssl:squid:1234`	`ssl:1234`
`example.com:1234`	`1234`
`ssl:192.168.0.123:1818`	`ssl:1818`
`tcp6:[2001:db8::123]:1818`	`tcp6:[::]:1818`
`tcp6:example.com:1818`	`tcp6:[::]:1818`
`ssl64:[2001:db8::123]:1818`	`ssl6:[::]:1818ssl64:[::]:1818`

Notes

The format of `P4PORT` for Helix server applications is ***`protocol`*** `:` ***`host`*** `:` ***`port`***, or ***`port`*** by itself if both the Helix server application and versioning service are running on the same host. Port numbers must be in the range `1024` through `32767`.

If you specify both an IP address *and* a port number in `P4PORT`, the versioning service ignores requests from any IP addresses other than the one specified in `P4PORT`.

If you do not specify a protocol, transmissions between the Helix server applications and the versioning service are performed in plaintext, and IPv4 addresses are assumed.

P4ROOT

Directory in which the Perforce service stores its versioned files and its database files.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4d -r` ***`directory`***	N/A

Value if not Explicitly Set

Operating System	Value
All	**`p4d`**'s directory. Windows administrators running the Helix server back-end process as a service should use `p4 set` **`-S`** ***`svcname`*** **`P4ROOT=`*****`directory`*** to set the value of **`P4ROOT`** for the named service.

Notes

Create this directory before starting the versioning service (**`p4d`**).

Only the account running **`p4d`** needs to have read/write permissions in this directory.

For more information on setting up a Helix server installation, see Installing the server in the *Helix Core Server Administrator Guide*.

P4SSLDIR

Directory containing a server's SSL keys and/or certificates.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	None	No

Value if not Explicitly Set

Operating System	Value
All	If `P4SSLDIR` is set to any value, the directory named by `P4SSLDIR` stores the files that contain server ssl credentials. If this variable is unset, or if the permissions of `P4SSLDIR` or its contents are incorrect, the service will not start in SSL mode.

Examples

`/path/to/dir`

Notes

All Helix server processes (**`p4d`**, **`p4p`**, **`p4broker`**) that accept SSL connections require a certificate and key pair (stored in this directory) on startup. In order for any of these processes to start, the following additional conditions must be met:

- `P4SSLDIR` must be set to a valid directory.
- The `P4SSLDIR` directory must be owned by the same userid as the one running the Helix server, proxy, or broker process. The `P4SSLDIR` directory must not be readable by any other user. On UNIX, for example, the directory's permissions must be set to 0700 (**`drwx------`**) or 0500 (**`dr-x------`**).
- Two files, named **`privatekey.txt`** and **`certificate.txt`**, must exist in `P4SSLDIR`.

These files correspond to the PEM-encoded unencrypted private key and certificate used for the SSL connection. They must be owned by the userid that runs the Helix server, proxy, and broker process, and must also have their permissions set such as to make them unreadable by other users. On UNIX, for example, the files' permissions must be set to 0600 (`-rw-------`) or 0400 (`-r--------`).

You can supply your own private key and certificate, or you can use **`p4d -Gc`** to generate a key and certificate pair. For more information, see Key and certificate generation in the *Helix Core Server Administrator Guide*.

- To generate a fingerprint from your server’s private key and certificate, run **`p4d -Gf`**. (**`P4SSLDIR`** must be configured with the correct file names and permissions, and the current date must be valid for the certificate.)

  After you have communicated this fingerprint to your end users, your end users can then compare the fingerprint the server offers with the fingerprint you have provided. If the two fingerprints match, users can use **`p4 trust`** to add the fingerprint to their **`P4TRUST`** files.

P4TARGET

- For the Helix Proxy and replica servers, the name and port number of the target Helix server (that is, the Helix server for which P4P acts as a proxy).
- For a replica or edge server, the upstream master or commit server from which it retrieves its data, and towards which changelists, if applicable, are forwarded.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?

No	Yes	`p4p -t host:port`
		`p4d -t host:port`
			N/A

Value if not Explicitly Set

Program	Value
Helix Proxy	`perforce:1666`
Replicated environments	None

Examples

Helix server server examples
`1818`
`master:11111`
`perforce.example.com:1234`
`192.168.0.123:1818`

Notes

The format of `P4TARGET` is ***host:port***, or ***port*** by itself if both the Helix server and the proxy, replica, or edge server are running on the same host.

Port numbers must be in the range `1024` through `32767`.

For more about replicas, edge servers, and the Helix Proxy, see Deployment architecture in the *Helix Core Server Administrator Guide*.

P4TICKETS

The location of the ticket file used by `p4 login`.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	N/A	Yes

Value if not Explicitly Set

Program	Value
Windows	`%USERPROFILE%\p4tickets.txt`
All others	`$HOME/.p4tickets`

Examples

```
/staff/username/p4tickets.txt
```

Notes

The `P4TICKETS` environment variable must point to the actual ticket file, not merely a directory in which `p4tickets.txt` or `.p4tickets` is expected to exist. If you set `P4TICKETS` to point to a directory, you will not be able to log in.

P4TRUST

Specifies the path to the SSL trust file. The trust file contains the fingerprints of the keys used for SSL connections; it is controlled by the `p4 trust` command.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	N/A	Yes

The `P4TRUST` configurable is used by any service acting as a client to any SSL-enabled server. This includes client applications, proxies, brokers, replica servers, edge servers, and so on. How each of these uses the configurable is covered in the chapters describing the service.

Value if not Explicitly Set

Program	Value
Windows	`%USERPROFILE%\p4trust.txt`
All others	`$HOME/.p4trust`

Notes

Your system administrator can help you confirm the accuracy of any fingerprint (or change to a fingerprint) provided to you by a Helix server.

P4USER

Current Helix server username.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -u` ***`username command`***	Yes

Value if not Explicitly Set

Operating System	Value
Windows	The value of the `USERNAME` environment variable.
All Others	The value of the `USER` environment variable.

Examples

```
edk
lisag
```

Notes

By default, the Helix server username is the same as the OS username.

If a particular Helix server user does not have a password set, then any other Helix server user can impersonate this user by using the `-u` option with their Helix server commands. To prevent this, users should set their password with the `p4 user` or `p4 passwd` command.

If a user has set their Helix server password, you can still run commands as that user (if you know the password) with `p4 -u` ***`username`*** `-P` ***`passwordcommand`***.

Helix server superusers can impersonate users without knowing their passwords. For more information, see Protections and passwords in the *Helix Core Server Administrator Guide*.

PWD

The directory used to resolve relative filename arguments to Helix server commands.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`p4 -d directory command`	No

Value if not Explicitly Set

Operating System	Value
UNIX	The value of `PWD` as set by the shell; if not set by the shell, `getcwd()` is used.
All Others	The actual current working directory.

Notes

Sometimes the `PWD` variable is not inherited properly across shells. For instance, if you're running **`ksh`** or **`sh`** on top of **`csh`**, `PWD` will be inherited from your **`csh`** environment but not updated properly, causing possible confusion in subsequent Helix server commands.

If you encounter such difficulties, check to be sure you've unset `PWD` in your `.profile` or `.kshrc` file. (If you're running **`sh`** or **`ksh`** as your login shell, `PWD` will be managed properly by the shell regardless of any environment variables unset in your startup files; the confusion only occurs when variables are exported to sub-shells.)

TMP, TEMP

The directory to which Helix server applications and services write temporary files.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	None	No

Value if not Explicitly Set

Operating System	Value
UNIX	`/tmp`
All Others	For Helix server applications: the current working directory. On Helix server: `P4ROOT`

Notes

If `TEMP` is set, `TEMP` is used. Otherwise, if `TMP` is set, this is used. If neither `TEMP` nor `TMP` are set, temporary files are written in the directories described in the table above.

When using triggers on Windows, `%formfile%` and other variables that use a temp directory should use the `TMP` and `TEMP` system variables in Windows, *not* the user's `TEMP` variables.

Global options

Global options for Helix server commands can be supplied on the command line before any Helix server command.

"Syntax" on page 19

```
p4 [-b batchsize -c client -d dir -H host -p port -P pass -u user -x
file -C charset
    -Q charset -L language] [-I] [-G] [-Mj] [-s] [-z tag] cmd [args ...]
```

```
p4 -V
```

```
p4 -h
```

Options

`-b batchsize`	Specifies a batch size (number of arguments) to use when processing a command from a file with the `-x argfile` option. By default, the batch size is `128`.
`-c client`	Overrides any `P4CLIENT` setting with the specified client name.
`-d dir`	Overrides any `PWD` setting (current working directory) and replaces it with the specified directory.
`-I`	Specify that progress indicators, if available, are desired. This option is not compatible with the `-s` and `-G` options.
`-i`	Although not global, the `-i` and `-o` options work with forms and represent standard in and standard out.
`-G`	Causes all output (and batch input for form commands with `-i`) to be formatted as marshaled Python dictionary objects. This is most often used when scripting. **Note** Use the `-G` option with the `-z tag` option. Otherwise the marshaled output might be invalid. See also the "Usage Notes" on page 692.

`-Mj`	Formats output as line-delimited JSON objects, with non-UTF8 characters replaced with `U+FFFD` **Note** Use the `-Mj` option with the `-z tag` option. Otherwise the marshaled output might be invalid.
`-H` *`host`*	Overrides any `P4HOST` setting and replaces it with the specified hostname.
`-o`	Although not global, the `-i` and `-o` options work with forms and represent standard in and standard out.
`-p` *`port`*	Overrides any `P4PORT` setting with the specified *`protocol:host:port`*.
`-P` *`pass`*	Overrides any `P4PASSWD` setting with the specified password.
`-r` *`retries`*	Specifies the number of times to retry a command (notably, `p4 sync`) if the network times out.
`-s`	Prepends a descriptive field (for example, `text:`, `info:`, `error:`, `exit:`) to each line of output produced by a Helix server command. This is most often used when scripting.
`-u` *`user`*	Overrides any `P4USER`, `USER`, or `USERNAME` setting with the specified user name.
`-x` *`argfile`*	Instructs Helix server to read arguments, one per line, from the specified file. If *argfile* is a single hyphen (`-`), instructs Helix server to read from standard input instead of from a file.
`-C` *`charset`*	Overrides any `P4CHARSET` setting with the specified character set.
`-Q` *`charset`*	Overrides any `P4COMMANDCHARSET` setting with the specified character set.
`-L` *`language`*	This feature is reserved for system integrators.
`-z tag`	Causes output of many reporting commands to be in the same tagged format as that generated by `p4 fstat`. **Note** Use this option for all marshaled output, such as that of `-G` and `-Mj`. Otherwise the marshaled output might be invalid.

`-q`	Quiet mode, which suppresses informational messages and reports only warnings or errors.
`-V`	Displays the version of the `p4` application and exits.
`-h`	Displays basic usage information and exits.

Usage Notes

- Be aware that the global options must be specified on the command line before the Helix server command. Options specified after the Helix server command will not be interpreted as global options, but as options for the command being invoked. It is therefore possible to have the same command line option appearing twice in the same command, being interpreted differently each time.

  For example, the command **`p4 -c`** ***`anotherclient`*** **`edit -c 140 file.c`** will open file `file.c` for edit in pending changelist 140 under client workspace ***`anotherclient`***.

- The `-x` option is useful for automating tedious tasks because it processes a sequence of arguments, line by line, from the specified file. For example, suppose you want an easy way to bring many files into the depot. Create one file that lists many files, each on a separate line. Let's use the UNIX `cat` command to verify we have such a file:

  ```
  cat filesToAdd.txt
  partOne.txt
  partTwo.txt
  partThree.txt
  ```

  The `-x` option enables the "p4 add" on page 53 command to process each line of the file as if we had issued a series of "p4 add" on page 53 commands, each time with a different file argument. In this case,

  ```
  p4 -x filesToAdd.txt add
  ```

  is the equivalent of:

  ```
  p4 add partOne.txt
  ```

  ```
  p4 add partTwo.txt
  ```

  ```
  p4 add partThree.txt
  ```

  Therefore, the result of `p4 -x filesToAdd.txt` can be:

  ```
  //depot/repo/partOne.txt#1 - opened for add
  //depot/repo/partTwo.txt#1 - opened for add
  //depot/repo/partThree.txt#1 - opened for add
  ```

  The `-x` option can be as powerful as whatever generates its input. For example, a UNIX developer who wants to edit any file referring to a `file.h` file can issue the command:

```
grep -l file.h *.c | cut -f1 -d: | p4 -x - edit
```

where:

- the **`grep`** command lists occurrences of **`file.h`** in the **`*.c`** files
- the **`-l`** option tells **`grep`** to list each file only once
- the **`cut`** command splits off the filename from **`grep`**'s output before passing it to the **`p4 -x - edit`** command

- The **`-s`** option can be useful in automated scripts.

  For example, a script could be written as part of an in-house build process which executes **`p4 -s`** commands, discards any output lines beginning with "**`info:`**", and alerts the user if any output lines begin with "**`error:`**".

- Python developers find the **`-G`** option useful for scripting. For instance, to get a dictionary of all the fields of a job whose ID is known, use the following:

  ```
  job_dict = marshal.load(os.popen('p4 -G job -o ' + job_id, 'rb'))
  ```

  In some cases, it might not be obvious which keys are used by the application. If you pipe the output of any **`p4 -G`** invocation to the following script, you will see every record printed out in key/value pairs:

  ```
  #!/usr/local/bin/python

  import marshal, sys

  try:
      num=0
      while 1:
          num=num+1
          print '\n' % num
          dict = marshal.load(sys.stdin)
          for key in dict.keys(): print "%s: %s" % (key,dict[key])

  except EOFError: pass
  ```

  Python developers on Windows should be aware of potential CR/LF translation issues. In the example above, it is necessary to call **`marshal.load()`** to read the data in binary ("**`rb`**") mode.

Tip
For additional examples and guidance about scripting with this option, see the Support Knowledgebase article, "Using p4 -G".

- The progress indicator requested when you use the `-I` option is only available with `p4 -I submit` and `p4 -I sync -q`
- `p4 help` is simpler than `p4 -c workspace help` and provides the same output.

Examples

`p4 -p new_service:1234 sync`	Performs a sync after connecting to `new_service` and port `1234`, regardless of the settings of the `P4PORT` environment variable.
`p4 -c new_client submit -c 100`	The first `-c` is the global option to specify the client workspace name. The second `-c` specifies a changelist number.
`p4 -s -x filelist.txt edit`	If `filelist.txt` contains a list of files, this command opens each file on the list for editing, and produces output suitable for parsing by scripts. Any errors as a result of the automated `p4 edit` commands (for example, a file in `filelist.txt` not being found) can be detected by examining the command's output for lines beginning with "`error:`"

File specifications

Any file can be specified within any Helix server command in client syntax, depot syntax, or local syntax. Workspace names and depot names share the same namespace. The Perforce service can always distinguish a workspace name from a depot name.

Syntax forms

Local syntax refers to filenames as specified by the local shell or operating system. Filenames referred to in local syntax can be specified by their absolute paths or relative to the current working directory. (Relative path components can only appear at the beginning of a file specifier.)

Helix server has its own method of file specification which remains unchanged across operating systems. If a file is specified relative to a client root, it is said to be in *client syntax*. If it is specified relative to the top of the depot, it is said to be in *depot syntax*. A file specified in either manner can be said to have been specified in Helix server syntax.

Helix server file specifiers always begin with two slashes (`//`), followed by the client or depot name, followed by the full pathname of the file relative to the client or depot root directory.

Path components in client and depot syntax are always separated by slashes (`/`), regardless of the component separator used by the local operating system or shell.

An example of each syntax:

Syntax	Example
Local syntax	`/staff/bruno/myworkspace/file.c` for Linux `c:\staff\bruno\myworkspace\file.c` for Windows
Depot syntax	`//depot/source/module/file.c`
Client syntax	`//myworkspace/file.c`

Wildcards

The Helix server system allows the use of these wildcards:

Wildcard	Meaning
`*`	Matches all characters except slashes within one directory.
`...`	Matches all files under the current working directory and all subdirectories. Matches anything, including slashes, and does so across subdirectories.
`%%1 - %%9`	Positional specifiers for substring rearrangement in filenames, when used in views.

Examples of wildcard expressions:

Expression	Matches
`J*`	Files in the current directory starting with `J`
`*/help`	All files called `help` in current subdirectories
`//gra*/dep*`	`//graph/depot/`, `//graphs/depots`, `gravity/deposits` but not `//graph/depot/release1/`
`./...`	All files under the current directory and its subdirectories
`./....c`	All files under the current directory and its subdirectories, that end in `.c`
`/usr/bruno_ws/...`	All files under `/usr/bruno_ws`
`//companytools/...` `//bruno_ws/...`	All files in the depot named `companytools` or the workspace named `bruno_ws`
`//depot/...`	All files in the depot (with the default name of "depot")
`//depot/main/rel...`	`//depot/main/rel/`, `//depot/main/releases`/, `//depot/main/release-note.txt`, `//depot/main/rel1/product1` and so on
`//...`	All files in all depots

Expression	Matches
`//depot/dir1/%%1.%%2 //bruno_ws/filesbytype/%%2/%%1`	This example uses positional specifiers in client view mapping to rearrange the sync'd files by file type. The depot files with a given extension, such as `.txt`, are sync'd into a workspace directory that bears the extension name, such as (`/txt`): ■ for the depot, • specify the file name at position 1 by using `%%1` • specify the file extension at position 2 by using `%%2` ■ for the workspace, • reverse the order so that the extension `%%2` is before the file name `%%1` • substitute a forward slash (`/`) of a directory for the period (`.`) of a file extension Get the latest version from the depot into the workspace: `sync -f //depot/dir1/...`

Depot file with extension ...	results in Workspace directory with file ...
`//depot/dir1/readme.doc`	`//bruno_ws/filesbytype/doc/readme`
`//depot/dir1/readme.md`	`/Users/bruno_ws/filesbytype/md/readme`
`//depot/dir1/readme.pl`	`/Users/bruno_ws/filesbytype/pl/readme`
`//depot/dir1/readme.txt`	`/Users/bruno_ws/filesbytype/txt/readme`

Using revision specifiers

File specifiers can be modified by appending `#` or `@` to them.

The `#` and `@` specifiers refer to specific revisions of files as stored in the depot:

Modifier	Meaning
`file#n`	Revision specifier: The *n*th revision of `file`.
`file#none` `file#0`	The nonexistent revision: If a revision of `file` exists in the depot, it is ignored. This is useful when you want to remove a file from the client workspace while leaving it intact in the depot, as in `p4 syncfile#none`. The filespec `#0` can be used as a synonym for `#none` - the nonexistent revision can be thought of as the one that "existed" before the first revision was submitted to the depot.
`file#head`	The head revision (latest version) of `file`. Except where explicitly noted, this is equivalent to referring to the file without a revision specifier.
`file#have`	The revision on the current client: the revision of file last `p4 sync`ed into the client workspace.
`file@n`	Change number: The revision of `file` immediately after changelist `n` was submitted.
`file@=n`	Change number: The revision of `file` at the specified changelist number `n`. `n` can be a submitted or a shelved change number. `n` cannot be a pending (non-shelved) change number.
`file @labelname`	Label name: The revision of `file` in the label `labelname`.
`file@ clientname`	Client name: The revision of `file` last taken into client workspace `clientname`. Note that deleted files (that is, files marked for `delete` at their latest revision) are not considered to be part of a workspace.
`file @datespec`	Date and time: The revision of `file` at the date and time specified. If no time is specified, the head revision at 00:00:00 on the morning of the date specified is returned. Dates are specified `yyyy/mm/dd:hh:mm:ss` or `yyyy/mm/ddhh:mm:ss` (with either a space or a colon between the date and the time). The datespec `@now` can be used as a synonym for the current date and time.

Revision specifiers can be used to operate on many files at once: `p4 sync //myclient/...#4` copies the fourth revision of all non-open files into the client workspace.

If specifying files by date and time (that is, using specifiers of the form `file@datespec`), the date specification should be parsed by your local shell as a single token. You may need to use quotation marks around the date specification if you use it to specify a time as well as a date.

Files that have been shelved can also be accessed with the `p4 diff`, `p4 diff2`, `p4 files`, and `p4 print` commands, using the revision specifier `@=`***`change`***, where ***`change`*** is the pending changelist number.

Some Helix server file specification characters may be intercepted and interpreted by the local shell, and need to be escaped before use. For instance, `#` is used as the comment character in most UNIX shells, and `/` may be interpreted by (non-Helix server) DOS commands as an option specifier. File names with spaces in them may have to be quoted on the command line.

For information on these and other platform-specific issues, see the *Release Notes* for your platform.

Using revision ranges

A few Helix server commands can use revision ranges to modify file arguments. Revision ranges are two separate revision specifications, separated by a comma. For example, `p4 changesfile#3,5` lists the changelists that submitted file `file` at its third, fourth, and fifth revisions.

Revision ranges have two separate meanings, depending on which command you're using. The two meanings are:

- Run the command on all revisions in the specified range. For example, `p4 jobs //...#20,52` lists all jobs fixed by any changelist that submitted any file at its 20th through 52nd revision.

  Revision ranges implicitly start at #1, for example, `p4 fixes//depot/file.c#5` implies all jobs fixed by revisions 1 through 5. (To see only those jobs that were fixed by revision 5, you would have to specify `p4 fixes//depot/file.c#5,5`.)

  This interpretation of revision ranges applies to `p4 changes`, `p4 fixes`, `p4 integrate`, `p4 jobs`, and `p4 verify`.

- Run the command on only the highest revision in the specified range. For example, the command `p4 print` **file@30,50** prints the highest revision of file `file` submitted between changelists 30 and 50. This is different than `p4 print` **file@50**: if revision #1 of file `file` was submitted in changelist 20, and revision #2 of file `file` was submitted in changelist 60, then `p4 print` **file@30,50** prints nothing, while `p4 print` **file@50** prints revision #1 of `file`.

  The commands `p4 files`, `p4 print`, and `p4 sync` all use revision ranges in this fashion.

Revision ranges can be very powerful. For example, the command `p4 changesfile#3,@`***`labelname`*** lists all changelists that submitted file `file` between its third revision and the revision stored in label ***`labelname`***.

Limitations on characters in filenames and entities

When you name files and entities, such as users and clients, be aware of the following limitations.

Character	Helix server Usage	Not allowed for the entity you create with

`...`
`*`
`%%`
	recursive subdirectory wildcard
	file matching wildcard
	positional substitution wildcard
		`p4 user`, `p4 client`, `p4 depot`, `p4 label`, `p4 job`, `p4 stream`

`/`	separating pathname components OK for `p4 user`, `p4 label`, and `p4 job`	`p4 client`, `p4 depot` and not allowed in the name of a file
`\`	No special meaning for Helix server.	(Allowed, but be aware that Microsoft Windows uses the backslash as the separator for pathname components.)
`@`	prefix to the identifier of a changelist, label, client, or datespec.	`p4 user`, `p4 client`, `p4 depot`, `p4 label`, `p4 job`, `p4 stream`
`#`	specifying the revision number	`p4 user`, `p4 client`, `p4 depot`, `p4 label`, `p4 job`, `p4 stream`
`1` or `1234`	changelist numbers are purely numeric	A purely numberic identifier is NOT allowed for a user, client, depot, label, job, or stream, so consider something like `1a` or `1_234`.

To refer to files containing the Helix server revision specifier wildcards (`@` and `#`), file matching wildcard (`*`), or positional substitution wildcard (`%%`) in either the file name or any directory component, use the ASCII expression of the character's hexadecimal value. ASCII expansion applies only to the following four characters:

Character	ASCII expansion
`@`	`%40`
`#`	`%23`
`*`	`%2A`
`%`	`%25`

To add a file such as `status@june.txt`, force a literal interpretation of special characters by using:

```
$ p4 add -f //depot/path/status@june.txt
```

When you submit the changelist, the characters are automatically expanded and appear in the change submission form as follows:

```
//depot/path/status%40june.txt
```

After submitting the changelist with the file’s addition, you must use the ASCII expansion in order to sync it to your workspace or edit it within your workspace:

```
$ p4 sync //depot/path/status%40june.txt
$ p4 edit //depot/path/status%40june.txt
```

Most special characters tend to be difficult to use in filenames in cross-platform environments: UNIX separates path components with `/`, while many DOS commands interpret `/` as a command line switch. Most UNIX shells interpret `#` as the beginning of a comment. Both DOS and UNIX shells automatically expand `*` to match multiple files, and the DOS command line uses `%` to refer to variables.

Similarly, although non-ASCII characters are allowed in filenames and Helix server identifiers, entering these characters from the command line may require platform-specific solutions. Users of GUI-based file managers can manipulate such files with drag-and-drop operations.

Views

There are three types of views: *client views*, *branch views*, and *label views*.

- Client views map files in the depot to files in the client workspace
- Branch views map files in the depot to other parts of the depot
- Label views associate groups of files in the depot with a single label.

Each type of view consists of lines that map files from the depot into the appropriate namespace. For client and branch views, the mappings consist of two file specifications. The left side of the mapping refers to the depot namespace, and the right side of the mapping refers to the client workspace or depot namespace. For label views, only the left side (the depot namespace) of the mapping is necessary because the files are automatically associated with the desired label.

- All views construct a one-to-one mapping between files in the depot and the files in the client workspace, branch, or label.
- If more than one mapping line refers to the same file(s), the earlier mappings are overridden.
- Mappings beginning with a hyphen (`-`) specifically exclude any files that match that mapping.
- In client views, mappings beginning with a plus sign (`+`) overlay previous mappings. (Overlay mappings do not apply to branch or label views.)

"File specifications" on page 695 within mappings are provided in the usual Helix server syntax, beginning with `//`, followed by the depot name or workspace name, and followed by the actual file name (s) within the depot or workspace. (You cannot use revision specifiers in views.)

Usage Notes

Views are set up through the **`p4 client`**, **`p4 branch`**, or **`p4 label`** commands as part of the process of creating a client workspace, label view, or branch view respectively.

The order of mappings in a client or branch view is important because a later mapping in a view always overrides an earlier mapping.

For example, in the view defined by the following two mappings:

```
//depot/...     //ws/...
//depot/dir/... //ws/dir2/...
```

the entire depot is mapped to the client workspace, but the file **`//depot/dir/file.c`** is mapped to **`//ws/dir2/file.c`** and overrides the previous mapping.

If the order of the lines in the view is reversed,

```
//depot/dir/... //ws/dir2/...
//depot/...     //ws/...
```

the file `//depot/dir/file.c` is mapped to `//ws/dir/file.c` because the second mapping overrides by the first mapping.

Spaces in path and file names

If a path or file name in a workspace view, branch view, or label view contains spaces, use quotes (`"`) to enclose the path:

```
//depot/v1/... "//ws/version one/..."
```

Special characters in path and file names

To map file and directory names that contain the characters `@`, `#`, `*`, or `%`, (that is, to interpret such characters as components of path and filenames, and *not* as Helix server wildcards), expand the characters to their ASCII equivalents :

Character	ASCII expansion
`@`	`%40`
`#`	`%23`
`*`	`%2A`
`%`	`%25`

Client Views

Client views are used to map files in the depot to files in client workspaces. A client workspace is an area in which users perform their work. Files are synced from the depot to a client workspace, opened for editing, edited, and checked back into the depot.

When files are synced, they are copied from the depot to the locations in the client workspace to which they were mapped. Likewise, when files are submitted back into the depot, the files are copied from the client workspace to their proper locations in the depot.

The following table lists some examples of client views:

Client View	Sample Mapping
Full client workspace mapped to entire depot	`//depot/... //ws/...`
Full client workspace mapped to part of depot	`//depot/dir/... //ws/...`

Client View	Sample Mapping

Some files in the depot are excluded from the client workspace with the hyphen (`-`)	`//depot/dir/...             //ws/...`
	`-//depot/dir/exclude/...   //ws/dir/exclude/...`

Some files in the depot are mapped to a different part of the client workspace	`//depot/...        //ws/...`
	`//depot/rel1/...   //ws/release1/...`

Files in the client workspace are mapped to different names than their depot names.	`//depot/dir/old.* //ws/renamed/new.*`
Portions of filenames in the depot are rearranged in the client workspace	`//depot/dir/%%1.%%2 //ws/dir/%%2.%%1`

By default, if two lines map to the same place in the workspace, the first line is ignored.	`//depot/dir1/... //ws/build/...`
	`//depot/dir2/... //ws/build/...`

An overlay mapping with the plus (`+`) sign allows "many-to-one" mapping. Files from more than one depot directory map to the same place in the workspace.	`//depot/`**`dir1`**`/...  //ws/build/...`
	`+//depot/`**`dir2`**`/... //ws/build/...`

A ditto mapping with the ampersand (`&`) sign allows "one-to-many" mapping. Files from the depot map to more than one location in the workspace. Note that workspace files on a line that begins with `&` are read-only.	`//depot/shared/include/...  //ws/`**`dir1`**`/include/...`
	`&//depot/shared/include/...`
	`//ws/`**`dir2`**`/include/...`

To create a client view, use `p4 client` to bring up a screen where you can specify how files in the depot are mapped to the files in your client workspace.

Note
A view spec can map files in a depot of type **`graph`**. For details, see the section "Including Graph Depot repos in your client" on page 113 in the topic **`p4 client`**.

Branch Views

Branching of the source tree allows multiple sets of files to evolve along different paths. The creation of a branch view allows Helix server to automatically manage the file copying and edit propagation tasks associated with branching.

Branch views map existing areas of the depot (the source files) onto new areas of the depot (the target files). They are defined in a manner similar to that used for defining client views, but rather than mapping files directly into a client workspace, they merely set up mappings within the depot. Because integration can take place in either direction, every line in a branch view must be unambiguous in both directions; overlay mappings are therefore not permitted in branch views.

Branch View	Sample Mapping
New code branching off from the main codeline	`//depot/main/...    //depot/1.1dev/...`

Rearranging directories in the new release	`//depot/main/...    //depot/1.1dev/...`
	`//depot/main/*.c    //depot/1.1dev/src/*.c`
	`//depot/main/*.txt`
	`//depot/1.1dev/doc/*.txt`

To create a branch view, use this syntax:

`"p4 branch" on page 75` ***`newbranch`***

This command brings up a screen (similar to the one associated with `"p4 client" on page 100`) and allows you to map the donor files from the main source tree onto the target files of the new branch.

No files are copied when a branch view is first created. To copy the files, you must ensure that the newly-created files are included in any client view intending to use those files. You can do this by adding the newly-mapped branch of the depot to your current client view and performing a `" p4 sync" on page 574` command.

Label Views

Label views assign a label to a set of files in the depot. Unlike client views and branch views, a label view does not copy any files; label views are used to limit the set of files that are taggable by a label.

Label View	Sample Mapping
A new release	`//depot/1.1final/...`
The source code for the new release	`//depot/1.1final/src/...`

A distribution suitable for clients	`//depot/1.1final/bin/...`
	`//depot/1.1final/doc/...`
	`//depot/1.1final/readme.txt`

To create a label, use **`p4 label`*`labelname`***, and enter the depot side of the view. Because a label is merely a list of files and revision levels, only the depot side (the left side) of the view needs to be specified, and overlay mappings are not permitted.

File types

Perforce supports a set of "Base filetypes" below. File type modifiers are then applied to the base types allowing for support of RCS keyword expansion, file compression, and more.

When adding files, Helix server:

- examines the typemap table to see if the system administrator has defined a file type for the file(s) being added. If a match is found, the file's type is set as defined in the typemap table. (See also "p4 typemap" on page 592)
- If a match is *not* found, Perforce examines the first bytes of the file based on the `filesys.binaryscan` configurable (by default, 65536 bytes) to determine whether it is `text` or `binary`, and the files are stored in the depot accordingly.

By default, text file revisions are stored in reverse delta format. Newly-added text files larger than the limit imposed by the `filetype.maxtextsize` configurable (by default, 10 MB) are assigned filetype `text+C` and stored in full. Files compressed in the `.zip` format (including `.jar` files) are also automatically detected and assigned the type `ubinary`. Other binary revisions are stored in full, with compression.

(Files in unicode environments are detected differently. For details, see the *Internationalization Notes*.)

Helix server administrators can use the

- type mapping feature (`p4 typemap`) to override Perforce's default file type detection mechanism. This feature is useful for `binary` file formats (such as Adobe PDF, or Rich Text Format) where files can start with large portions of ASCII text, and might otherwise be mistaken for `text` files.
- `filesys.binaryscan` and `filetype.maxtextsize` configurables (see `p4 configure`) to change the default limits of 65536 bytes for text/binary detection and the 10 MB RCS text file size limit.

Base filetypes

The base Perforce file types are:

Keyword	Description	Comments	Stored as
`text`	Text file	Synced as text in the workspace. Line-ending translations are performed automatically.	deltas in RCS format
`binary`	Non-text file	Synced as binary files in the workspace. Stored compressed within the depot.	full file, compressed
`symlink`	Symbolic link	Helix server applications on UNIX, OS X, recent versions of Windows treat these files as symbolic links. On other platforms, these files appear as (small) text files. On Windows, you require `admin` privileges or an appropriate group policy must be set, otherwise, you get text files.	deltas in RCS format
`unicode`	Unicode file	Services operating in unicode mode support the `unicode` file type. These files are translated into the local character set specified by `P4CHARSET`. Line-ending translations are performed automatically. Services not in unicode mode do not support the `unicode` file type. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format
`utf8`	Unicode file	Synced in the client workspace **with** the UTF-8 BOM (byte order mark). Whether the service is in unicode mode or not, files are transferred as UTF-8 in the client workspace. Line-ending translations are performed automatically. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format **without** the UTF-8 BOM (byte order mark).
`utf16`	Unicode file	Whether the service is in unicode mode or not, files are transferred as UTF-8, and translated to UTF-16 (with byte order mark, in the byte order appropriate for the user's machine) in the client workspace. Line-ending translations are performed automatically. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format

Keyword	Description	Comments	Stored as
`apple`		A legacy format of Apple Inc.	
`resource`		For resource forks on legacy Apple computers, which are no longer supported by Apple Inc.	

File type modifiers

The file type modifiers are:

Modifier	Description	Comments
`+w`	File is always writable on client	
`+x`	Execute bit set on client	Used for executable files, such as a shell script.
`+k`	RCS keyword expansion	Expands RCS (Revision Control System) keywords. RCS keywords are case-sensitive. When using keywords in files, a colon after the keyword (for instance, `$Id:$`) is optional. UTC keywords are better suited to describe events in globally distributed installations. Supported keywords are as follows: ▪ `$Id$` ▪ `$Header$` ▪ `$Date$` Date of submission ▪ `$DateUTC$` Date of submission in UTC time zone ▪ `$DateTime$` Date and time of submission ▪ `$DateTimeUTC$` Date and time of submission in UTC time zone. ▪ `$DateTimeTZ$` Date and time of submission in the server's time zone, but including the actual time zone in the result. ▪ `$Change$` ▪ `$File$` ▪ `$Revision$` ▪ `$Author$`

Modifier	Description	Comments
`+ko`	RCS keyword expansion of ID and Header only	
`+l`	Exclusive open (locking)	If set, only one user at a time will be able to open a file for editing. Useful for binary file types (such as graphics) where merging of changes from multiple authors is meaningless.
`+C`	Perforce stores the full compressed version of each file revision	Default storage mechanism for `binary` files and newly-added `text`, `unicode`, and `utf16` files larger than 10MB.
`+D`	Perforce stores deltas in RCS format	Default storage mechanism for `text` files.
`+F`	Perforce stores full file per revision, uncompressed	Useful for large binaries, or for long ASCII files that are not read by users as text, such as PostScript files. In a depot of type graph, binary+F is valid for Git Large File Storage (LFS).
`+S`	Only the head revision is stored	Older revisions are purged from the depot upon submission of new revisions. Useful for executable or `.obj` files.
`+S`***`n`***	Only the most recent ***`n`*** revisions are stored, where ***`n`*** is a number from 1 to 10, or 16, 32, 64, 128, 256, or 512.	Older revisions are purged from the depot upon submission of more than ***`n`*** new revisions, or if you change an existing `+S`***`n`*** file's ***`n`*** to a number less than its current value. Earlier revisions unaffected. For details, see "Usage Notes" on page 712.
`+m`	Preserve original modtime	The file's timestamp on the local filesystem is preserved upon submission and restored upon sync. Useful for third-party DLLs in Windows environments.
`+X`	Archive trigger required	The Perforce service runs an `archive` trigger to access the file. See Triggering to affect archiving in the *Helix Core Server Administrator Guide*.

A file's type is normally preserved between revisions, but can be overridden or changed with the `-t` option during `add`, `edit`, or `reopen` operations:

- `p4 add -t` ***`filetype filespec`*** adds the files as the specified type.
- `p4 edit -t` ***`filetype filespec`*** opens the file for `edit` as the specified type. The file's type is changed to the specified *filetype* only after it is submitted to the depot.
- `p4 reopen -t` ***`filetype filespec`*** changes the type of a file already open for `add` or `edit`.

The ***`filetype`*** argument is specified as `[`***`basetype`***`] +`***`modifiers`***. For example, to change `script.sh`'s type to executable text with RCS keyword expansion, use `p4 edit` **-t text+kx script.sh**.

Partial filetypes are also acceptable. For example, to change an existing `text` file to `text+x`, use `p4 reopen` **-t +x script.sh**. Most partial filetype modifiers are added to the filetype, but the storage modifiers (`+C`, `+D`, and `+F`) replace the file's storage method. To remove a modifier, you must specify the full filetype.

Perforce file types for common file extensions

To learn about mapping file names to Perforce file types, see the `p4 typemap` command and the "File type modifiers" on page 709.

Some examples of how a file extension might correspond to a Perforce file type:

Perforce file type	File Extension	Note
`text`	`.css`, `.html`, `.ini`, `.java`, `.js`, `.ps`, `.svg`	text saved in the client's character set
`text+wx`	`.sh`	shell script that is writable and executable
`binary`	`.dll`, `gif`, `.pdf`, `.png`, `.jpg`, `.zip`	image files, compressed archives
`binary+F`	`.avi`, `.gz`, `.mpg`, `.docx`	Helix does not compress this file type, which might have its own compression
`binary+w`	`.exp`, `.lib`	some software development file types that should be writable by default
`binary+x`	`.exe`	executable on client

Keyword Expansion

RCS keywords are expanded as follows:

Keyword	Expands To	Example
`$Id$`	File name and revision number in depot syntax.	`$Id: //depot/path/file.txt#3 $`

Keyword	Expands To	Example
`$Header$`	Synonymous with `$Id$`.	`$Header: //depot/path/file.txt#3 $`
`$Date$`	Date of last submission in format *`YYYY/MM/DD`*.	`$Date: 2010/08/18 $`
`$DateTime$`	Date and time of last submission in format *`YYYY/MM/DDhh:mm:ss`*. Date and time are as of the local time on the Perforce service at time of submission.	`$DateTime: 2010/08/18 23:17:02 $`
`$Change$`	Perforce changelist number under which file was submitted.	`$Change: 439 $`
`$File$`	File name only, in depot syntax (without revision number).	`$File: //depot/path/file.txt $`
`$Revision$`	Perforce revision number.	`$Revision: #3 $`
`$Author$`	Perforce user submitting the file.	`$Author: edk $`

Usage Notes

- The type of an existing file can be determined with `p4 opened` or `p4 files`.
- *Delta storage* (the default mode with `text` files) is a method whereby only the differences (or *deltas*) between revisions of files are stored. *Full file* storage (the default mode with `binary` files) involves the storage of the entire file. The file's type determines whether full file or delta storage is used. Perforce uses RCS format for delta storage.
- Some of the file types are compressed to `gzip` format for storage in the depot. The compression occurs during the submission process, and decompression happens while syncing. The process is transparent to the user because the client workspace always contains the file as it was submitted.
- Symbolic links in non-UNIX client workspaces appear as small text files containing a relative path to the linked file. Editing these files on a non-UNIX client should be done with caution, as submitting them to the depot may result in a symbolic link pointing to a nonexistent file on the UNIX workspace.
- Changing a file's type does not affect earlier revisions stored in the depot.

For instance, changing a file's type by adding the `+S`*`n`* (temporary object) modifier tells Perforce to store only the most recent *`n`* revisions of the file in the depot. If you change an existing file into a temporary object, subsequent revisions (after the *`n`*th) will purge the revisions stored after the old head revision, but revisions to the file stored in the depot *before* the `+S`*`n`* modifier was used will remain unaffected. (Syncing to a non-head revision submitted *after* the `+S`*`n`* modifier was used will delete the file from your workspace. Such revisions are displayed as `purge` operations in the output of `p4 filelog`.)

- Running `p4 integrate` on temporary object files (`+S` and `+S`*`n`*) does not produce a lazy copy. The integrated `tempobj` file consumes additional diskspace on the shared versioning service.
- The modtime (`+m`) modifier is a special case: It is intended for use by developers who need to preserve a file's original timestamp.

  If a client workspace uses the `modtime` option, the file date is not guaranteed to advance for each revision. For example, if a file is copy integrated ("accept theirs"), its timestamp will reflect that of the source file. If a user checks in a file with an old date, the client workspace file will reflect that same, old date. Normally, Perforce updates the timestamp when a file is synced. The modtime option enables a user to ensure that the timestamp of a file in a client workspace after a `p4 sync` will be the original timestamp existing *on the file* at the time of submission (that is, *not* the time at the Perforce versioning service at time of submission, and *not* the time on the user's workstation at the time of sync).

  The most common case where this is useful is development involving the third-party DLLs often encountered in Windows environments. Because the timestamps on such files are often used as proxies for versioning information (both within the development environment and also by the operating system), it is sometimes necessary to preserve the files' original timestamps regardless of a Perforce user's client settings.

  If the `+m` modifier on a file is set, when syncing the file Perforce restores the file's original timestamp at the time of submit. This means that Perforce ignores:

  - the `modtime` ("file's timestamp at time of submission")
  - the `nomodtime` ("date and time on the client at time of sync") option setting of the client workspace
- Versions of Perforce prior to the year 2000 used a set of keywords to specify file types. The following table lists the older keywords and their current base file types and modifiers:

Old Keyword	Description	Base Filetype	Modifiers
`text`	Text file	`text`	none
`xtext`	Executable text file	`text`	`+x`
`ktext`	Text file with RCS keyword expansion	`text`	`+k`

Old Keyword	Description	Base Filetype	Modifiers
`kxtext`	Executable text file with RCS keyword expansion	`text`	`+kx`
`binary`	Non-text file	`binary`	none
`xbinary`	Executable binary file	`binary`	`+x`
`ctext`	Compressed text file	`text`	`+C`
`cxtext`	Compressed executable text file	`text`	`+Cx`
`symlink`	Symbolic link	`symlink`	none
`ltext`	Long text file	`text`	`+F`
`xltext`	Executable long text file	`text`	`+Fx`
`ubinary`	Uncompressed binary file	`binary`	`+F`
`uxbinary`	Uncompressed executable binary file	`binary`	`+Fx`
`tempobj`	Temporary object	`binary`	`+FSw`
`ctempobj`	Temporary object (compressed)	`binary`	`+Sw`
`xtempobj`	Temporary executable object	`binary`	`+FSwx`
`xunicode`	Executable unicode	`unicode`	`+x`
`xutf16`	Executable UTF-16	`utf16`	`+x`

Configurables

Configurables allow you to customize a Helix Core service.

	To see details about a configurable, click a letter: "A" on page 723 — "C" on page 733 — "D" on page 734 — "F" on page 747 — "J" on page 756 — "L" on page 757 — "M" on page 763 — "N" on page 768 — "P" on page 792 — "R" on page 793 — "S" on page 803 — "T" on page 833 — "Z" on page 834

Configurables that affect the server

Use `p4 configure` to set or unset configurables that affect a Helix server. These configurables are also described in `p4 help configurables`. For more information on options for setting server configurables and their order of precedence, see `p4 configure`.

Tip

- The `DistributedConfig` field of the server spec shows a line for each configurable that is set to a non-default value. In this field, you can edit the value, add a new line to set a different configurable to a non-default value, or delete a line to reset that configurable to its default value.
- "p4 configure" on page 122 `history` allows the super user to track the history of changes to the values of configurables.

server restart

Most configurables can be set dynamically. A subset of the configurables require the server to be stopped and restarted. In the alphabetical list of configurables, look for this Note:

> After you change the value of this configurable, you must explicitly "stop" the server.
>
> **Note**
> `p4 admin restart` is not sufficient.

For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

For Windows, see Starting and stopping the Helix server.

Configurables that affect the client

You can set configurables that affect the client in the following ways (shown in order of precedence):

- As command line global options that are passed at server startup. For example:

  ```
  $ p4 -u bruno -p perforce:1666 sync
  ```

- As entries in a `P4CONFIG` file. Set configurables like this:

  ```
  P4USER=bruno
  P4PORT=perforce:1666
  ```

  The following configurables can be set in a config file, and you can also set the variables listed for the `p4 help environment` command:

  - "filesys.binaryscan" on page 747, "filesys.bufsize" on page 747
  - "lbr.verify.out" on page 762
  - "net.keepalive.count" on page 773, "net.keepalive.disable" on page 774, "net.keepalive.interval" on page 774, "net.maxwait" on page 777, "net.rfc3484" on page 789, "net.tcpsize" on page 791,
  - "sys.rename.max" on page 827, "sys.rename.wait" on page 828

- As entries in a `P4ENVIRO` file.

  You can use both `P4ENVIRO` and `P4CONFIG` files to define environment variables:

  - use the `P4CONFIG` file for those variables that have different values for different workspaces
  - use the `P4ENVIRO` file for those variables that remain constant for all projects. Values set in a `P4CONFIG` file override those set in a `P4ENVIRO` file.

- As set by the "p4 set" on page 513 command for Windows and OS X. For example:

  ```
  $ p4 set P4PORT=ssl:tea:1666
  ```

Configurables that affect the proxy

You can set configurables that affect the proxy in the following ways:

- Using a command line option. For example:

```
$ p4p -p tcp64:[::]:1999 -t central:1666 -r /var/proxyroot -v
proxy.monitor.level=2
```

- Using environment variables
- On Windows, using the "p4 set" on page 513 command:

```
C:\> p4 set -S "perforce_proxy" P4POPTIONS="-v
myconfig=myvalue"
```

Configurables - alphabetical list

The following table is an alphabetical list of the configurables. To see details about a configurable, click a letter and a name in the list.

Click a letter

"A" below — "C" below — "D" on the next page — "F" on the next page — "J" on page 719 — "L" on page 719 — "M" on page 719 — "N" on page 719 — "P" on page 720 — "R" on page 720 — "S" on page 721 — "T" on page 722 — "Z" on page 723

A
auth.2fa.persist auth.autologinprompt auth.default.method auth.id auth.ldap.cafile auth.ldap.order.N auth.ldap.pagesize auth.ldap.ssllevel auth.ldap.timeout auth.ldap.userautocreate auth.sso.allow.passwd auth.sso.args auth.sso.allow.nonldap auth.tickets.nounlocked
C
client.readonly.dir client.sendq.dir

security
server
server.allowfetch
server.allowpush
server.allowremotelocking
server.allowrewrite
server.commandlimits
server.depot.root
server.extensions.dir
server.global.client.views
server.locks.archive
server.locks.dir
server.locks.global
server.locks.sync
server.maxcommands
server.maxcommands.allow
serverlog.counter.N
serverlog.file.N
serverlog.maxmb.N
serverlog.retain.N
serverlog.version.N
serviceUser
spec.hashbuckets
ssl.secondary.suite
ssl.tls.version.min
ssl.tls.version.max
startup.N
statefile
submit.allowbgtransfer
submit.autobgtransfer
submit.collision.check
submit.identity
submit.noretransfer
submit.unlocklocked
sys.rename.max
sys.rename.wait
sys.threading.groups

T

template.client
template.label
track
triggers.io

Z
zerosyncPrefix

A

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`auth.2fa.persist`	Server	`1`	To disable "p4 login2" on page 351 `-p`, set to `0`. To make "p4 login" on page 349 `-p` implicitly invoke `p4 login2 -p`, set to `2`.	No
`auth.autologinprompt`	Server	`1`	The default value causes the command-line user to be prompted to log in. To disable, set to `0`	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`auth.default.method`	Server	`perforce`	The default method to use for authenticating new users. ■ `perforce` specifies that the user is to be authenticated using the `db.user` table. This is the default setting. If there are no active LDAP configurations, this setting might cause a new user to be authenticated against an AD/LDAP server, using an authentication trigger if such a trigger exists. ■ `ldap` specifies the user be authenticated against an AD/LDAP server without having to use authentication triggers. In addition, if you want new users to be automatically created when they have successfully authenticated against an AD/LDAP server, set the configurable "auth.ldap.userautocreate" on page 732 to a non-zero value.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`auth.id`	Server	`0`	An alphanumeric identifier that must be set for all servers in a distributed configuration if you want to implement single login to the master, which is then valid across all replica instances. You must also set "`rpl.forward.login`" on page 796 to `1` for each replica participating in the distributed configuration.	**On edge servers (not the master):** After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.
`auth.ldap.cafile`	Server	none	The path to a file that contains one or more PEM-formatted certificates used to verify the certificate presented by the AD/LDAP server when using SSL or TLS and `"auth.ldap.ssllevel" on page 728` is >=1.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

`auth.ldap.order.N`	Server	none	Specifies the name of the LDAP configuration to use for authentication and the order in which it should be used to search for a given user name. The lowest number confers the highest priority.
			
			You may skip numbers. For example:
			`auth.ldap.order.1=UK_LDAP`
			`auth.ldap.order.2=US_LDAP`
			`auth.ldap.order.5=RU_LDAP`
			
			If you want LDAP authentication to replace trigger-based authentification, see LDAP authentication in the Helix Core Server Administrator Guide, and note that the Testing and enabling LDAP configurations procedure requires a server restart.
				After you change the value of this configurable, you must explicitly "stop" the server.
				
				Note **`p4 admin restart`** is not sufficient.
				
				For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.
`auth.ldap.pagesize`	Server	`0`	Specifies the paging limit for LDAP searches with paged results. Set the configurable to a value less than the result limit of the LDAP server. The default value, `0`, means that paging is not enabled.	No
`auth.ldap.ssllevel`	Server	`0`	Level of SSL certificate validation: ▪ `0`: No validation; default. ▪ `1`: Certificate must be valid, but the common name is not checked. ▪ `2`: Certificate must be valid and the certificate common name matches the AD/LDAP server's host name.	No
`auth.ldap.timeout`	Server	`30`	The time in seconds to wait before giving up on a connection attempt.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`auth.sso.allow.passwd`	Server	`0`	To allow users who authenticate against the Perforce database (as opposed to LDAP or other auth triggers) to fall back to password authentication despite an **`auth-check-sso`** trigger being on the server, set to **`1`** If LDAP is enabled, see the "auth.sso.nonldap" on page 731 configurable. See "p4 login behavior with auth-check-sso trigger" under "Single signon and auth-check-sso triggers" in *Helix Core Server Administrator Guide*.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`auth.sso.args`	Server	unset	This configurable can be set to string value to send to the client-side "P4LOGINSSO" on page 666 script when an `auth-check-sso` trigger is in use. This string is substituted for the `%ssoArgs%` variable in the `P4LOGINSSO` environment variable, as the client executable is being invoked. For example, the SAML agent can get the identity provider (IdP) URL as follows: `p4 configure set auth.sso.args=--idpUrl=`*`yourURLforIdP`* where `--idpUrl=`*`yourURLforIdP`* replaces `%ssoArgs%`	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`auth.sso.nonldap`		`0`	■ When set to `0` and LDAP authentication is enabled, users whose AuthMethod is perforce will authenticate by password against the Perforce database. ■ If set to `1`, those users will be required to authenticate using a client-side "P4LOGINSSO" on page 666 script. ■ Note: If this configurable and `auth.sso.allow.passwd` are both set to `1`, users whose AuthMethod is perforce will be able to authenticate using a client-side "P4LOGINSSO" on page 666 script, or fallback to authenticating by password against the Perforce database. See "p4 login behavior with auth-check-sso trigger" under "Single signon and auth-check-sso triggers" in *Helix Core Server Administrator Guide*.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

`auth.tickets.nounlocked`	Server	`0`	If set to `1` or `2`, prevents "p4 login" on page 349 `-a` from issuing host unlocked tickets. In other words, `1` or `2` enforce host locked tickets, which are restricted to the one host with the correct IP address.
			
			`1` means the `-a` flag is silently ignored and the users are always issued host locked tickets.
			
			`2` means the `-a` flag is explicitly disabled and users get an error if they try to use it.
			
			If either value is set, the tagged output from "p4 info" on page 261
			
			`p4 -ztag info`
			
			shows that `unlockedTickets` is disabled:
			
			`... unlockedTickets disabled`
				No

`auth.ldap.userautocreate`	Server	`0`	`0` means no automatic creation of users. `1` means that if `auth.default.method` is set to `ldap`, users are auto-created when they log in to Perforce and they have been successfully authenticated against an active directory AD/LDAP server using `p4 login`. `2` is similar to `1` but also requires the user already have permissions on the server through the protections table.	No

C

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`client.readonly.dir`	Server	none	The path of the directory where db.* files for a read-only client will be placed. For example, if you create a read-only client whose name is `myroc` and `client.readonly.dir` is set to `/perforce/1`, then syncing files using this client will write to the database `/perforce/1/server.dbs/client/`***`hashdir`***`/db.myroc`.	No
`client.sendq.dir`	Server	none	For parallel sync: To avoid lock contention on the database table used for processing parallel syncs, set the `client.sendq.dir` configurable. This specifies the directory that will contain a separate db.sendq table for each client. Consider specifying the same directory you use for `client.readonly.dir`.	No

D

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`db.monitor.interval`	Server	`30` in 2018.2 and `0` prior to 2018.2	The value of `0` means the feature is off. A non-zero value specifies the number of seconds that the Helix Server waits before checking if any process in the monitor table is marked for termination. If the Helix Server determines that any such process is blocked because it is waiting for client input, the Helix Server terminates it. See also: ■ "Enabling process monitorning" in the Helix Core Server Administrator Guide ■ the Support Knowledgebase article, "Fixing a hung Helix server server"	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`db.monitor.shared`	Server	256 K	The value sets the maximum size of memory allotted to the `db.monitor` table, which tracks the p4d commands that are currently running. Setting this configurable to `0` means that the table is written to disk. However, writing the table to memory is recommended to improve performance. If the size of the table exceeds the value of `db.monitor.shared`, an error is returned. Commands are still executed, but they are not recorded in the table. **Note** The value is in database pages of 8 kilobytes. For example, a value of 4096 means 32 MB.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service. For Windows, see Starting and stopping the Helix server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
			See also the "Options" on page 126 under the **`p4 configure`** command, and note that "`p4 configure`" on page 122 **`show`** indicates the actual maximum, but **`p4 configure show allservers`** indicates a manual preference that is only enforced if sufficient memory is available.	
`db.monitor.term.allow`	Server	unset	To allow users to terminate their own processes, set to 1. To also allow users to pause and resume their own processes, set to 2. **Note** A minimum of **`read`** permission in the protections table is required for these users to be able to use "`p4 monitor`" on page 371 **`terminate`**.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`db.peeking`	Server	`2`	Enable and configure lockless reads. When enabled, many common commands no longer block other commands attempting to update the database. See "Commands implementing lockless reads" in the *Helix Core Server Administrator Guide* . `0`: Disable peeking. Behavior is identical to 2013.2 and earlier. `1`: New locking order is enabled, peeking is disabled, (diagnostic use only). `2`: New locking order is enabled, peeking is enabled, **`hx`**/**`dx`** optimization on. `3`: New locking order is enabled, peeking is enabled, **`hx`**/**`dx`** optimization is off.	**Note** Beginning with the 2017.1 release, this configurable is dynamic and no server restart is required.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`db.reorg.disable`	Server	`1`	`1`: The default disables passive B-tree reorganization, which we recommend for solid-state drive (SSD) storage. `0`: Enable passive B-tree reorganization. This might: ▪ be beneficial if data storage involves a spinning disk because it might reduce the need to periodically recreate database tables from a checkpoint to improve performance ▪ increase the size of some database tables due to the allocation of new contiguous pages	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`db.replication`	Server	unset	Control behavior of commands that access metadata (`db.*` files) on the Helix server server: `readonly`: User commands that read metadata are accepted; commands that modify metadata are rejected. Equivalent to starting a replica with the `p4d -M readonly` option. This configurable cannot be set globally; you must specify a server id.	No
`dbjournal.bufsize`	Server	`16K`	Buffer size for journal and checkpoint read/write operations.	No
`dbopen.nofsync`	Server	`0`	Set to `1` to disable `fsync()` call when server closes a `db.*` database file, and permit the OS to determine when to write the modified data.	No
`defaultChangeType`	Server	none	Default type for new changelists: either `public` or `restricted`. If unset, new changelists are `public`.	No
`dm.annotate.maxsize`	Server	`10M`	Maximum revision size for `p4 annotate`.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.domain.accessforce`	Server	`3600`	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.	No
`dm.domain.accessupdate`	Server	`300`	Wait this many seconds before requesting a write lock to update an access time.	No
`dm.grep.maxrevs`	Server	`10K`	Maximum number of revisions that can be searched with `p4 grep`.	No
`dm.info.hide`	Server	`0`	If set to `1`, and the user is not authenticated, "p4 info" on page 261 hides: `Server name` `Server address` `Server uptime` `Server license ip address` and the license string is either `none` or `Licensed`.	No
`dm.integ.engine`	Server	`3`	By default, use new integration engine with `p4 integrate`. (The `p4 merge` command always uses the v3 integration engine regardless of this setting.) Sites that wish to continue to use the old (2006.1) integration logic must set this configurable to 2 by running `p4 configure set dm.integ.engine=2`.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.keys.hide`	Server	`0`	If set to `1` or `2`, `p4 keys` requires `admin` access. If set to `2`, `p4 key` requires `admin` access.	No
`dm.password.minlength`	Server	`8`	Default minimum password length for servers where `security` is set to a nonzero value.	No
`dm.protects.allow.admin`	Server	`0`	Allow Perforce administrators to use `-a`, `-g`, and `-u` with `p4 protects`. By default, only superusers can use these options.	No
`dm.proxy.protects`	Server	`1`	Determine (in accord with the use of IP addresses in the protections table) whether a user can access a server from a given IP address. By default, if a connection comes through an intermediary, the `proxy-` prefix is prepended to the client IP address. Set this variable to `0` if you do not want to have connections that come in through an intermediary to have the `proxy-` prefix. For more information, see the `p4 protect` command.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.repo.noautocreate`	Server	`0`	Control behavior of automatic repo creation in a depot of type `graph`. `0`: When a user does a git push, if that repo does not already exist in the graph depot, this new repo is added in the graph depot. `1`: When a user does a git push, if that repo does not already exist in the graph depot, the git push fails and the repo is not added to the graph depot.	No
`dm.resolve.attribs`	Server	`1`	Enable resolve for attributes set with `p4 attribute`.	No
`dm.rotatelogwithjnl`	Server	`1`	Set to `0` to disable log rotation after journal rotation. By default, when the journal is rotated, any structured logs are also rotated. Disabling this behavior can help when you're doing frequent journal rotations and you want the log rotated on a different schedule.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.shelve.accessupdate`	Server	300 seconds	When a shelf is viewed or modified, update its access time if its last access time was longer than the limit specified by the value of `dm.shelve.accessupdate` Use the `p4 -Ztag change -o` command to display the access time for shelved files.	No
`dm.shelve.maxfiles`	Server	`10M`	Maximum number of files that can be shelved with `p4 shelve`.	No
`dm.shelve.maxsize`	Server	`0`	Maximum size of a file that can be shelved, or `0` for unlimited.	No
`dm.shelve.promote`	Server	`0`	Enable to make edge servers automatically promote shelved files to the commit server. An alternative is manual promotion with the `-p` option of `p4 shelve`. To determine whether to set this configurable, see "Explicitly promoting shelves" under "Promoting shelved changelist" in *Helix Core Server Administrator Guide*.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.user.allowselfupdate`	Server	`1`	(MFA): The default of `1` allows a user to set the value of her or his `%email%` and `%fullname%` variables. To prevent users from changing the value of these variables, set to `0`.	No
`dm.protects.streamspec`	Server	`0`	To start enforcing the stream spec permissions described at "p4 protect" on page 401, set to `1`	No
`dm.user.accessforce`	Server	`3600`	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.	No
`dm.user.accessupdate`	Server	`300`	Wait this many seconds before requesting a write lock to update an access time.	No
`dm.user.loginattempts`	Server	`3`	Number of password attempts before delay. When the number of consecutive failed login attempts equal this value, a delay is added before next attempt is possible. The delay is 1 second plus 1 second for every next failed attempt up to the maximum of 10 seconds.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.user.noautocreate`	Server	`0`	Control behavior of automatic user creation. **Warning** By default, Helix server creates a new user whenever a previously unknown user invokes any command that can update the repository or its metadata. When executed by a nonexistent user, most Perforce commands cause a user to be created. You can control this behavior by setting the configurable with the `p4 configure` command. For greatest security, we recommend that only the Helix server superuser be allowed to create new users: `$ p4 configure set dm.user.noautocreate=2`	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
			Value | Meaning 0 | A user record is created whenever any new user invokes **any** command that updates the depot or its metadata (default). Many such commands exist, including "p4 ping" on page 391. 1 | New users must create their own user records by **explicitly** running `p4 user`. 2 | Only the Helix server superuser can create a new user, and the superuser does so by **explicitly** running `p4 user -f` *`username`*.	

Value	Meaning
`0`	A user record is created whenever any new user invokes **any** command that updates the depot or its metadata (default). Many such commands exist, including "p4 ping" on page 391.
`1`	New users must create their own user records by **explicitly** running `p4 user`.
`2`	Only the Helix server superuser can create a new user, and the superuser does so by **explicitly** running `p4 user -f` *`username`*.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`dm.user.resetpassword`	Server	`0`	If set, all new users created with a password are forced to reset their password before issuing any commands. This configurable applies only if the passwords for newly created users are set using the `Password:` field of the user specification. The password reset behavior for new users that get initial passwords using the `p4 passwd` command after the user is created is not affected by the setting of this configurable.	No

F

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.binaryscan`	Client	`64K`	Scan the first `filesys.binaryscan` bytes for binary data when running `p4 add`.	No
`filesys.bufsize`	Client, Server	`64K`	Buffer size for client-side read/write operations.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.checklinks`	Server	`0`	Use to control symbolic links. `0` means no link checking, so directory `symlinks` can occur. `1` blocks attempts to `p4 add` a symlink to a directory. ▪ `p4 add /path/to/a/symlinkDirectory` fails ▪ `p4 add /path/to/an/embeddedSymlink/readme.txt` fails `2` same as `1` except allows the user to bypass the check by using the `-f` (force) option. ▪ `p4 add -f /path/to/a/symlinkDirectory` succeeds ▪ `p4 add -f /path/to/an/embeddedSymlink/readme.txt` succeeds	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
			`3` allows a symlink to a directory that terminates the path without having to use `-f`: ▪ `p4 add /path/to/a/symlinkDirectory` succeeds without using `-f` because the directory that is represented by a symlink is at the end of the path ▪ `p4 add /path/to/an/embeddedSymlink /subdirectory/readme.txt` fails because the directory that is represented by a symlink is NOT at the end of the path ▪ `p4 add -f /path/to/an/embeddedSymlink /subdirectory/readme.txt` succeeds because `-f` is used	

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.depot.min`	Server	`250M`	Minimum disk space required for any depot before server rejects commands. (If there is less than **`filesys.depot.min`** disk space available for any one depot, commands are rejected for transactions involving all depots.) To specify size, use the following binary abbreviations, which are slightly different from the more familiar decimal abbreviations: **`t`** or **`T`** for tebibytes (1 T is approximately 1.1 TB) **`g`** or **`G`** for gibibytes (1 G is approximately 1.07 GB) **`m`** or **`M`** for mebibytes (1 M is approximately 1.05 MB) **`k`** or **`K`** for kibibytes (1 K is 1,024 bytes) You can also use a percentage to specify the relative amount of free disk space required.	No
`filesys.extendlowmark`	Client	`32K`	Minimum filesize before preallocation (Windows).	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.P4JOURNAL.min`	Server	`250M`	Minimum disk space required on server journal filesystem before server rejects commands. To specify size, use the following binary abbreviations, which are slightly different from the more familiar decimal abbreviations: `t` or `T` for tebibytes (1 T is approximately 1.1 TB) `g` or `G` for gibibytes (1 G is approximately 1.07 GB) `m` or `M` for mebibytes (1 M is approximately 1.05 MB) `k` or `K` for kibibytes (1 K is 1,024 bytes) You can also use a percentage to specify the relative amount of free disk space required.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.P4LOG.min`	Server	`250M`	Minimum disk space required on server log filesystem before server rejects commands. To specify size, use the following binary abbreviations, which are slightly different from the more familiar decimal abbreviations: `t` or `T` for tebibytes (1 T is approximately 1.1 TB) `g` or `G` for gibibytes (1 G is approximately 1.07 GB) `m` or `M` for mebibytes (1 M is approximately 1.05 MB) `k` or `K` for kibibytes (1 K is 1,024 bytes) You can also use a percentage to specify the relative amount of free disk space required.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.P4ROOT.min`	Server	`250M`	Minimum disk space required on server root filesystem before server rejects commands. To specify size, use the following binary abbreviations, which are slightly different from the more familiar decimal abbreviations: `t` or `T` for tebibytes (1 T is approximately 1.1 TB) `g` or `G` for gibibytes (1 G is approximately 1.07 GB) `m` or `M` for mebibytes (1 M is approximately 1.05 MB) `k` or `K` for kibibytes (1 K is 1,024 bytes) You can also use a percentage to specify the relative amount of free disk space required.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.TEMP.min`	Server	`250M`	Minimum disk space required for temporary operations before server rejects commands. To specify size, use the following binary abbreviations, which are slightly different from the more familiar decimal abbreviations: `t` or `T` for tebibytes (1 T is approximately 1.1 TB) `g` or `G` for gibibytes (1 G is approximately 1.07 GB) `m` or `M` for mebibytes (1 M is approximately 1.05 MB) `k` or `K` for kibibytes (1 K is 1,024 bytes) You can also use a percentage to specify the relative amount of free disk space required.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`filesys.windows.lfn`	Server, Client, Proxy	`1`	Set to `1` to support filenames longer than 260 characters on Windows platforms. Depending on the depth of your workspace path, this might need to be set on the client, server, and/or proxy (which acts as a client). A file name length of up to 32,767 characters is allowed. Each component of the path is limited to 255 characters. The server root or client root cannot be a long path.	No
`filetype.maxtextsize`	Server	`10M`	Maximum file size for text type detection.	No

J

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

`journalPrefix`	Server	unset	Prefix or directory location for rotated journals and checkpoints:
			`p4 configure show journalPrefix`
			`master#journalPrefix=/p4/ckps/master (configure)`
			`p4 admin checkpoint`
			`p4 -F %jfile% journals -m2`
			`/p4/ckps/master.ckp.539`
			`/p4/ckps/master.jnl.538`
				No

L

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`lbr.autocompress`	Server	`0`	Enabling this configurable specifies the storage method as compressed text (`ctext`) rather than RCS format text. The user still sees the file type as `text`. We recommend setting this variable with `p4 configure set lbr.autocompress=1` when: ■ using a commit/edge configuration ■ sharing archive files between servers ■ using pull-archive or edge-content triggers for external archive transfer	No
`lbr.bufsize`	Server, Proxy	`64K`	Buffer size for read/write operations to server's archive of versioned files.	No
`lbr.proxy.case`	Proxy	`1`	`1`: File paths are always case-insensitive. `2`: File paths are case-insensitive if server is case-insensitive. `3`: File paths are always case-sensitive.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`lbr.replica.notransfer`	Server	`0`	If set to `1`, suppresses direct on-demand file fetch, which forces the use of pull-archive triggers for transfers. See Triggers for external file transfer in *Helix Core Server Administrator Guide*.	No
`lbr.replication`	Server	unset	Control behavior of user commands that access versioned files on the Helix server server: `readonly`: Replicates version files when they are updated on the master. `shared`: For shared storage. (See "Configuring a replica with shared archives" in *Helix Core Server Administrator Guide*.) `cache`: Replicates version files only when referenced if they do not already exist on the replica. `none`: No access to versioned files is permitted. This configurable cannot be set globally; you must specify a server id. Equivalent to starting a replica `p4d` process with one of the `-D readonly`, `-D shared`, `-D cache`, or `-D none` options.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`lbr.retry.max`	Server	`50`	In the event of a failed transfer, a replica will make `lbr.retry.max` attempts to retrieve the file.	No
`lbr.stat.interval`	Server	`0`	Proxy file status interval. If set to a value, such as `30`, the LbrStatus table entry for a large file will be updated every 30 seconds, indicating that the proxy is making progress on the file transfer. If the proxy has not read some data for that file within the 30 seconds then the entry will not be updated. For more information, if you have installed the P4P, see the proxy help by typing at the command line `p4p -h`	No
`lbr.storage.allowsymlink`	Server	unset	To allow symlinks, set the `lbr.storage.allowsymlink` configurable to `1` This should only be done if all the symlinks only reference files and directories that are not under any scanned directory. See "p4 storage" on page 532.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`lbr.storage.delay`	Server	`86400`	Required number of seconds before "p4 storage" on page 532 `-d` scanner considers deleting storage records. 86400 seconds equates to 24 hours. This delay is to allow any in-progress submits and in-progress shelves to complete.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`lbr.storage.skipkeyed`	Server	`0`	If `0` (default), a digest is created for all keyword revisions. If set to `1`, no digest is created for keyword revisions and a warning message is logged. If set to `2`, no digest is created for keyword revisions and no message is logged.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.
`lbr.verify.in`	Server	`1`	Verify contents from the client to server? (`1` for yes, `0` for no)	No
`lbr.verify.out`	Client, Server	`1`	Verify contents from the server to client? (`1` for yes, `0` for no)	No
`lbr.verify.script.out`	Server	`1`	Set to `0` to prevent files of type `+X` from having their digest checked when transmitted from server to client. When source watermarking is used, sites have configured a `+X` archive trigger script that returns different results each time a file is sync'd or printed, in order to embed a user-specific string into the file contents during sync. This defeats the digest verification performed when sending the file to disk. Setting `lbr.verify.script.out` disables digest verification in this situation. Other files are still verified normally, as determined by the setting of `lbr.verify.out`.	No

M

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`minClient`	Server	none	Lowest version of client software permitted to connect to this server. The `minClient` version can be lower than, or equal to, the server version, but not higher than the server version. The syntax to set this configurable is: **`p4 configure set minClient=`*`version`*** The value of *version* can be a version string: **`p4 configure set minClient=2017.2`** or a client protocol level: **`p4 configure set minClient=83`** **Note** To discover which client protocol version corresponds to a given P4V version string, see the server log. The following server log entry shows that P4V 2017.3 is using v83 as the protocol version: **P4V**/MACOSX1011X86_64/**2017.3**/1582486/**v83** See also the Support Knowledgebase articles: ■ "How to force users to upgrade their client	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
			software" - "Helix Client Protocol Levels" **Tip** We recommend to that you provide your users a message if their client is blocked by this configurable. See `minClientMessage`.	
`minClientMessage`	Server	none	Message to issue if client software is too old, set by `p4 configure set minClientMessage=`*`message`*.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`monitor`	Server	`0`	Valid values for the monitor configurable are: ▪ `0`: Server process monitoring off. (Default) ▪ `1`: monitor active commands ▪ `2`: active commands and idle connections ▪ `3`: sames as `2`, but also includes connections that failed to initialize (stuck at the Init() phase) ▪ `5`: sames as `2`, but also includes a list of the files locked by the command for more than one second ▪ `10`: same as `5`, but also includes lock wait times ▪ `25`: sames as `10`, except that the list of files locked by the command includes files locked for *any* duration See the `p4 monitor` command.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

`monitor.lsof`	Server	none	When set on Unix platforms, enables the use of the `p4 monitor` command to display a list of locked files. Set to the following value:
			`$ path/lsof -F pln+`
			The value for ***path*** varies with the version of Unix you are using. For example:
			`$ /usr/bin/lsof -F pln`
			See the `p4 monitor` command.
				No

N

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.autotune`	Server, Client, Proxy, Broker	`1`	TCP connection changes to improve performance over long latency connections. To disable, set to `0`. Clients set by using "p4 set" on page 513 or "P4CONFIG" on page 650 files. Servers set by using "p4 configure" on page 122. **Note** On Windows-based platforms, send buffer sizes are not autotuned but are manually configurable with "net.tcpsize" on page 791. The 2017.2, 2018.1, and 2018.2 releases defaulted to `0`, for reasons explained in the Support Knowledgebase article, "Autotune - Improved performance over long latency TCP connections".	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.
`net.backlog`	Server, Proxy	`128`	Maximum length of queue for pending connections. Consider increasing if users cannot connect to servers that are heavily loaded.	No
`net.heartbeat.interval`	Server	`2000`	Milliseconds between sending heartbeats to the target server. **Tip** You can interactively invoke "p4 heartbeat" on page 251 to observe the effect of varying the value for this configurable.	The server must be restarted for any background heartbeat threads to start using the updated value for this configurable.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.heartbeat.wait`	Server	`2000`	Milliseconds to wait for a response from the target server. If this interval is exceeded, your `heartbeat-missing` trigger will fire if this is the first missing response. See Triggering on heartbeat (server responsiveness) in *Helix Core Server Administrator Guide*. **Tip** You can interactively invoke "p4 heartbeat" on page 251 to observe the effect of varying the value for this configurable.	The server must be restarted for any background heartbeat threads to start using the updated value for this configurable.
`net.heartbeat.missing.interval`	Server	`2000`	Milliseconds between sending heartbeats after a missing response. **Tip** You can interactively invoke "p4 heartbeat" on page 251 to observe the effect of varying the value for this configurable.	The server must be restarted for any background heartbeat threads to start using the updated value for this configurable.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.heartbeat.missing.wait`	Server	`4000`	Milliseconds to wait for a response from the target server after a missed response. If the response arrives before the request times out, your **`heartbeat-resumed`** trigger will fire. See Triggering on heartbeat (server responsiveness) in *Helix Core Server Administrator Guide*. **Tip** You can interactively invoke "p4 heartbeat" on page 251 to observe the effect of varying the value for this configurable.	The server must be restarted for any background heartbeat threads to start using the updated value for this configurable.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.heartbeat.missing.count`	Server	`5`	The number of consecutive missed heartbeat responses (see `net.heartbeat.missing.wait`) before the heartbeat is considered non-responsive or dead. When the count of consecutive missed responses reaches `net.heartbeat.missing.count`, your `heartbeat-dead` trigger will fire. See Triggering on heartbeat (server responsiveness) in *Helix Core Server Administrator Guide*. **Tip** You can interactively invoke "p4 heartbeat" on page 251 to observe the effect of varying the value for this configurable.	The server must be restarted for any background heartbeat threads to start using the updated value for this configurable.
`net.keepalive.count`	Server	`0`	Number of unacknowledged keepalives before failure. Similar to `tcp_keepalive_probes` at https://linux.die.net/man/7/tcp If `0`, defaults to the operating system behavior. See the TCP keepalive section in the Administrators Guide.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.keepalive.disable`	Server	`0`	If `0` and keepalive functionality is supported by the OS, keepalives are enabled on the socket. If `1`, keepalives are disabled on the socket. See the TCP keepalive section in the Administrators Guide.	No
`net.keepalive.idle`	Server	`0`	Idle time (in seconds) before starting to send keepalives. Similar to **`tcp_keepalive_time`** at https://linux.die.net/man/7/tcp If `0`, defaults to the operating system behavior. See the TCP keepalive section in the Administrators Guide.	No
`net.keepalive.interval`	Server	`0`	Interval (in seconds) between sending keepalive packets. Similar to **`tcp_keepalive_intvl`** at https://linux.die.net/man/7/tcp If `0`, defaults to the operating system behavior. See the TCP keepalive section in the Administrators Guide.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.maxfaultpub`	Proxy	`100`	A value in megabytes that controls the proxy's cache faulting behavior. A single `p4 sync` will not publish more than `net.maxfaultpub` megabytes of faults into `pdb.lbr`. You must restart the server after changing the value of this configurable.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.maxwait`	Client, Server, Proxy	unset	Time, in seconds, before a network connection times out. Best practice is *not* to set server-wide: if set on server, requires that users complete command-line forms within this limit. If set in user's individual `P4CONFIG` file, applies to user's workstation (and requires only that the versioning service reply to user requests within the allotted time limit). You must restart the server after changing the value of this configurable.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.mimcheck`	Server, Proxy	`1`	Man-in-the-middle network security level: This enables checks for possible interception or modification of data such as using an SSH tunnel or other TCP forwarder for users with passwords set. `0`: Disable MitM checks. `1`: Check proxy/broker connections in legacy contexts. `2`: Connections from clients are checked for TCP forwarding. `3`: Connections from clients, proxies, and brokers are checked for TCP forwarding. `4`: All connections are checked; client software older than release 2010.1 cannot connect. `5`: Requires that proxies, brokers, and all Perforce intermediate servers have valid logged-in service users associated with them. This allows administrators to prevent unauthorized proxies and services from being used. You must restart the server after changing the value of this configurable.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.
`net.parallel.batch`	Server	`8`	Specifies the number of files in a batch. See **p4 sync** on "Parallel processing" on page 578.	No
`net.parallel.batchsize`	Server	`512`	Specifies the number of bytes in a batch. See **p4 sync** on "Parallel processing" on page 578.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.parallel.max`	Server	`0`	Specifies your maximum number of threads for syncing files concurrently. A value greater than `1` enables parallel processing up to the specified number of threads, when syncing a client or submitting files. In addition to setting this variable, you must use the `--parallel` option to the `p4 sync` command or the `p4 submit` command to further describe the processing desired. If you use `net.parallel.submit.*` configurables to automate parallel processing, you do not need to use the `--parallel` option. Values can range between `0` and `100`. See the "p4 sync" on page 574 command or the "p4 submit" on page 558 command. **Tip** To enable parallel processing, you must set this configurable to a value greater than zero and also greater than or equal to a non-zero value of "net.parallel.threads" on page 786.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
			See also "Parallel processing" on page 578.	
`net.parallel.min`	Server	`9`	Specifies the minimum number of files in a parallel sync. A sync that is too small does not initiate parallel file transfers. See **p4 sync** on "Parallel processing" on page 578.	No
`net.parallel.minsize`	Server	`576K`	Specifies the minimum number of bytes in a parallel sync. A sync that is too small does not initiate parallel file transfers. See **p4 sync** on "Parallel processing" on page 578.	No
`net.parallel.shelve.batch`	Server	`8`	For automatic parallel processing: specifies the number of files in a batch. (See also `net.parallel.submit.batch`)	No
`net.parallel.shelve.min`	Server	`8`	For automatic parallel processing: specifies the number of files in a batch. (See also `net.parallel.submit.min`)	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.parallel.shelve.threads`	Server	`9`	For automatic parallel processing: specifies the number of threads to be used for sending files in parallel. The specified threads grab work in batches. The size of a batch is specified using the `net.parallel.shelve.batch` configurable.	No
`net.parallel.submit.batch`	Server	`0`	For automatic parallel processing: specifies the number of files in a batch. See the `p4 submit` command on "Parallel submits" on page 560.	No
`net.parallel.submit.min`	Server	`8`	For automatic parallel processing: specifies the minimum number of files to be sent in a parallel submit. See the `p4 submit` command on "Parallel submits" on page 560.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.parallel.submit.threads`	Server	`9`	For automatic parallel processing, specifies the number of threads for sending files in parallel for each submit. The specified threads grab work in batches. The size of a batch is specified using the "net.parallel.submit.batch" on the previous page configurable. See the `p4 submit` command on "Parallel submits" on page 560.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.parallel.sync.svrthreads`	Server	`0`	Can help prevent the degradation of network response. Reduces the number of parallel transmit threads for sync commands when the total number of concurrent user-transmit threads from all commands, including submit, would exceed the value of this configurable. Does NOT reduce parallel transmit threads for submit commands. To determine the value for this configurable, consider the average network utilization of each user-transmit thread and how much spare bandwidth to allocate for occasional peak loads. If parallel syncs are saturating the network, use the "p4 monitor" on page 371 **`show`** command to find out how many concurrent transmit threads are executing in the server. The default value of `0` means no reduction of parallel sync threads. For this configurable to take effect, also set the "monitor" on page 766 configurable to `1` or greater.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.parallel.threads`	Server	`0`	Specifies your default number of threads for syncing files concurrently. **Tip** To enable parallel processing, you must set this configurable to a value greater than `1`, but less than or equal to the value of "net.parallel.max" on page 781. See also "Parallel processing" on page 578. The specified threads grab work in batches.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.reuseport`	Server	`0`	Set `SO_REUSEPORT` for listening socket. You must restart the server after changing the value of this configurable.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.rfc3484`	Client, Server	`0`	If `1`, permit the operating system to determine whether IPv4 or IPv6 is used when resolving hostnames. This is applicable only if a host name (either F or unqualified is used). If an IPv4 literal address (for example, 127.0.0.1) is used, the transport is always `tcp4`, and if an IPv6 literal address (for example, `::1`) is used, then the transport is always `tcp6`. You must restart the server after changing the value of this configurable.	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`net.tcpsize`	Client, Server, Proxy	`512K`	TCP send and receive buffer sizes, set on connection. Consider increasing for high-latency connections, such as the Proxy. Actual buffer size is the larger of this value and that defined by the OS. (See also "net.autotune" on page 769)	After you change the value of this configurable, you must explicitly "stop" the server. **Note** `p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.

P

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`proxy.monitor.interval`	Proxy	`10`	Set the proxy monitoring interval. Default is 10 seconds.	No
`proxy.monitor.level`	Proxy	`0`	`0`: Monitoring disabled (default). `1`: Monitor file transfers only. `2`: Monitor all operations. `3`: Monitor all traffic for all operations.	No
`pull.trigger.dir`	Server	None	Temporary directory for alternative archive copy.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`push.unlocklocked`	Server	`0`	When set, unlock locked files if "p4 push" on page 427 fails.	No

R

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rcs.nofsync`	Server	`0`	Set to `1` to disable `fsync()` call when server writes to a versioned file in RCS format, and permit the OS to determine when to write the modified data.	No
`rejectList`	Server	none	Specifies one or more clients whose requests should be blocked. For more information, see "Blocking Clients" in *Helix Core Server Administrator Guide*.	No
`rpl.checksum.auto`	Server	`0`	Level of database table checksum verification to perform when rotating journal. Each level corresponds to a larger set of database tables. `0`: Disable checksums. `1`: Verify the most important system and revision tables. `2`: Verify all of level 1, plus tables that hold metadata that does not vary between replicas. `3`: Verify all metadata, including metadata that is expected to vary on build-farm and edge-server replicas.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rpl.checksum.change`	Server	`0`	Level of on-the-fly changelist verification to perform. `0`: Perform no verification. `1`: Write journal note at the end of a submit. `2`: Replica verifies changelist summary and writes to **`integrity.csv`** if the changelist does not match. `3`: Replica verifies changelist summary and writes to **`integrity.csv`** even if the changelist does match. Setting affects **`p4 submit`**, **`p4 push`**, **`p4 fetch`**, **`p4 populate`**, and **`p4 unzip`** commands.	No
`rpl.checksum.table`	Server	`0`	Level of table checksumming to perform. `0`: Perform table-level checksumming only. `1`: Journal notes for table-unload and table-scan are processed by the replica, and are logged to **`integrity.csv`** if the check fails. `2`: Results of journal note processing in the replica are logged even if the results match.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rpl.compress`	Server	`0`	Enable replica/master network compression: `0`: No data stream compression. `1`: Data streams used for archive transfer to the replica (`p4 pull` **-u**) are compressed. `2`: Data streams used by `p4 pull` **-u** and `p4 pull` are compressed. `3`: All data streams (`p4 pull` **-u**, `p4 pull`, and data streams for commands forwarded to the master or commit server) are compressed. `4`: Compress only the journal pull and journal copy connections between the replica and the master.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rpl.forward.login`	Server	`0`	Set to `1` on each replica to enable single-sign-on authentication for users in a distributed configuration. The `auth.id` configurable must also be the same for all servers participating in a distributed configuration. For more information, see "Authenticating users" in *Helix Core Server Administrator Guide*.	**On edge servers (not the master):** After you change the value of this configurable, you must explicitly "stop" the server. **Note**

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				`p4 admin restart` is not sufficient. For UNIX, see Stopping the Perforce Service and Starting the Perforce Service.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				For Windows, see Starting and stopping the Helix server.
`rpl.jnlwait.adjust`	Server	`25`	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.	No
`rpl.jnlwait.interval`	Server	`50`	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.	No
`rpl.jnlwait.max`	Server	`1000`	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rpl.journalcopy.location`	Server	`0`	Set to `0` means that the journalcopy thread writes the journal directly to the filepath with a prefix of the standby server's "journalPrefix" on page 756. Set to `1` means that the journalcopy thread writes the journal to where the standby server's "P4JOURNAL" on page 663 would be written. The journal is then rotated to the filepath with a prefix of the standby server's "journalPrefix" on page 756. Note that journals written by the journalcopy thread always have their journal number in the suffix. Changing this configurable takes effect when the standby server replicates the "P4TARGET" on page 683's journal rotation.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rpl.labels.global`	Server	`0`	A multi-site deployment supports both local and global labels. See "Introduction to federated services" in *Helix Core Server Administrator Guide*. **0 means local is the default** / **1 changes default to global** 0: a local label is restricted to the single edge server on which it is created and updated, and cannot be used on other servers 1: a global label is created and updated on the commit server, and visible to all servers The `-g` option of `p4 label`, `p4 labelsync`, and `p4 tag` specifies whether the label being applied is local to an edge server, or globally available from the commit server:	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

0 means local is the default	1 changes default to global
To create or update a global label on an edge server, use the `-g` option. For example, `p4 label -g myGlobalLabel`	The command `p4 configure set rpl.labels.global=1` means that any new label will be global. Existing local labels remain local. To update a local label, use the `-g` option. To convert a local label into a global label, use "p4 unload" on page 599, then "p4 reload" on page 436 to load the label onto the commit server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`rpl.replay.userrp`	Server	`0`	To replicate the `db.user.rp` table, set to `1` . The `db.user.rp` table contains information about users who have directly logged into and used a replica. It records the ticket that the replica issued to the user and the last time the user accessed the replica. This table is currently journaled, but it is not replicated by default. Typically you would not want to replicate this data. But there are times when this might be needed. For example, if you are chaining a read-only replica to another replica to provide a warm standby for failover, you might want your warm standby to replicate the `db.user.rp` table. This would disable the automatic filtering of `db.user.rp` records, and your replica would then replay (and re-journal) all the `db.user.rp` journal records it receives from its target.	No
`rpl.submit.nocopy`	Server	`0`	Disable default submit archive file copy	No
`rpl.verify.cache`	Server	`0`	If set, a replica server will re-verify the integrity of a cached file every time it delivers the file to the user, If the files do not match, it will re-fetch the file from the upstream server. This is computationally expensive on the replica and typically only useful in conjunction with Perforce technical support.	No
`run.users.authorize`	Server	`0`	If set to `1`, requires a user to authenticate before running `p4 users`.	No

S

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`security`	Server	`0`	Server security level: `0`: Legacy support: passwords not required, strength requirements unenforced. `1`: Strong passwords required, existing passwords not reset, compatible with pre-2003.2 client software. `2`: Strong passwords required, existing passwords reset, requires 2003.2 or higher client software. `3`: Passwords must be strong, and ticket-based authentication (`p4 login`) is required. `4`: All of the above restrictions. Also, authenticated service users must be used for all replica server and remote depot connections to this server. `5`: Requires that any intermediary (such as a proxy or broker) has a valid authenticated service user. `6`: Requires each intermediary to have a valid server spec, where the service user must match the user named in the User field of the spec. For details, see Server security levels in *Helix Core Server Administrator Guide*.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server`	Server	`1`	Server command logging level: `server=1` ensures that the start information for each command is logged `server=2` extends server tracing to include command start and stop `server=3` adds a "compute end" message and "Sync Network Estimates" for " p4 sync" on page 574 `server=4` adds errors sent to the client to the server log See also: ■ "Setting server trace and tracking flags" in Helix Core Server Administrator Guide ■ "P4LOG" on page 665 ■ the Support Knowledgebase article, "Helix server Trace Flags"	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.allowfetch`	Server	`0`	Determines whether changes can be fetched. ■ If set to `1`, this server can fetch from other servers. ■ If set to `2`, other servers can fetch from this server. ■ If set to `3`, both `1` and `2` are allowed.	No
`server.allowpush`	Server	`0`	Determines whether changes can be pushed. ■ If set to `1`, this server can push to other servers. ■ If set to `2`, other servers can push to this server. ■ If set to `3`, both `1` and `2` are allowed.	No
`server.allowremotelocking`	Server	`0`	DVCS configurations with files of type `+l` can use the `--remote` flag on the "p4 edit" on page 180, "p4 delete" on page 143, and "p4 revert" on page 481 commands. This locks `+l` file types exclusively on the shared server. The locks are released automatically when the modified files are pushed. The shared server must be a commit server and this configurable must have `1`, not `0`, as its value. To learn about `+l`, see the Helix Core Server User Guide on File type modifiers.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.allowrewrite`	Server	`0`	If set to a non-zero value, allows this server to run the `p4 unsubmit` and `p4 fetch -t` commands.	No
`server.commandlimits`	Server	`0`	Policy for per-command resource limits: `0`: All users can use command-line overrides for `MaxResults`, `MaxScanRows`, and `MaxLockTime` limits defined in the `p4 group` specs. `1`: Per-command options can specify lower, but not higher, resource limits. `2`: All command-line resource limit options are silently ignored.	No
`server.depot.root`	Server	none	The filesystem location with respect to which a relative address given in the `Map:` field of a depot form is evaluated. If it is not set, the `Map:` field relative address is evaluated with respect to the value stored in `P4ROOT`. For more information, see the `p4 depot` command.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.extensions.dir`	Server		Directory for Extension-owned storage	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.global.client.views`	Server	`0`	Controls whether the **view maps** (or **client maps**) of a non-stream client on an edge server are made global when a client is modified. **View maps** of a client on a replica must be made global if that client is to be used as a template on another server. This configurable can be set globally, or individually for each server. Setting this configurable can make client view maps global upon the subsequent client modification. Clearing this configurable does not delete the view maps of any clients, but does prevent subsequent changes to a client's view map from being propagated to other servers. If a client with global view maps is deleted, its view maps are also deleted globally regardless of the value of `server.global.client.views`.	No
`server.locks.archive`	Server	`1`	By default, "p4 archive" on page 67 and "p4 restore" on page 474 lock the global metadata while archiving or restoring revisions. To disable locking, set the value of this configurable to `0`.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.locks.dir`	Server	`server.locks`	Directory for server locks, specified relative to `P4ROOT`. To disable server locking, set this configurable to `disabled`. (If `db.peeking` is nonzero (enabled), `server.locks` cannot be `disabled`; you can disable locking by setting `server.locks.sync` to 0.)	No
`server.locks.global`	Server	`0`	Set this configurable to `1` to make `p4 lock` from an edge server take global locks on the commit server by default.	No
`server.locks.sync`	Server	`0`	When set, the `p4 sync` command takes a client workspace lock in shared mode. The default value of `0` prevents sync from taking a client workspace lock. If `db.peeking` is enabled, the `server.locks.dir` directory must exist. The changes to locking behavior that occur when you enable `db.peeking` obviate the need to set `server.locks.dir` to `disabled`, but if performance issues arise with respect to multiple concurrent, large, and/or interrupted `p4 sync` commands, you can obtain the old behavior for syncing by setting `server.locks.sync` to `0`.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.maxcommands`	Server	`0`	If monitoring is enabled, and if this configurable is set to a nonzero value, and the limit is exceeded: ▪ Helix Core server refuses to accept more than this many simultaneous command requests ▪ users receive the `TooManyCommands` error You must restart the server after changing the value of this configurable. See also "server.maxcommands.allow" on the next page.	

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`server.maxcommands.allow`	Server	`1`	Allow `super` and `operator` users access to a subset of commands even if `"server.maxcommands" on the previous page` is reached. For the list of these commands, see "Limiting simultaneous connections" in the *Helix Core Server Administrator Guide*. To disable, set to `0`. You must restart the server after changing the value of this configurable.	

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`serverlog.counter.`***`N`***	Server	none	The counter name for the structured log file designated by ***`n`***. (For example, if the structured log file is `errors.csv`, ***`n`*** is `3`.) See "Logging and structured files" in the *Helix Core Server Administrator Guide* for more information.	No
`serverlog.file.`***`N`***	Server	none	Server log file name associated with each structured log file. See `p4 logparse` for a list of valid filenames. *n* may not exceed 500.	No
`serverlog.maxmb.`***`N`***	Server	none	For each structured log file, the size, in megabytes, at which the associated log file is rotated.	No
`serverlog.retain.`***`N`***	Server	none	For each structured log file, the number of rotated log files to retain on the server at any one time.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`serverlog.version.`*`N`*	Server	the current server protocol level	The serverlog version matches a server protocol level and can be set to retain a level prior to an upgrade. See Protocol levels of the Helix server in *Helix Core Server Administrator Guide*.	No
`serviceUser`	Server	none	The service user as which a server (or proxy) authenticates against a master server in a replication/proxy configuration, or against a remote server in the context of remote depots.	No
`spec.hashbuckets`	Server	`99`	Number of buckets (subdirectories) into which files in the spec depot are hashed. Set to `0` to disable hashing, which may slow performance on older filesystems where performance is a function of the number of files per directory.	No
`ssl.secondary.suite`	Server	`0`	By default, Perforce's SSL support is based on the AES256-SHA cipher suite. To use CAMELLIA256-SHA, set this tunable to `1`.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

`ssl.tls.version.min`
and
`ssl.tls.version.max`
		`10` (min)
		and
		`12` (max)
			Controls the TLS version(s) allowed for SSL connections. The connection uses the highest version in common between server and client.
			
			`ssl.tls.version.min` specifies the lowest version, and `ssl.tls.version.max` specifies the highest version.
			
			Valid values are `10` for TLSv1.0, `11` for TLSv1.1, `12` for TLSv1.2, and `13` for TLSv1.3.
			
			To see the values on the server:
			
			`p4 configure show ssl.tls.version.min`
			`p4 configure show ssl.tls.version.max`
			
			To set the values on the server, issue commands, then **restart the server so the changes take effect**. For example, to allow TLSv1.1 or TLSv1.2, but exclude TLSv1.0:
			
			`p4 configure set ssl.tls.version.min=11`
			`p4 configure set ssl.tls.version.max=12`
			
			To allow only a single version, assign the same value to both configurables.
			
			On a client, to verify that TLSv1.0 does not connect:
			
			`p4 -v ssl.tls.version.min=10 -v ssl.tls.version.max=10 info`
			
			Tip
			TLS 1.3 is faster than TLS 1.2 at file transfers, but establishing a TLS 1.3 connection requires more overhead. (Note that the higher the latency, the less the connection overhead matters.)
			
			Applications that rely on many short-lived connections
				After you change the value of this configurable, you must explicitly "stop" the server.
				
				Note
				
				`p4`

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`startup.N`	Server	none	For replica servers, set `startup.1` through `startup.N` to be `p4 pull` threads to be spawned at startup. All servers with a configured `ServerID` can set `startup.1` through `startup.N` to be background tasks to be spawned at startup. The `startup.N` configurables are processed sequentially. Processing stops at the first gap in the numerical sequence. Any commands after a gap are ignored. **Tip** Prior to restarting the server, you can interactively invoke: ▪ "p4 bgtask" on page 72 ▪ "p4 cachepurge" on page 81 ▪ "p4 heartbeat" on page 251 ▪ "p4 pull" on page 420 to observe the behavior of associated options, as well as of configurables, such as "net.heartbeat.interval" on page 770 and "pull.trigger.dir" on page 792.	

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`statefile`	Server	`state`	For replica servers, the file used by the server to track the current journal position.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`submit.allowbgtransfer`	Server	`0`	To enable background archive transfer, set to `1` on ALL the servers that participate in your replicated environment. See: ■ "submit.autobgtransfer" on page 820 ■ "Background archive transfer for edge server submits" in *Helix Core Server Administrator Guide*.	If you use "p4 heartbeat" on page 251 in a "startup.N" on page 816 script,

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`submit.autobgtransfer`	Server	`0`	Assuming you have set "submit.allowbgtransfer" on page 818 on ALL the servers that participate in your replicated environment, to make "p4 submit" on page 558 function as if it were an alias for `p4 submit -b` on one or more servers, set to `1` on those servers. See "Background archive transfer for edge server submits" in *Helix Core Server Administrator Guide*.	

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?

`submit.collision.check`	Server	1	The default, `1`, prevents submitting a changelist that contains what the server perceives as a name collision between directory name and file name. For example:
			`$ p4 files ...`
			`//depot/foo/bar/myfile#1 - add change 10 (text)`
			
			`$ p4 add bar`
			`//depot/foo/bar#1 - opened for add`
			
			`$ p4 submit -d test Submitting change 11.`
			`Locking 1 files ...`
			`Cannot add file '//depot/foo/bar', filename collides with an existing directory path in the depot.`
			
			To remove the check, set this configurable to `0` and be aware of the risk of accidentally overwriting file content.
				No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`submit.identity`	Server	none	Enable the generation of global changelist ids. This is relevant for users of the Helix Core server's distributed versioning (DVCS) features. ▪ `uuid` generates the id in uuid format. ▪ `checksum` generates the id in checksum format. ▪ `serverid` generates the id in serverid+change format. For more information on global changelist ids, see the section "Track a changelist's identity from server to server" in the "Fetching and Pushing" chapter of *Using Helix Core Server for Distributed Versioning*.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`submit.noretransfer`	Server	`0`	Always re-transfer files after a failed submit. Set to `1` if you want the server to check whether files are already in the expected archive location and not re-transfer such files. You can override the set behavior by using the `--noretransfer` option to the `p4 submit` command.	After you change the value of this configurable, you must explicitl

Configurable	ClientorServerorProxy?	DefaultValue	Meaning	ServerRestartRequired?
				y "stop" the server. `p4 admin restart` is not sufficient. For UN

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				IX, see Stopping the Perforce Service and Starting the Perforce Servic

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				e. For Windows, see Starting and stopping the Helix server.

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`submit.unlocklocked`	Server	`0`	When set, open files that users have locked (with the `p4 lock` command) are automatically unlocked after a failed `p4 submit`.	No
`sys.rename.max`	Server and client	`10`	Limit the number to retries to rename a file if renaming fails. Affects Windows `Rename()` retry loop. Set on the server and each client.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`sys.rename.wait`	Server and client	`1000`	Timeout in microseconds between file rename attempts. Affects Windows **`Rename()`** retry loop. Set on the server and each client.	No

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`sys.threading.groups`	Server	`0`	If set to `1`, adds support for utilizing multiple processor groups on Windows Server 2008 R2 or later.	After you change the value of this configurable, you must explicitl

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				y "stop" the server. `p4 admin restart` is not sufficient. For UN

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				IX, see Stopping the Perforce Service and Starting the Perforce Servic

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
				e. For Windows, see Starting and stopping the Helix server.

T

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`template.client`	Server	none	Specifies the default client to be used as a template if the user omits the `-t` option on the `p4 client` command.	No
`template.label`	Server	none	Specifies the default label to be used as a template if the user omits the `-t` option on the `p4 label` command.	No
`track`	Server		Set by the server. See "Performance Tracking" under Diagnostic flags for monitoring the server in *Helix Core Server Administrator Guide*.	No
`triggers.io`	Server	`0`	If set, specifies that triggers will not receive their parameters via command line variables. Rather, they will receive a dictionary of key/value pairs sent to their STDIN. Triggers can use their dictionary response to reply to the server via STDOUT.	No

Z

Configurable	Client or Server or Proxy?	Default Value	Meaning	Server Restart Required?
`zerosyncPrefix`	Server	none	If set, changes default behavior of `p4 sync` such that if a client workspace begins with this prefix, all sync operations to affected workspaces assume `p4 sync` **-k**, and do not alter contents of the workspace.	No

Glossary

A

access level

A permission assigned to a user to control which commands the user can execute. See also the 'protections' entry in this glossary and the 'p4 protect' command in the P4 Command Reference.

admin access

An access level that gives the user permission to privileged commands, usually super privileges.

APC

The Alternative PHP Cache, a free, open, and robust framework for caching and optimizing PHP intermediate code.

archive

1. For replication, versioned files (as opposed to database metadata). 2. For the 'p4 archive' command, a special depot in which to copy the server data (versioned files and metadata).

atomic change transaction

Grouping operations affecting a number of files in a single transaction. If all operations in the transaction succeed, all the files are updated. If any operation in the transaction fails, none of the files are updated.

avatar

A visual representation of a Swarm user or group. Avatars are used in Swarm to show involvement in or ownership of projects, groups, changelists, reviews, comments, etc. See also the "Gravatar" entry in this glossary.

B

base

For files: The file revision, in conjunction with the source revision, used to help determine what integration changes should be applied to the target revision. For checked out streams: The public have version from which the checked out version is derived.

binary file type

A Helix server file type assigned to a non-text file. By default, the contents of each revision are stored in full, and file revision is stored in compressed format.

branch

(noun) A set of related files that exist at a specific location in the Perforce depot as a result of being copied to that location, as opposed to being added to that location. A group of related files is often referred to as a codeline. (verb) To create a codeline by copying another codeline with the 'p4 integrate', 'p4 copy', or 'p4 populate' command.

branch form

The form that appears when you use the 'p4 branch' command to create or modify a branch specification.

branch mapping

Specifies how a branch is to be created or integrated by defining the location, the files, and the exclusions of the original codeline and the target codeline. The branch mapping is used by the integration process to create and update branches.

branch view

A specification of the branching relationship between two codelines in the depot. Each branch view has a unique name and defines how files are mapped from the originating codeline to the target codeline. This is the same as branch mapping.

broker

Helix Broker, a server process that intercepts commands to the Helix server and is able to run scripts on the commands before sending them to the Helix server.

C

change review

The process of sending email to users who have registered their interest in changelists that include specified files in the depot.

changelist

A list of files, their version numbers, the changes made to the files, and a description of the changes made. A changelist is the basic unit of versioned work in Helix server. The changes specified in the changelist are not stored in the depot until the changelist is submitted to the depot. See also atomic change transaction and changelist number.

changelist form

The form that appears when you modify a changelist using the 'p4 change' command.

changelist number

An integer that identifies a changelist. Submitted changelist numbers are ordinal (increasing), but not necessarily consecutive. For example, 103, 105, 108, 109. A pending changelist number might be assigned a different value upon submission.

check in

To submit a file to the Helix server depot.

check out

To designate one or more files, or a stream, for edit.

checkpoint

A backup copy of the underlying metadata at a particular moment in time. A checkpoint can recreate db.user, db.protect, and other db.* files. See also metadata.

classic depot

A repository of Helix server files that is not streams-based. Uses the Perforce file revision model, not the graph model. The default depot name is depot. See also default depot, stream depot, and graph depot.

client form

The form you use to define a client workspace, such as with the 'p4 client' or 'p4 workspace' commands.

client name

A name that uniquely identifies the current client workspace. Client workspaces, labels, and branch specifications cannot share the same name.

client root

The topmost (root) directory of a client workspace. If two or more client workspaces are located on one machine, they should not share a client root directory.

client side

The right-hand side of a mapping within a client view, specifying where the corresponding depot files are located in the client workspace.

client workspace

Directories on your machine where you work on file revisions that are managed by Helix server. By default, this name is set to the name of the machine on which your client workspace is located, but it can be overridden. Client workspaces, labels, and branch specifications cannot share the same name.

code review

A process in Helix Swarm by which other developers can see your code, provide feedback, and approve or reject your changes.

codeline

A set of files that evolve collectively. One codeline can be branched from another, allowing each set of files to evolve separately.

comment

Feedback provided in Helix Swarm on a changelist, review, job, or a file within a changelist or review.

commit server

A server that is part of an edge/commit system that processes submitted files (checkins), global workspaces, and promoted shelves.

conflict

1. A situation where two users open the same file for edit. One user submits the file, after which the other user cannot submit unless the file is resolved. 2. A resolve where the same line is changed when merging one file into another. This type of conflict occurs when the comparison of two files to a base yields different results, indicating that the files have been changed in different ways. In this case, the merge cannot be done automatically and must be resolved manually. See file conflict.

copy up

A Helix server best practice to copy (and not merge) changes from less stable lines to more stable lines. See also merge.

counter

A numeric variable used to track variables such as changelists, checkpoints, and reviews.

CSRF

Cross-Site Request Forgery, a form of web-based attack that exploits the trust that a site has in a user's web browser.

D

default changelist

The changelist used by a file add, edit, or delete, unless a numbered changelist is specified. A default pending changelist is created automatically when a file is opened for edit.

deleted file

In Helix server, a file with its head revision marked as deleted. Older revisions of the file are still available. in Helix server, a deleted file is simply another revision of the file.

delta

The differences between two files.

depot

A file repository hosted on the server. A depot is the top-level unit of storage for versioned files (depot files or source files) within a Helix Core server. It contains all versions of all files ever submitted to the depot. There can be multiple depots on a single installation.

depot root

The topmost (root) directory for a depot.

depot side

The left side of any client view mapping, specifying the location of files in a depot.

depot syntax

Helix server syntax for specifying the location of files in the depot. Depot syntax begins with: //depot/

diff

(noun) A set of lines that do not match when two files, or stream versions, are compared. A conflict is a pair of unequal diffs between each of two files and a base, or between two versions of a stream. (verb) To compare the contents of files or file revisions, or of stream versions. See also conflict.

donor file

The file from which changes are taken when propagating changes from one file to another.

E

edge server

A replica server that is part of an edge/commit system that is able to process most read/write commands, including 'p4 integrate', and also deliver versioned files (depot files).

exclusionary access

A permission that denies access to the specified files.

exclusionary mapping

A view mapping that excludes specific files or directories.

extension

Similar to a trigger, but more modern. See "Helix Core Server Administrator Guide" on "Extensions".

F

file conflict

In a three-way file merge, a situation in which two revisions of a file differ from each other and from their base file. Also, an attempt to submit a file that is not an edit of the head revision of the file in the depot, which typically occurs when another user opens the file for edit after you have opened the file for edit.

file pattern

Helix server command line syntax that enables you to specify files using wildcards.

file repository

The master copy of all files, which is shared by all users. In Helix server, this is called the depot.

file revision

A specific version of a file within the depot. Each revision is assigned a number, in sequence. Any revision can be accessed in the depot by its revision number, preceded by a pound sign (#), for example testfile#3.

file tree

All the subdirectories and files under a given root directory.

file type

An attribute that determines how Helix server stores and diffs a particular file. Examples of file types are text and binary.

fix

A job that has been closed in a changelist.

form

A screen displayed by certain Helix server commands. For example, you use the change form to enter comments about a particular changelist to verify the affected files.

forwarding replica

A replica server that can process read-only commands and deliver versioned files (depot files). One or more replicate servers can significantly improve performance by offloading some of the master server load. In many cases, a forwarding replica can become a disaster recovery server.

G

Git Fusion

A Perforce product that integrates Git with Helix, offering enterprise-ready Git repository management, and workflows that allow Git and Helix server users to collaborate on the same projects using their preferred tools.

graph depot

A depot of type graph that is used to store Git repos in the Helix server. See also Helix4Git and classic depot.

group

A feature in Helix server that makes it easier to manage permissions for multiple users.

H

have list

The list of file revisions currently in the client workspace.

head revision

The most recent revision of a file within the depot. Because file revisions are numbered sequentially, this revision is the highest-numbered revision of that file.

heartbeat

A process that allows one server to monitor another server, such as a standby server monitoring the master server (see the p4 heartbeat command).

Helix server

The Helix server depot and metadata; also, the program that manages the depot and metadata, also called Helix Core server.

Helix TeamHub

A Perforce management platform for code and artifact repository. TeamHub offers built-in support for Git, SVN, Mercurial, Maven, and more.

Helix4Git

Perforce solution for teams using Git. Helix4Git offers both speed and scalability and supports hybrid environments consisting of Git repositories and 'classic' Helix server depots.

hybrid workspace

A workspace that maps to files stored in a depot of the classic Perforce file revision model as well as to files stored in a repo of the graph model associated with git.

I

iconv

A PHP extension that performs character set conversion, and is an interface to the GNU libiconv library.

integrate

To compare two sets of files (for example, two codeline branches) and determine which changes in one set apply to the other, determine if the changes have already been propagated, and propagate any outstanding changes from one set to another.

J

job

A user-defined unit of work tracked by Helix server. The job template determines what information is tracked. The template can be modified by the Helix server system administrator. A job describes work to be done, such as a bug fix. Associating a job with a changelist records which changes fixed the bug.

job daemon

A program that checks the Helix server machine daily to determine if any jobs are open. If so, the daemon sends an email message to interested users, informing them the number of jobs in each category, the severity of each job, and more.

job specification

A form describing the fields and possible values for each job stored in the Helix server machine.

job view

A syntax used for searching Helix server jobs.

journal

A file containing a record of every change made to the Helix server's metadata since the time of the last checkpoint. This file grows as each Helix server transaction is logged. The file should be automatically truncated and renamed into a numbered journal when a checkpoint is taken.

journal rotation

The process of renaming the current journal to a numbered journal file.

journaling

The process of recording changes made to the Helix server's metadata.

L

label

A named list of user-specified file revisions.

label view

The view that specifies which filenames in the depot can be stored in a particular label.

lazy copy

A method used by Helix server to make internal copies of files without duplicating file content in the depot. A lazy copy points to the original versioned file (depot file). Lazy copies minimize the consumption of disk space by storing references to the original file instead of copies of the file.

license file

A file that ensures that the number of Helix server users on your site does not exceed the number for which you have paid.

list access

A protection level that enables you to run reporting commands but prevents access to the contents of files.

local depot

Any depot located on the currently specified Helix server.

local syntax

The syntax for specifying a filename that is specific to an operating system.

lock

1. A file lock that prevents other clients from submitting the locked file. Files are unlocked with the 'p4 unlock' command or by submitting the changelist that contains the locked file. 2. A database lock that prevents another process from modifying the database db.* file.

log

Error output from the Helix server. To specify a log file, set the P4LOG environment variable or use the p4d -L flag when starting the service.

M

mapping

A single line in a view, consisting of a left side and a right side that specify the correspondences between files in the depot and files in a client, label, or branch. See also workspace view, branch view, and label view.

MDS checksum

The method used by Helix server to verify the integrity of versioned files (depot files).

merge

1. To create new files from existing files, preserving their ancestry (branching). 2. To propagate changes from one set of files to another. 3. The process of combining the contents of two conflicting file revisions into a single file, typically using a merge tool like P4Merge.

merge file

A file generated by the Helix server from two conflicting file revisions.

metadata

The data stored by the Helix server that describes the files in the depot, the current state of client workspaces, protections, users, labels, and branches. Metadata is stored in the Perforce database and is separate from the archive files that users submit.

modification time or modtime

The time a file was last changed.

MPM

Multi-Processing Module, a component of the Apache web server that is responsible for binding to network ports, accepting requests, and dispatch operations to handle the request.

N

nonexistent revision

A completely empty revision of any file. Syncing to a nonexistent revision of a file removes it from your workspace. An empty file revision created by deleting a file and the #none revision specifier are examples of nonexistent file revisions.

numbered changelist

A pending changelist to which Helix server has assigned a number.

O

opened file

A file you have checked out in your client workspace as a result of a Helix Core server operation (such as an edit, add, delete, integrate). Opening a file from your operating system file browser is not tracked by Helix Core server.

owner

The Helix server user who created a particular client, branch, or label.

P

p4

1. The Helix Core server command line program. 2. The command you issue to execute commands from the operating system command line.

p4d

The program that runs the Helix server; p4d manages depot files and metadata.

P4PHP

The PHP interface to the Helix API, which enables you to write PHP code that interacts with a Helix server machine.

PECL

PHP Extension Community Library, a library of extensions that can be added to PHP to improve and extend its functionality.

pending changelist

A changelist that has not been submitted.

Perforce

Perforce Software, Inc., a leading provider of enterprise-scale software solutions to technology developers and development operations (“DevOps”) teams requiring productivity, visibility, and scale during all phases of the development lifecycle.

project

In Helix Swarm, a group of Helix server users who are working together on a specific codebase, defined by one or more branches of code, along with options for a job filter, automated test integration, and automated deployment.

protections

The permissions stored in the Helix server’s protections table.

proxy server

A Helix server that stores versioned files. A proxy server does not perform any commands. It serves versioned files to Helix server clients.

R

RCS format

Revision Control System format. Used for storing revisions of text files in versioned files (depot files). RCS format uses reverse delta encoding for file storage. Helix server uses RCS format to store text files. See also reverse delta storage.

read access

A protection level that enables you to read the contents of files managed by Helix server but not make any changes.

remote depot

A depot located on another Helix server accessed by the current Helix server.

replica

A Helix server that contains a full or partial copy of metadata from a master Helix server. Replica servers are typically updated every second to stay synchronized with the master server.

repo

A graph depot contains one or more repos, and each repo contains files from Git users.

reresolve

The process of resolving a file after the file is resolved and before it is submitted.

resolve

The process you use to manage the differences between two revisions of a file, or two versions of a stream. You can choose to resolve file conflicts by selecting the source or target file to be submitted, by merging the contents of conflicting files, or by making additional changes. To resolve stream conflicts, you can choose to accept the public source, accept the checked out target, manually accept changes, or combine path fields of the public and checked out version while accepting all other changes made in the checked out version.

reverse delta storage

The method that Helix server uses to store revisions of text files. Helix server stores the changes between each revision and its previous revision, plus the full text of the head revision.

revert

To discard the changes you have made to a file in the client workspace before a submit.

review access

A special protections level that includes read and list accesses and grants permission to run the p4 review command.

review daemon

A program that periodically checks the Helix server machine to determine if any changelists have been submitted. If so, the daemon sends an email message to users who have subscribed to any of the files included in those changelists, informing them of changes in files they are interested in.

revision number

A number indicating which revision of the file is being referred to, typically designated with a pound sign (#).

revision range

A range of revision numbers for a specified file, specified as the low and high end of the range. For example, myfile#5,7 specifies revisions 5 through 7 of myfile.

revision specification

A suffix to a filename that specifies a particular revision of that file. Revision specifiers can be revision numbers, a revision range, change numbers, label names, date/time specifications, or client names.

RPM

RPM Package Manager. A tool, and package format, for managing the installation, updates, and removal of software packages for Linux distributions such as Red Hat Enterprise Linux, the Fedora Project, and the CentOS Project.

S

server data

The combination of server metadata (the Helix server database) and the depot files (your organization's versioned source code and binary assets).

server root

The topmost directory in which p4d stores its metadata (db.* files) and all versioned files (depot files or source files). To specify the server root, set the P4ROOT environment variable or use the p4d -r flag.

service

In the Helix Core server, the shared versioning service that responds to requests from Helix server client applications. The Helix server (p4d) maintains depot files and metadata describing the files and also tracks the state of client workspaces.

shelve

The process of temporarily storing files in the Helix server without checking in a changelist.

status

For a changelist, a value that indicates whether the changelist is new, pending, or submitted. For a job, a value that indicates whether the job is open, closed, or suspended. You can customize job statuses. For the 'p4 status' command, by default the files opened and the files that need to be reconciled.

storage record

An entry within the db.storage table to track references to an archive file.

stream

A "branch" with built-in rules that determines what changes should be propagated and in what order they should be propagated.

stream depot

A depot used with streams and stream clients. Has structured branching, unlike the free-form branching of a "classic" depot. Uses the Perforce file revision model, not the graph model. See also classic depot and graph depot.

submit

To send a pending changelist into the Helix server depot for processing.

super access

An access level that gives the user permission to run every Helix server command, including commands that set protections, install triggers, or shut down the service for maintenance.

symlink file type

A Helix server file type assigned to symbolic links. On platforms that do not support symbolic links, symlink files appear as small text files.

sync

To copy a file revision (or set of file revisions) from the Helix server depot to a client workspace.

T

target file

The file that receives the changes from the donor file when you integrate changes between two codelines.

text file type

Helix server file type assigned to a file that contains only ASCII text, including Unicode text. See also binary file type.

theirs

The revision in the depot with which the client file (your file) is merged when you resolve a file conflict. When you are working with branched files, theirs is the donor file.

three-way merge

The process of combining three file revisions. During a three-way merge, you can identify where conflicting changes have occurred and specify how you want to resolve the conflicts.

trigger

A script that is automatically invoked by Helix server when various conditions are met. (See "Helix Core Server Administrator Guide" on "Triggers".)

two-way merge

The process of combining two file revisions. In a two-way merge, you can see differences between the files.

typemap

A table in Helix server in which you assign file types to files.

U

user

The identifier that Helix server uses to determine who is performing an operation. The three types of users are standard, service, and operator.

V

versioned file

Source files stored in the Helix server depot, including one or more revisions. Also known as an archive file. Versioned files typically use the naming convention 'filenamev' or '1.changelist.gz'.

view

A description of the relationship between two sets of files. See workspace view, label view, branch view.

W

wildcard

A special character used to match other characters in strings. The following wildcards are available in Helix server: * matches anything except a slash; ... matches anything including slashes; %%0 through %%9 is used for parameter substitution in views.

workspace

See client workspace.

workspace view

A set of mappings that specifies the correspondence between file locations in the depot and the client workspace.

write access

A protection level that enables you to run commands that alter the contents of files in the depot. Write access includes read and list accesses.

X

XSS

Cross-Site Scripting, a form of web-based attack that injects malicious code into a user's web browser.

Y

yours

The edited version of a file in your client workspace when you resolve a file. Also, the target file when you integrate a branched file.

License Statements

To get a listing of the third-party software licenses that Helix Core server uses, at the command line, type the `p4 help legal` command.

To get a listing of the third-party software licenses that the local client (such as P4V) uses, at the command line, type the `p4 help -l legal` command.