Perforce 2010.2 Command Reference

December 2010

Table of Contents

About This Manual

Synopsis

This is the *Perforce 2010.2 Command Reference*.

Description

This manual documents every Perforce command and environment variable. This manual is intended for users who prefer to learn by means of UNIX-style man pages, and for users who already understand the basics of Perforce and need to quickly find information on a specific command.

The following table provides an index to the *Command Reference* by functional area:

Function	Where to look
Help	`p4 help`, `p4 info`, *File Specifications*, *Views*, *Global Options*, *File Types*
Client workspace	`p4 client`, `p4 clients`, `p4 flush`, `p4 have`, `p4 sync`, `p4 where`, `p4 workspace`, `p4 workspaces`
Files	`p4 add`, `p4 copy`, `p4 delete`, `p4 diff`, `p4 diff2`, `p4 dirs`, `p4 edit`, `p4 files`, `p4 fstat`, `p4 grep`, `p4 move`, `p4 lock`, `p4 print`, `p4 rename`, `p4 revert`, `p4 sizes`, `p4 unlock`
Changelists	`p4 change`, `p4 changelist`, `p4 changes`, `p4 changelists`, `p4 describe`, `p4 filelog`, `p4 opened`, `p4 reopen`, `p4 review`, `p4 shelve`, `p4 submit`, `p4 unshelve`
Jobs	`p4 fix`, `p4 fixes`, `p4 job`, `p4 jobs`, `p4 jobspec`
Branching and Merging	`p4 branch`, `p4 branches`, `p4 integrate`, `p4 integrated`, `p4 label`, `p4 labels`, `p4 labelsync`, `p4 tag`, `p4 resolve`, `p4 resolved`
Administration	`p4 admin`, `p4 archive`, `p4 configure`, `p4 counter`, `p4 counters`, `p4 dbschema`, `p4 dbstat`, `p4 depot`, `p4 depots`, `p4 license`, `p4 lockstat`, `p4 logger`, `p4 logstat`, `p4 logtail`, `p4 monitor`, `p4 obliterate`, `p4 ping`, `p4 pull`, `p4 replicate`, `p4 restore`, `p4 reviews`, `p4 triggers`, `p4 typemap`, `p4 verify`

Function	Where to look
Security	`p4 group`, `p4 groups`, `p4 login`, `p4 logout`, `p4 passwd`, `p4 protect`, `p4 protects`, `p4 tickets`, `p4 user`, `p4 users`, `P4CLIENTPATH`
Environment	`p4 browse`, `p4 set`, *Environment and Registry Variables*, `P4AUDIT`, `P4AUTH`, `P4BROKEROPTIONS`, `P4CHANGE`, `P4CHARSET`, `P4COMMANDCHARSET`, `P4CLIENT`, `P4CONFIG`, `P4DEBUG`, `P4DESCRIPTION`, `P4DIFF`, `P4DIFFUNICODE`, `P4EDITOR`, `P4HOST`, `P4JOURNAL`, `P4LANGUAGE`, `P4LOG`, `P4MERGE`, `P4MERGEUNICODE`, `P4NAME`, `P4PAGER`, `P4PASSWD`, `P4PCACHE`, `P4PFSIZE`, `P4POPTIONS`, `P4PORT`, `P4ROOT`, `P4TARGET`, `P4TICKETS`, `P4USER`, `P4ZEROCONF`, `PWD`, `TMP`, `TEMP`

If you'd prefer to learn the concepts on which Perforce is based, or you prefer a style featuring more examples and tutorials than what you find here, see the *P4 User's Guide*, available from our web site at: `http://www.perforce.com`.

Options

This manual is available in PDF and HTML.

Usage Notes

Both the PDF and HTML versions of this manual have been extensively cross-referenced. When viewing the PDF manual online, you can read the description of any particular command by clicking on a reference to that command from any other chapter.

If there's anything we've left out that you think should be included, let us know. Please send your comments to `manual@perforce.com`.

p4 add

Synopsis

Open file(s) in a client workspace for addition to the depot.

Syntax

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

Description

`p4 add` opens files within the client workspace for addition to the depot. The specified file(s) are linked to a changelist; the files are not actually added to the depot until the changelist is sent to the server 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.

To open a file with `p4 add`, the file must exist in your client workspace *view*, but does not need to exist in your workspace at the time of `p4 add`. The file must, however, exist in your workspace when you run `p4 submit`, or the submission will fail. `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` flag. (To move files from the default changelist to a numbered changelist, use the `p4 change` command.)

When adding files, Perforce first examines the typemap table (`p4 typemap`) 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. If a match is *not* found, Perforce examines the first bytes of the file based on the `filesys.binaryscan` configurable (by default, 8192 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.

The `-t filetype` flag explicitly specifies a file type, overriding both the typemap table and Perforce's default file type detection mechanism.

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

If you open a file for edit or move/add, and another 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` flag ("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, the Perforce 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` flag, 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, the Perforce 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 flag is not used, the files are linked to the default changelist.
`-d`	Downgrade file open status to simple add.
`-f`	Use the `-f` flag to force inclusion of wildcards in filenames. See the *File Specifications* chapter for details.
`-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. Please see the *File Types* chapter for a list of Perforce file types.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- *Wildcards* in file specifications provided to `p4 add` are expanded by the local operating system, not by the Perforce server. For instance, the `...` wildcard cannot be used with `p4 add`.
- In Perforce, 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`, Perforce 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 all `*.c` files in the user's `~/src` directory for `add`; also opens the `README` file in the user's current working 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 Perforce 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 300.

Related Commands

To open a file for edit	`p4 edit`
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 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 admin

Synopsis

Perform administrative operations on the server.

Syntax

```
p4 [g-opts] admin checkpoint [ -z ] [ prefix ]
p4 [g-opts] admin journal [ -z ] [ prefix ]
p4 [g-opts] admin stop
p4 [g-opts] admin updatespecdepot [ -a | -s type ]
```

Description

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

To stop the server, use `p4 admin stop`. This locks the database to ensure that it is in a consistent state upon server restart, and then shuts down the Perforce background process. (For Windows users, this works whether you are running Perforce as a server or a service.)

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. 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 the *System Administrator's Guide*. The files are created in the server root specified when the Perforce server was started.

The `p4 admin updatespecdepot` command causes the Perforce Server to archive stored forms (specifically, `client`, `depot`, `branch`, `label`, `typemap`, `group`, `user`, and `job` forms) into the spec depot. If the `-a` flag is used, all of the form specification types are archived. If the `-s` flag option is used, then only those of the specified *type* are archived. Only those forms that have not yet been archived are created.

Options

`-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.
`-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`, `branch`, `label`, `typemap`, `group`, `user`, or `job`.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- Because `p4 admin stop` shuts down the Perforce server, you may see an error message indicating that the connection between the client and server 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.
- For more about administering Perforce, see the *Perforce System Administrator's Guide*.

Examples

`p4 admin stop`	Stop the Perforce server
`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 annotate

Synopsis

Print file lines along with their revisions.

Syntax

```
p4 [g-opts] annotate [ -a -c -i -I -q -dflag ] file[revRange] ...
```

Description

The `p4 annotate` command displays the revision number for each line of a revision (or range of revisions) of a file (or files). You can then run `p4 filelog` on the indicated revision(s) to find out who made each change, when, and why.

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, and *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.

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.
`-dflags`	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the *Usage Notes* below for a listing of these flags.

`-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, and so on. The use of the `-I` option implies the `-c` option.
`-i`	Follow file history across branches. If a file was created by branching, Perforce includes revisions up to the branch point. The use of the `-i` option implies the `-c` option.
`-q`	Quiet mode; suppress the one-line header for each file.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- The output of `p4 annotate` is highly amenable to scripting or other forms of automated processing.
- By default, `p4 annotate` ignores changes to text files over 10 MB in length. Perforce superusers can override this limit by setting the `dm.annotate.maxsize` configurable.
- The diff flags supported by `p4 annotate` are:

Flag	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

Synopsis

Archive obsolete revisions to an archive depot.

Syntax

`p4 [g-opts] archive [ -n -h -p -q ] -D depot file[revRange]...`

Description

Moves 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 (for example, `p4 sync`, `p4 diff`, and so on) skip `archive` revisions, but commands that do not require access to file content (such as `p4 filelog`, for example) continue to report to metadata concerning the archived revisions.

Use `p4 archive -p` ***with caution.*** This is the one of only two commands (along with `p4 obliterate`) in Perforce that actually remove file data.

Options

`-n`	Do not archive revisions; report on which revisions would have been archived.
`-h`	Do not archive head revisions.
`-p`	Purge any archives of the specified files named in the archive depot. (The action for affected revisions is set to `purge` on completion. File contents are no longer accessible from `p4 restore`.)
`-q`	Quiet mode; suppress messages about skipped revisions.
`-D depot`	Specify an archive depot to which files are to be archived.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- Only files stored in full (`+F`) or compressed (`+C`) may be archived. The files must be in a `local` depot (not a `remote` or another `archive` depot), and must neither be copied nor branched to (or from) another revision.

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

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 branch

Synopsis

Create or edit a branch mapping and its view.

Syntax

```
p4 [g-opts] branch [ -f ] branchspec
p4 [g-opts] branch -o branchspec
p4 [g-opts] branch -d [ -f ] branchspec
p4 [g-opts] branch -i [ -f ]
```

Description

`p4 branch` enables you to construct 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.

Once you have named and created a branch mapping, integrate files by typing `p4 integrate -b branchname`; the branch mapping automatically maps all *`toFiles`* to their corresponding *`fromFiles`*.

Saving a `p4 branch` form has no immediate effect on any files in the depot or your client workspace; you must call `p4 integrate -b branchspecname` to create the branched files in your workspace and to open the files in a changelist.

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`.
`Access:`	read-only	The date the branch mapping was last accessed.
`Update:`	read-only	The date the branch mapping was last changed.

Field Name	Type	Description
`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 flag.
`-f`	Force flag. Combined with `-d`, allows Perforce administrators to delete branches they don't own. Also allows administrators to change the modification date of the branch mapping (the `Update:` field is writable when using the `-f` flag).
`-i`	Read the branch mapping from standard input without invoking an editor.
`-o`	Write the branch mapping to standard output without invoking an editor.
`g-opts`	See the *Global Options* section.

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. See the *Views* chapter for more information on specifying views.

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

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

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

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

Synopsis

List existing branch mappings.

Syntax

```
p4 [g-opts] [ -u user ] [ -e namefilter -m max ] branches
```

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 namefilter` option to limit the output to branches whose name matches the `namefilter` pattern.

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

Options

`-m max`	List only the first `max` branch mappings.
`-e namefilter`	List only branches matching `namefilter`.
`-u user`	List only branches owned by `user`.
`g-opts`	See the *Global Options* section.

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 create or edit a branch mapping	`p4 branch`

p4 browse

Synopsis

Browse for a list of Zeroconf-registered Perforce servers.

Syntax

```
p4 [g-opts] browse
```

Description

Lists all Perforce servers that have registered with Zeroconf.

Options

g-opts	See the *Global Options* section.

Usage Notes

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

p4 change

Synopsis

Create or edit a changelist specification.

Syntax

```
p4 [g-opts] change [ -s ] [ -f | -u ] [ changelist# ]
p4 [g-opts] change -d [ -f -s ] changelist#
p4 [g-opts] change -o [ -s ] [ -f ] [ changelist# ]
p4 [g-opts] change -i [ -s ] [ -f | -u ]
p4 [g-opts] change -t restricted|public [ -f | -u ] changelist#
```

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.

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

Changelist numbers are assigned in sequence; Perforce may renumber changelists automatically on submission in order to keep the numeric order of submitted changelists identical to the chronological order.

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

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

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

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.
`Client:`	Read-only	Name of current client workspace.

Field Name	Type	Description
`Date:`	Read-only	Date the changelist was last modified.
`User:`	Read-only	Name of current Perforce user. 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.
`Status:`	Read-only	`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`, and `submitted` when its contents have been stored in the depot with `p4 submit`.
`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.
`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.
`Type:`	Writable, optional	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	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.

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` flag.
`-f`	Force flag. 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` flag. The `-u` and the `-f` flags are mutually exclusive.
`-f -d`	Forcibly delete a previously submitted changelist. Only a Perforce administrator or superuser can use this command, and the changelist must have had all of its files removed from the system with `p4 obliterate`.
`-o`	Write a changelist description to standard output.
`-i`	Read a changelist description from standard input. Input must be in the same format used by the `p4 change` form.
`-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.
`-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` flags are mutually exclusive.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- 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.

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`
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 changelists

Synopsis

List submitted and pending changelists.

Syntax

```
p4 [g-opts] changelists [-i -t -l -L -c client -m max -s status -u user]
[file[RevRange]...]
```

Description

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

p4 changelist

Synopsis

Create or edit a changelist specification.

Syntax

```
p4 [g-opts] changelist [ -f -s ] [changelist#]
p4 [g-opts] changelist -d [ -f -s ] changelist#
p4 [g-opts] changelist -o [ -s ] [changelist#]
p4 [g-opts] changelist -i [ -f -s ]
```

Description

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

p4 changes

Synopsis

List submitted and pending changelists.

Syntax

`p4 [g-opts] changes [-i -t -l -L -c client -m max -s status -u user] [file[RevRange]...]`

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 date hh: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` flag for the first 250 characters, or the `-l` flag 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. Including a revision range lists all changes that affect files within the range; providing a single revision specifier lists all changes from 1 to the specified revision.

Use the `-c client` and `-u user` flags to limit output to only those changelists made from the named client workspace or the named user.

Use the `-s status` flag to limit output to only those changelists with the provided *status* (`pending`, `shelved`, or `submitted`) value.

You can combine flags and file patterns to substantially limit the changelists that are displayed. You can also use the `-m max` flag to further limit output to *max* changes.

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

Options

`-i`	Include changelists that affected files that were integrated with the specified files.
`-t`	Display the time as well as the date of each change.
`-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.
`-c` *`client`*	List only changes made from the named client workspace.
`-m` *`max`*	List only the highest numbered *`max`* changes.
`-s` *`status`*	Limit the list to the changelists with the given status (`pending` or `submitted`)
`-u` *`user`*	List only changes made from the named user.
`g-opts`	See the *Global Options* section.

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.

Examples

`p4 changes -m 5 //depot/project/...`	Show the last five submitted changelists that include any file under the `project` directory
`p4 changes -m 5 -c eds_elm`	Show the last five submitted changelists from client workspace `eds_elm`.

`p4 changes -m 5 -u edk`	Show the last five submitted changelists from user `edk`.
`p4 changes file.c@2000/05/01,2000/06/01`	Show any changelists that include file `file.c`, as mapped to the depot through the client view, during the month of May 2000.
`p4 changes -m 1 -s submitted`	Output a single line showing the changelist number of the last submitted changelist.
`p4 changes @2001/04/01,@now`	Display all changelists submitted from April 1, 2001 to the present.
`p4 changes @2001/04/01`	Display all changelists submitted *before* April 1, 2000.

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 client

Synopsis

Create or edit a client workspace specification and its view.

Syntax

```
p4 [g-opts] client [-f -t template] [clientname]
p4 [g-opts] client -o [-t template] [clientname]
p4 [g-opts] client -d [-f] clientname
p4 [g-opts] client -i [-f]
```

Description

A Perforce client workspace is a set of files on a user's machine that mirror a subset of the files in the depot. The `p4 client` command is used to create or edit a client workspace specification; invoking this command displays a form in which the user enters the information required by Perforce to maintain the client workspace.

Although there is always a one-to-one mapping between a client workspace file and a depot file, these files do not need to be stored at the same relative locations, nor must they have the same names. The *client view*, which is specified in the `p4 client` form's `View:` field, specifies how files in the client workspace are mapped to the depot, and vice-versa.

When called without a *`clientname`* argument, `p4 client` operates on the client workspace specified by the `P4CLIENT` environment variable or one of its equivalents. If called with a *`clientname`* argument on a `locked` client, the client specification is read-only.

When `p4 client` completes, the new or altered client workspace specification is stored within the Perforce database; the files in the client workspace are not touched. The new client view doesn't take effect until the next `p4 sync`.

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

Form Fields

Field Name	Type	Description
`Client:`	Read-only	The client workspace name, as specified in the `P4CLIENT` environment variable or its equivalents.
`Owner:`	Writable	The Perforce user name of the user who owns the client workspace. The default is the user who created the client workspace.
`Update:`	Read-only	The date the client workspace specification was last modified.

Field Name	Type	Description
`Access:`	Read-only	The date and time that any part of the client workspace specification was last accessed by any Perforce command.
`Host:`	Writable, optional	The name of the host machine on which this client workspace resides. If included, operations on this client workspace can be run *only* from this host. The hostname must be provided exactly as it appears in the output of `p4 info` when run from that host. This field is meant to prevent accidental misuse of client workspaces on the wrong machine. It doesn't provide security, since the actual value of the host name can be overridden with the `-H` flag 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 client 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.
`AltRoots:`	Writable, optional	Up to two optional alternate client workspace roots. Perforce client programs use the first of the main and alternate roots to match the client program's current working directory. This enables users to use the same Perforce client specification on multiple platforms with different directory naming conventions. If you are using a Windows directory in any of your client roots, you must specify the Windows directory as your main client root and specify your other client root directories 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.

Field Name	Type	Description
`Options:`	Writable, mandatory	A set of seven switches that control particular client options. See the *Usage Notes*, below, for a listing of these options.
`SubmitOptions:`	Writable, mandatory	Flags 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 Perforce. • `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. • `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. • `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.
`LineEnd:`	Writable, mandatory	A set of four switches that control carriage-return/linefeed (CR/LF) conversion. See the *Usage Notes*, below, for a listing of these options.
`View:`	Writable, multi-line	Specifies the mappings between files in the depot and files in the client workspace. See *Views* for more information.

Options

`-t clientname`	Copy client workspace *`clientname`*'s view and client options into the `View:` and `Options:` field of this client workspace. (i.e, use *`clientname`*'s `View:` as a template)
`-f`	Allows the last modification date, which is normally read-only, to be set. Administrators can use this flag to delete or modify locked workspaces owned by other users.
`-d clientname`	Delete the specified client workspace if it is unlocked, whether or not the client is owned by the user. (The `-f` flag allows Perforce administrators to delete locked client workspaces that they don't own.)
`-i`	Read the client description from standard input.
`-o`	Write the client specification to standard output.
`g-opts`	See the *Global Options* section.

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 client workspace names are translated to underscores. For example, typing the command `p4 client "my client"` creates a client workspace called `my_client`.
- The `Options:` field contains six values, separated by spaces. Each of the six options have two possible settings; the following table provides the option values and their meanings:

Option	Choice	Default
`[no]allwrite`	If set, unopened files on the client are left writable. If `allwrite` is set, this option overrides `noclobber`.	`noallwrite`
`[no]clobber`	If set, a `p4 sync` overwrites ("clobbers") writable-but-unopened files in the client that have the same name as the newly-synced files. If `allwrite` is set, the `noclobber` option is ignored.	`noclobber`

Option	Choice	Default
`[no]compress`	If set, the data stream between the client and the server is compressed. (Both client and server must be version 99.1 or higher, or this setting is ignored.)	`nocompress`
`[no]crlf`	***Note***: *2000.2 or earlier only!* On Windows, if `crlf` is set, CR/LF translation is performed automatically when copying files between the depot and the client workspace.	`crlf`
`[un]locked`	Grant or deny other users permission to edit or delete the client specification (To make a `locked` client specification truly effective, you should also set the workspace's owner's password with `p4 passwd`.) If `locked`, only the owner is able to use or edit the client spec. Perforce administrators can override the lock by using the `-f` (force) flag with `p4 client`.	`unlocked`
`[no]modtime`	For files *without* the `+m` (modtime) file type modifier: • For Perforce clients at the 99.2 level or earlier, if `modtime` is set, the modification date (on the local filesystem) of a newly synced file is the date and time *at the server* when the file was submitted to the depot. • For Perforce clients at the 2000.1 level or higher, 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 Perforce client version. For files *with* the `+m` (modtime) file type modifier: • For Perforce clients at the 99.2 level or earlier, the `+m` modifier is ignored, and the behavior of `modtime` and `nomodtime` is as documented above. • For Perforce clients at the 2000.1 level or higher, 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.	`nomodtime` (i.e. date and time of sync) for most files. Ignored for files with the `+m` file type modifier.

Option	Choice	Default
`[no]rmdir`	If set, `p4 sync` deletes empty directories in a client if all files in the directory have been removed.	`normdir`

- By default, any user can edit any workspace specification with `p4 client -c clientname`. To prevent this from happening, set the `locked` option and use `p4 passwd` to create a password for the client workspace owner.
- The `compress` option speeds up client/server 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 for line speeds under T1, and should be left unset otherwise.
- The `LineEnd:` field controls the line-ending character(s) used for text files in the client workspace.

Note The `LineEnd:` option is new to Perforce 2001.1. It renders the previous convention of specifying `crlf` or `nocrlf` in the `Options:` field obsolete.

The behavior of the mutually-contradictory combination of `LineEnd: win` and `Options: crlf` is undefined.

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`	Macintosh 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 that are synced into a client workspace; however, when files are submitted back to the Perforce Server, 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`s untouched. When you sync your client workspace, line endings are set to `LF`. If you edit the file on a Windows machine, and your editor inserts `CR`s before each `LF`, the extra `CR`s 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.

For more information about how Perforce uses the line-ending settings, see "CR/LF Issues and Text Line-endings" in the Perforce knowledge base:

`http://kb.perforce.com/article/63`

- By default, if a directory in the client workspace is empty, (for instance, because all files in the depot mapped to that directory have been deleted since the last sync), a `p4 sync` operation will still leave the directory intact. If you use the `rmdir` option, however, `p4 sync` deletes the empty directories in the client workspace.

  If the `rmdir` option is active, a `p4 sync` operation may sometimes remove your current working directory. If this happens, just change to an existing directory before continuing on with your work.

- Files with the `modtime` (`+m`) type are primarily intended for use by developers who need to preserve original timestamps on files. The use of `+m` in a file type overrides the client's `modtime` or `nomodtime` setting. For a more complete discussion of the `+m` modifier, see the *File Types* section.

- If you are using multiple or alternate client roots (the `AltRoots:` field), you can always tell which client root is in effect by looking at the `Client root:` reported by `p4 info`.

- To specify a Perforce client on Windows that spans multiple drives, use a `Root:` of `null`, and specify the drive letters in the client workspace view. For instance, the following client spec with a `null` client 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/...
```

  Use lowercase drive letters when specifying workspaces across multiple drives.

Examples

`p4 client`	Edit or create the client workspace specification named by the value of `P4CLIENT` or its equivalents.

`p4 client -t sue joe`	Create or edit client workspace `joe`, opening the form with the field values and workspace options in client workspace `sue` as defaults.
`p4 client -d release1`	Delete the client workspace `release1`.

Related Commands

To list client workspaces known to the system	`p4 clients`
To read files from the depot into the client workspace	`p4 sync`
To open new files in the client workspace for addition to the depot	`p4 add`
To open files in the client workspace for edit	`p4 edit`
To open files in the client workspace for deletion	`p4 delete`
To write changes in client workspace files to the depot	`p4 submit`

p4 clients

Synopsis

List all client workspaces currently known to the system.

Syntax

`p4 [g-opts] clients [ -u user ] [ -e namefilter -m max ]`

Description

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

```
Client clientname moddate root clientroot description
```

For example:

```
Client paris 1999/02/19 root /usr/src 'Joe's client'
```

describes a client workspace named `paris`, last modified on February 19, 1999 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 namefilter` option to limit the output to labels whose name matches the *namefilter* pattern.

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

`-m max`	List only the first *max* client workspaces.
`-e namefilter`	List only client workspaces matching *namefilter*.
`-u user`	List only client workspaces owned by *user*.
g-opts	See the *Global Options* section.

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 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 Perforce users	`p4 users`

p4 configure

Synopsis

Manage server configuration variables.

Syntax

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

Description

Use `p4 configure` to change the configuration of an active server. The configuration variables are described in `p4 help configurables`.

Changes to most server configurables are immediate; you do not have to stop and restart the server for the change to take effect.

After installing Perforce, it is good practice to enable server process monitoring by setting `monitor` to 1 or 2, require ticket-based authentication by setting `security` to 3, and preventing the automatic creation of new users by setting `dm.user.autocreate` to 1 or 2.

Where a configurable refers to a number of bytes, “K” and “M” abbreviations are interpreted as the appropriate powers of two. For other configurables, “K” and “M” refer to 1000 and 1,000,000.

The following configurables are supported:

Configurable	Client or Server?	Default Value	Meaning
`dbjournal.bufsize`	Server	16K	Buffer size for journal and checkpoint read/write operations.
`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.

Configurable	Client or Server?	Default Value	Meaning
`db.replication`	Server	unset	Control behavior of commands that access metadata (`db.*` files) on the Perforce 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` flag.
`dm.annotate.maxsize`	Server	10M	Maximum revision size for `p4 annotate`
`dm.domain.accessupdate`	Server	300	Wait this many seconds before requesting a write lock to update an access time.
`dm.domain.accessforce`	Server	3600	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.
`dm.grep.maxrevs`	Server	10K	Maximum number of revisions that can be searched with `p4 grep`.
`dm.shelve.maxfiles`	Server	10M	Maximum number of files that can be shelved with `p4 shelve`.
`dm.shelve.maxsize`	Server	0	Maximum size of a file that can be shelved, or 0 for unlimited.
`dm.user.accessupdate`	Server	300	Wait this many seconds before requesting a write lock to update an access time.
`dm.user.accessforce`	Server	3600	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.

Configurable	Client or Server?	Default Value	Meaning

`dm.user.noautocreate`	Server	1	Control behavior of automatic user creation.
			`0`: A user is created whenever a nonexistent user runs any Perforce command
			`1`: new users may only be created by running `p4 user`
			`2`: new users may only be created by superusers running `p4 user`

`filesys.binaryscan`	Client	8K	Scan the first `filesys.binaryscan` bytes for binary data when running `p4 add`.
`filesys.bufsize`	Client, Server	4K	Buffer size for client-side read/write operations.
`filesys.extendlowmark`	Client	32K	Minimum filesize before preallocation (Windows)
`filetype.maxtextsize`	Client	10M	Maximum file size for text type detection.
`lbr.bufsize`	Server, Proxy	4K	Buffer size for read/write operations to server's archive of versioned files.
`lbr.replication`	Server	unset	Control behavior of user commands that access versioned files on the Perforce server: `readonly`: user commands that read depot files are accepted; user commands that modify files are rejected. `none`: no access to versioned files is permitted. Equivalent to starting a replica with the `p4d -D readonly` or `-D none` flags.
`lbr.verify.in`	Server	1	Verify contents from the client to server? (1 for yes, 0 for no)

Configurable	Client or Server?	Default Value	Meaning
`lbr.verify.out`	Client, Server	1	Verify contents from the server to client? (1 for yes, 0 for no)
`minClient`	Server	none	Lowest version of client software permitted to connect to this server, set by `p4 configure set minClient=`*`version`*.
`minClientMessage`	Server	none	Message to issue if client software is too old, set by `p4 configure set minClientMessage=`*`message`*.
`monitor`	Server	0	Server process monitoring: `0`: Server process monitoring off `1`: monitor active processes only `2`: monitor both active and idle processes. See `p4 monitor` for details.
`net.backlog`	Server, Proxy	10	Maximum length of queue for pending connections. Consider increasing if users find themselves unable to connect to extremely heavily-loaded servers.

Configurable	Client or Server?	Default Value	Meaning
`net.mimcheck`	Server, Proxy	1	Man-in-the-middle network security level: `0`: Disable MitM checks. `1`: Check proxy/broker connections in Zeroconf 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`: All intermediate services are checked, and all service users must have valid tickets. Requires 2010.2 server and intermediate services.
`net.tcpsize`	Client, Server, Proxy	64K	TCP send and receive buffer sizes, set on connection. Consider increasing for high-latency connections, such as the Proxy. Actual buffer size is large of this value and that defined by the OS.
`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.

Configurable	Client or Server?	Default Value	Meaning
`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. See `p4 monitor` for details.
`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.
`spec.hashbuckets`	Server	0	Number of buckets (subdirectories) into which files in the spec depot are hashed. By default, do not hash. May improve performance on older filesystems where performance is a function of the number of files per directory.
`startup.`*`n`*	Server	none	For replica servers, set `startup.1` through `startup.`*`n`* to be `p4 pull` threads to be spawned at startup.

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`
`show allservers`	Shows the configuration variables for all servers known to the system
`show variable`	Shows the setting of a specific configuration variable
`show P4NAME`	If a Perforce server was invoked with `-In P4NAME` or with the `P4NAME` environment variable set to a server name, shows the settings of the named server.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- Configurables that affect client-side behavior can also be set in `P4CONFIG` files.

  When used in place of environment variables, certain server-related configurables are read-only; you cannot change `P4ROOT`, or `P4JOURNAL` with `p4 configure`.

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

- Changes to most configurables take effect immediately; for example, you do not have to restart the server in order for changes to configurables such as `monitor` (enable/disable the `p4 monitor` command) or `security` (set the server security level) to take effect.

  Changes to `P4PORT`, `P4ZEROCONF`, the `startup.n configurables` used in replicated environments, `net.tcpsize`, and `net.backlog` require a server restart.

- Servers can be identified by name. In multiserver and replicated environments, a master server can control the settings of multiple replica servers by specifying the server name as part of the configurable. See the *System Administrator's Guide* for details.

Related Commands

To list all counters and their values	`p4 counters`

p4 copy

Synopsis

Copy files from one location in the depot to another.

Syntax

```
p4 [g-opts] copy [-c change] [-n -v] [-t filetype] fromFile[rev] toFile
p4 [g-opts] copy [-c change] [-n -v] -b branch [-r] [toFile[rev]] ...
p4 [g-opts] copy [-c change] [-n -v] -b branch -s fromFile[rev] [toFile
...]
```

Description

The `p4 copy` command schedules an integration and resolve actions necessary to make a set of target files identical to a set of source files. Files opened for integrate are resolved as copies and (by default) synced to the workspace. Changes in the target that were not previously merged into the source are overwritten.

A `p4 copy` is equivalent to invoking `p4 integrate fromFiles toFiles`, and then immediately running `p4 resolve -at`.

Options

`-c change`	If a changelist number is provided, the files are opened in the numbered pending changelist rather than in the default changelist.
`-n`	Preview the copy that would be performed, without actually opening the files for integrate.
`-v`	Open files without syncing `toFiles` into the client workspace. By default, `p4 copy` syncs the destination files to the client workspace from `fromFiles`. With this flag, you can save time by not copying `toFiles` to the client workspace. Instead, you can fetch them with `p4 sync` when you need them.
`g-opts`	See the *Global Options* section.

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`

Examples

`p4 copy //depot/test/... //depot/r1.0/...`	Copy files from a testing branch into a release branch.

p4 counter

Synopsis

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

Syntax

```
p4 [g-opts] counter countername
p4 [g-opts] counter countername value
p4 [g-opts] counter -d countername
p4 [g-opts] counter -f [ change|job|journal ] value
p4 [g-opts] counter -i countername
```

Description

Counters provide long-term variable storage for scripts that access Perforce. For example, the Perforce review daemon uses a counter (`review`) that stores the number of the last processed changelist.

When used in the form `p4 counter countername`, the value of variable *countername* is returned. When `p4 counter countername value` is used, the value of variable *countername* is set to *value*, and if *countername* does not already exist, it is created.

The Perforce server uses three counters in the course of its regular operations: `change`, `job`, and `journal`. Superusers can use the `-f` flag to force changes to these counters. Changes to these counters are not without risk; see the *Release Notes* for examples of the types of situations in which manually resetting these counters might be appropriate.

Options

`-d` *countername*	Delete variable *countername* from the Perforce server.
`-f [change|job|journal]`	Force a change to one of three internal counters used by Perforce. Most installations rarely, if ever, need to use this flag.
`-i` *countername*	Increment variable *countername* by 1 and return the new value. This option can only be used with numeric counters.
g-opts	See the *Global Options* section.

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` flag

- 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 last changelist number known to the Perforce server (the output of `p4 counter change`) includes pending changelists created by users, but not yet submitted to the depot. If you're writing change review daemons, you may also want 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
  ```

- Prior to Release 2010.2, you could also set the server security level and the server monitoring level with the `security` and `monitor` counters, respectively. The preferred method is to use `p4 configure`. The old commands have been preserved for compatibility, but as of Release 2010.2, the changes take effect without having to restart the server.

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.

Related Commands

To control server configuration	`p4 configure`
To list all server 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

Synopsis

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

Syntax

`p4 [g-opts] counters`

Description

The Perforce 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.

Options

g-opts	See the *Global Options* section.

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 dbschema

Synopsis

Report information about metadata in the Perforce database.

Syntax

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

Description

The `p4 dbschema` command reports information about the database structure in which the Perforce 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 Perforce 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 the *Global Options* section.

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

Synopsis

Display size or simple statistics for a database table.

Syntax

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

Description

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

To obtain size information, use `p4 admin dbstat -s`.

Options

`-a`	For `p4 admin dbstat`, display statistics for all tables.
`-h`	Display a histogram showing distances between leaf pages.
`dbtable`	Display statistics for only the specified tables (for instance, `db.have`, `db.user`, and so on.)
`-s`	Report file sizes of database tables.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- Because `p4 dbstat` locks out 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.

p4 delete

Synopsis

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

Syntax

```
p4 [g-opts] delete [-c changelist#] [-n -v] 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 sent to the server 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 will not appear on the client's *have list*.

Options

`-c change#`	Opens the files for `delete` within the specified changelist. If this flag 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.
`-v`	Delete a file that is not synced into the client workspace.
`g-opts`	See the *Global Options* section.

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.

- Perforce 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`.

Examples

`p4 delete //depot/README`	Opens the file called `README` in the depot's top level directory for deletion. The corresponding file within the client 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 client workspace for deletion. The file is immediately removed from the client workspace, but won't be deleted from the depot until changelist 40 is sent to the server 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 depot

Synopsis

Create or edit a depot specification.

Syntax

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

Description

The files on a Perforce server are stored in a depot. By default, there is one depot on every Perforce server, and its name is `depot`.

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

Other `local` depots work the same way the default `depot` is used. For example, to sync a file `README` in the `rel2` directory of the depot `new`, add `//new/rel2/...` to the left-hand side of your client workspace mapping, and run `p4 sync //new/rel2/README`.

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

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

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

For instance, if the spec depot is present and 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 current set of client, depot, branch, label, typemap, group, user, and job forms. For more information about setting up a spec depot, see the *System Administrator's 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.

Form Fields

Field Name	Type	Description
`Depot:`	Read-Only	The depot name as provided in `p4 depot` *`depotname`*.
`Owner:`	Writable	The user who owns the depot. By default, this is the user who created the depot.
`Description:`	Writable	A short description of the depot's purpose. Optional.
`Type:`	Writable	`local`, `remote`, `spec`, or `archive`. Local depots are writable; remote depots are proxies for depots residing on other servers, and cannot be written to. The spec depot, if present, archives edited forms. Archive depots are used for offline storage of infrequently-used content.
`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.
`Suffix:`	Writable	If the `Type:` is `spec`, this field holds an optional suffix for generated paths to objects in the spec depot. The default suffix is `.p4s`. You do not need a suffix to use the spec depot, but supplying a file extension to your Perforce server's versioned specs enables users of GUI client software to associate Perforce specifications with a preferred text editor. If the `Type:` is `local` or `remote`, this field is ignored.
`Map:`	Writable	If the `Type:` is `local`, `spec`, or `archive`, set the map to point to the relative location of the depot subdirectory relative to the Perforce server's `P4ROOT`. 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.

Options

`-d` *`depotname`*	Delete the depot *`depotname`*. The depot must not contain any files; the Perforce 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.
`-i`	Read a depot specification from standard input.
`-o` *`depotname`*	Write a depot specification to standard output.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- A depot created with `p4 depot` is not physically created in the server until files have been added to it with `p4 add`.
- Users are not able to access a new depot created with `p4 depot` until permission to access the depot is granted with `p4 protect`.
- Remote depots are accessed by a virtual user named `remote` (or, if configured, by the service user configured for the server that originates the request), and by default, all files on any Perforce server 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 Perforce 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.
- By default, the `Map:` field on a local depot points to a depot directory matching the depot name, relative to the server root (`P4ROOT`) setting for your server. 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`.

Related Commands

To view a list of all depots known to the Perforce 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

Synopsis

Display a list of depots known to the Perforce server.

Syntax

`p4 [g-opts] depots`

Description

Lists all the remote and local depots known to the Perforce server, in the form:

```
Depot name date type address map description
```

where *name*, *date*, *type*, *address*, *map*, and *description* are as defined in the `p4 depot` form.

Options

g-opts	See the *Global Options* section.

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 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

Synopsis

Provides information about changelists and the changelists' files.

Syntax

`p4 [g-opts] describe [ -dflag -s -S ] changelist#...`

Description

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

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 server.)

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.

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

While running `p4 describe`, the server uses Perforce's built-in client-side diff subroutine. The `P4DIFF` variable has no effect on this command.

Options

`-s`	Display a shortened output that excludes the files' diffs.
`-S`	Display files shelved for the specified changelist, including diffs of those files against their previous depot revision.
`-dflag`	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the *Usage Notes* below for a flag listing.
`g-opts`	See the *Global Options* section.

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 flags supported by `p4 describe` are:

Flag	Meaning
`-dn`	RCS
`-dc`	context
`-ds`	summary
`-du`	unified
`-db`	ignore changes made within whitespace
`-dw`	ignore whitespace altogether

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 diff

Synopsis

Compare a client workspace file to a revision in the depot.

Syntax

```
p4 [g-opts] diff [-dflag -f -m max -sa -sb -sd -se -sr -sl -t] [file[rev#]...]
```

Description

`p4 diff` runs a diff program on the Perforce client, comparing files in the client 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.

By default, the diff routine used is the one built into the `p4` client program. To change this diff routine to an external diff program, set the `P4DIFF` environment or registry variable to point to the new program.

Options

`-f`	Force the diff (if no revision is specified, against the head revision), even when the client file is not open for `edit`.
`-dflags`	Pass flags to the underlying diff routine (see the *Usage Notes* below for details)
`-m max`	Limit output to diffs (or status) of only the first *max* files, unless the `-s` flag is used, in which case the `-m` flag is ignored.
`-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.
`-sr`	Show only the names of opened files in the client workspace that are identical to 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` flag together with the `-sl` flag, files that are open for edit are also compared and their status is listed.
`-t`	Diff the revisions even if the files are not of type `text`.
`g-opts`	See the *Global Options* section.

Usage Notes

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

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

Flag	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc`	context output format, showing line number ranges and three lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du`	unified output format, showing added and deleted lines with sufficient context for compatibility with the `patch(1)` utility.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs
`-db`	ignore changes made within whitespace; this flag implies `-dl`.
`-dw`	ignore whitespace altogether; this flag implies `-dl`.

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

  ```
  p4 diff -dub file
  ```

  specifies a unified diff that ignores changes in whitespace.
- The header line of a unified diff produced with the `-du` option for use with `patch(1)` displays filenames in Perforce syntax, not local syntax.

Examples

`p4 diff file#5`	Compare the client workspace revision of file `file` to the fifth depot revision.
`p4 diff @1999/05/22`	Compare all open files in the client workspace to the revisions in the depot as of midnight on May 22, 1999.
`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 is the UNIX version of this command; it uses a pipe. Most operating systems have some equivalent way of performing these operations in series). For more information about the `-x` option to `p4`, see the *Global Options* section.

Related Commands

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

p4 diff2

Synopsis

Compare two depot file revisions.

Syntax

```
p4 [g-opts] diff2 [-dflags -q -t -u] file1[rev] file2[rev]
p4 [g-opts] diff2 [-dflags -q -t -u] -b branch [[fromfile[rev]] tofile[rev]]
```

Description

`p4 diff2` uses the Perforce server'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.

`p4 diff2` does not use the diff program specified by the environment variable `P4DIFF`. The diff algorithm used by `p4 diff2` runs on the machine hosting the Perforce server, and always uses the server's built-in diff routine.

You can specify file patterns as arguments in place of specific files, with or without revision specifiers; this causes Perforce to perform multiple 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, and be sure that the wildcards in the two file patterns match.

Perforce 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 will display the *`summary`* as `<none>`.

Options

`-q`	Quiet diff. Display only the header, and don't even display that when the file revisions' contents and types are identical.
`-d`*`flags`*	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the *Usage Notes* below for a listing of these flags.
`-b` *`branchname fromfile[rev] tofile[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 *`fromfile`* or *`tofile`*.
`-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 Perforce syntax.
`g-opts`	See the *Global Options* section.

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

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

Flag	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc`	context output format, showing line number ranges and three lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.
`-du`	unified output format, showing added and deleted lines with sufficient context for compatibility with the `patch(1)` utility.

Flag	Name
`-db`	ignore changes made within whitespace
`-dw`	ignore whitespace altogether

- To pass more than one flag 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 Perforce 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] ]` may seem incorrect at first. Since the branch mapping maps *fromfiles* to *tofiles*, why would you specify both *fromfile* and *tofile* file patterns? You wouldn't, but this syntax 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 fist 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 dirs

Synopsis

List the immediate subdirectories of specified depot directories.

Syntax

`p4 [g-opts] dirs [-C -D -H] depot_directory[revRange]...`

Description

Use `p4 dirs` to find the immediate subdirectories of any depot directories provided as arguments. Any directory argument must be provided in depot or local syntax and must end with the `*` wildcard. *If you use the "..." wildcard, you will receive the wrong results!*

`p4 dirs` only lists the immediate subdirectories of the directory arguments. To recursively list all of a directory's subdirectories, call `p4 dirs` multiple times.

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` flag.

This command is meant to be used in scripts that call Perforce; it is unlikely that you'll have a need to call it from the command line.

Options

`-C`	Display only those directories that are mapped through the current client workspace 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.
`g-opts`	See the *Global Options* section.

Usage Notes

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

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

- Perforce does not track directories in its database; thus, the subdirectory values are not looked up, but are computed. This accounts for some of the strange details of the `p4 dirs` implementation, such as the fact that the "`...`" wildcard is not supported.

Examples

`p4 dirs //depot/projects/*`	Returns a list of all the immediate subdirectories of `//depot/projects`.
`p4 dirs //depot/a/* //depot/b/*`	Returns a list of all immediate subdirectories of `//depot/a` and `//depot/b`.
`p4 dirs //depot/...`	The "`...`" wildcard is not supported by `p4 dirs`.

Related Commands

To list all the files that meet particular criteria	`p4 files`
To list all depots on the current Perforce server	`p4 depots`

p4 edit

Synopsis

Opens file(s) in a client workspace for edit.

Syntax

```
p4 [g-opts] edit [-c changelist#] [-k] [-n] [-t type] file...
```

Description

`p4 edit` opens files for editing within the client workspace. The specified file(s) are linked to a changelist, but the files are not actually changed in the depot until the changelist is sent to the server by `p4 submit`.

Perforce 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`.

Options

`-c change#`	Opens the files for edit within the specified changelist. If this flag is not provided, the files are linked to the default changelist.
`-t type`	Stores the new file revision as the specified type, overriding the file type of the previous revision of the same file. See the *File Types* section for a list of file types.
`-k`	Keep existing workspace files; mark the file as open for edit even if the file is not in the client workspace view. Use `p4 edit -k` only in the context of reconciling work performed while disconnected from the Perforce Server.
`-n`	Preview which files would be opened for edit, without actually changing any files or metadata.
`g-opts`	See the *Global Options* section.

Usage Notes

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

Since `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. Since this file revision is not the head revision, you muse use `p4 resolve` before the file can be stored in the depot with `p4 submit`.

By default, Perforce 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 Perforce, `p4 edit` was called `p4 open`.

Examples

`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 Perforce wildcards, see "Limitations on characters in filenames and entities" on page 300.

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 export

Synopsis

Extract journal or checkpoint records.

Syntax

```
p4 export -c token [-J prefix] [-f] [-l lines] [-F filter] [-T tables]
p4 export -j token [-J prefix] [-f] [-l lines] [-F filter] [-T tables]
p4 export -j token [-J prefix] -r [-F filter] [-T tables]
```

Description

This command reports checkpoint and journal metadata from a Perforce 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` flag with a list of tables whose data you wish to exclude. To exclude data from multiple tables, the list must be quoted, and the table names separated by spaces. The table names must begin with "`db.`", following the naming convention used for database files in the server root directory.

Options

`-c`	Specifies a checkpoint number or position token of the form *`checkpointnum#byteoffset`*.
`-j`	Specify a journal number or position token of the form *`journalnum/byteoffset`*.
`-f`	Format the output so that non-textual datatypes are formatted appropriately.
`-r`	Display raw journal output; this argument applies to journals only.
`-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.
`-F` *`filter`*	Limit output to records that match the specified *`filter`* pattern.
`-T` *`tables`*	Supply a list of database tables (for example, `db.have`) to exclude from export.
`g-opts`	See the *Global Options* section.

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 filelog

Synopsis

Print detailed information about files' revisions.

Syntax

```
p4 [g-opts] filelog [ -c change -h -i -l -L -t -m maxrev -s ] file...
```

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.

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: `add`, `edit`, `delete`, `branch`, `import`, or `integrate`;

  If the action is `import` (that is, integrate from a remote depot) or `integrate`, Perforce displays a second line description, formatted as

  ```
  ... #integration-action partner-file
  ```

  See `p4 integrated` for a full description of integration actions.
- *`date`* is the submission date (by default), or date and time (if the `-t` flag 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; and
- *`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.

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 via `p4 integrate`, Perforce 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).
`-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.
`-t`	Display the time as well as the date.
`-m maxrev`	List only the first *maxrev* changes per file output.
`-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)
g-opts	See the *Global Options* section.

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` flags 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 `+Sn`) 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 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 files

Synopsis

Provide information about files in the depot without accessing their contents.

Syntax

```
p4 [g-opts] files [-a] [-A] [-m max] file[revRange]...
```

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 change# (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`, or `integrate`
- `change#` is the number of the changelist that this revision was submitted in, and
- `filetype` is the Perforce *file type* of this file at the head revision.

Unlike most Perforce commands, `p4 files` reports on any file in the depot; it is not limited to only those files that are visible through the client view. Of course, if a file pattern on the command line is given in client syntax, only client files 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.
`-m max`	Limit output to the first `max` files.
`g-opts`	See the *Global Options* section.

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`.)
- Since 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 @2000/12/10`	Provides information about all depot file revisions that existed on December 10, 2000.
`p4 files @2001/03/31:08:00,@2001/03/31:17:00`	Lists all files and revisions changed during business hours on March 31, 2001.
`p4 files //depot/proj2/...@p2lab`	Lists files and revisions under the directory `//depot/proj2/...` that are included in 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`)

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 fix

Synopsis

Link jobs to the changelists that fix them.

Syntax

```
p4 [g-opts] fix [ -d ] [ -s status ] -c changelist# jobName ...
```

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. Since 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

`-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` flag 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 the *Global Options* section.

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 Perforce 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` flag 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

Synopsis

List jobs and the changelists that fix them.

Syntax

```
p4 [g-opts] fixes [-i -m max -j job -c changelist#] [file[revRange]...]
```

Description

After a job has been linked to a particular numbered changelist with `p4 fix`, `p4 change`, or `p4 submit`, the job is said to have been *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 flags 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` flag with a file pattern argument to include fixes made by changelists that were integrated into the specified files.
- Use the `-m max` flag to limit the output to the first `max` fixes.

Options

`-c changelist#`	Limit the displayed fixes to those that include the specified changelist.
`-j jobname`	Limit the displayed fixes to those that include the specified job.
`-i files...`	Include fixes made by changelists that affected files integrated into the specified files.
`-m max`	List only the first `max` fixes.
`g-opts`	See the *Global Options* section.

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 -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

Synopsis

Update a client workspace's have list without actually copying any files.

Syntax

`p4 [g-opts] flush [-n] [file[revRange]...]`

Warning

Using `p4 flush` incorrectly ***can be dangerous***.

If you use `p4 flush` incorrectly, the server's metadata will not reflect the actual state of your client workspace, and subsequent Perforce commands will not operate on the files you expect! Do not use `p4 flush` until you fully understand its purpose.

It is rarely necessary to use `p4 flush`.

Description

`p4 flush` performs half the work of a `p4 sync`. Running `p4 sync filespec` has two effects:

- The file revisions in the *filespec* are copied from the depot to the client workspace;
- The client workspace's *have list* (which tracks which file revisions have been synced, and is stored on the Perforce server) is updated to reflect the new client workspace contents.

`p4 flush` performs only the *second* of these steps. Under most circumstances, this is not desirable, since a client workspace's have list should always reflect the client workspace's true contents. However, if the client 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. Since `p4 flush` performs no actual file transfers, this command is much faster then the corresponding `p4 sync`.

Use `p4 flush` only when you need to update the have list to match the actual state of the client workspace. The *Examples* subsection describes two such situations.

Options

`-n`	Display the results of the flush without actually performing the flush. This lets you make sure that the flush does what you think it will do before you do it.
g-opts	See the *Global Options* section.

Usage Notes

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

- Since `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, since 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.

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. The standard method calls for each user to run identical `p4 sync` commands, but since the line speed is slow, there's a faster way:
  - One user runs `p4 sync` *`files`* from his 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.

  Since `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/...
  ```

  He decides that all the files under `/usr/joe/project1` need to be included in the workspace, and accomplishes this by using `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's surprised to see that Perforce deletes every non-open file in the client workspace and replaces it with an identical copy of the same file!

Perforce behaves this way because the have list describes each file's location relative to the client root, and the physical location of each file is only computed when each Perforce command is run. Thus, Perforce thinks that each file has been relocated, and the `p4 sync` deletes the file from its old location and copies it into its new location.

To make better use of Perforce, Joe might have performed a `p4 flush #have` instead. This would have updated his client workspace's have list to reflect the files' "new" locations 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

Synopsis

Dump file info in format suitable for parsing by scripts.

Syntax

```
p4 [g-opts] fstat [-F filter -T fields -m max -r] [-c|-e change] [-Ox -Rx
-Sx] file[rev]...
```

Description

The `p4 fstat` command dumps information about each file, with each item of information on a separate line.

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

To change the field on which output is sorted, use one of the -S*x* options, and to reverse sort order, use the `-r` option.

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` flag.

To limit output to the set of fields specified in a *fields* argument, use the `-T fields` flag.

The output is best used within a Perforce API application where the items can be accessed as variables, but is also suitable for parsing by scripts.

Form Fields

Field Name	Description	Example/Notes
`clientFile`	local path to file (in local syntax by default, or in Perforce syntax with the `-Op` option)	`/staff/userid/src/file.c` (or `//workspace/src/file.c` in Perforce syntax)
`depotFile`	depot path to file	`//depot/src/file.c`
`path`	local path to file	`//workspace/src/file.c`
`isMapped`	set if mapped client file is synced	set (`... isMapped`) or not present
`shelved`	set if file is shelved	
`headAction`	action taken at head revision, if in depot	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, or `integrate`

Field Name	Description	Example/Notes
headChange	head revision changelist number, if in depot	1, 2, 3... *n*
headRev	head revision number, if in depot	1, 2, 3... *n*
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
headType	head revision type, if in depot	text, binary, text+k, etc. (see the chapter on *File Types*.)
haveRev	revision last synced to workspace, if on workspace	1, 2, 3... *n*
desc	changelist description (if using -e *changelist* and if the file was part of *changelist*)	A Perforce changelist
digest	MD5 digest of a file (requires -Ol option)	A 32 hexadecimal digit string
fileSize	file length in bytes (requires -Ol option)	63488
action	open action, if opened in your workspace	one of add, edit, delete, branch, move/add, move/delete, or integrate
type	open type, if opened in your workspace	A Perforce file type
actionOwner	the user who opened the file, if open	A Perforce username

Field Name	Description	Example/Notes
`change`	open changelist number, if opened in your workspace	`1`, `2`, `3`... *n*
`resolved`	the number, if any, of resolved integration records	`1`, `2`, `3`... *n*
`unresolved`	the number, if any, of unresolved integration records	`1`, `2`, `3`... *n*

`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@cws1`
		`..... otherOpen1 user2@cws2`
		`..... otherOpen2 user3@cws3`

`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	unset (`... otherLock`) or not present
`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`, or `integrate`
`otherChange`*n*	for every changelist with the file open, the changelist	A changelist number

Field Name	Description	Example/Notes
`ourLock`	present and set to null if the current user has the file locked, otherwise not present	unset (`... ourLock`) or not present
`resolveAction`*n* `resolveBaseFile`*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.
`totalFileCount`	The number of files examined.	Appears in the first file's output when you use the `-m` *`max`* option in conjunction with one of the `-S`*`x`* or `-r` sorting options.

Options

`-F` *`filter`*	List only those files that match the criteria specified by *`filter`*. Please see the *Usage Notes* below for a discussion of filters.
`-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.
`-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.
`-m` *`max`*	Produce fstat output for only the first *`max`* files.
`-r`	Sort the output in reverse order.
`-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. On servers older than release 2005.1, this field may be expensive to compute, particularly for text files with many revisions.
`-Op`	Display the `clientFile` in Perforce 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).

`-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.
`-Rr`	Limit output to open files that have been resolved.
`-Rs`	Limit output to shelved files. Requires `-e` *`changelist`* option.
`-Ru`	Limit output to open files that are unresolved.
`-St`	Sort by filetype.
`-Sd`	Sort by date.
`-Sr`	Sort by head revision.
`-Sh`	Sort by have revision.
`-Ss`	Sort by filesize.
`g-opts`	See the *Global Options* section.
	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`.

Filters used for `fstat` are case-sensitive. All alphanumeric strings (including words including embedded punctuation) separated by whitespace are indexed as words. For case-insensitive matching, use the case-insensitive equality operator: (`~=`).

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.

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.

The usual comparison operators (=, >, <, >=, and <=) are available.

Additionally, you can use the NOT operator (^) to negate the sense of some comparisons.

To search for text containing characters that are filter expression operators, escape the characters with a backslash (\) character. Using backslashes to escape search queries has two special cases: you can escape the Perforce "..." wildcard with \..., and you can search for empty fields with \0.

The behavior of these operators depends on the type of field you're comparing against:

Field Type	Use of Comparison Operators in Filters
word	The equality operator (=) or case-insensitive equality operator (~=) must match the value in the word field exactly. The relational operators perform comparisons in ASCII order.
text	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.
line	As for field type text, above.
select	The equality operator (=) or case-insensitive equality operator (~=) match a file 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.

Other Usage Notes

- If you use `-e changelist#` with the `-Ro` option, only pending changes are considered, so that files open for add are included in the output.
- The syntax of `p4 fstat` was changed in Release 2004.2. The older `-C`, `-H`, `-W`, `-P`, `-l`, and `-s` options are supported for compatibility purposes.
- For files containing the special characters `@`, `#`, `*`, and `%`, the `clientFile` displays the special character, and the `depotFile` displays the filename containing the ASCII expression of the character's hexadecimal value.
- The `size` and `digest` fields 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 fstat file.c`	Displays information on `file.c`
`p4 fstat -Rc 20 *.c`	Displays information on all `.c` files affected after the checking-in of files under changelist 20.
`p4 fstat -Os file.c`	No client workspace information lines (i.e. `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 -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 grep

Synopsis

Print lines in files (or revisions of files) that match a pattern.

Syntax

```
p4 [g-opts] grep [ options ] -e pattern file[revRange]...
options: -a -i -n -v -Anum -Bnum -Cnum -l -L -T -s -F -G
```

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.

Options

`-e` *`pattern`*	The *`pattern`*s used by `p4 grep` are regular expressions comparable to those used in UNIX; their syntax is fully defined in the output of `p4 help grep`.
`-a`	Search all revisions within the specified range, rather than only the highest revision in the range
`-i`	Perform case-insensitive pattern matching. (By default, matching is case-sensitive.)
`-n`	Display a matching line number after the file revision number.
`-v`	Display files with non-matching lines.
`-F`	Interpret the pattern as a fixed string
`-G`	Interpret the pattern as a regular expression.
`-L`	Display the name of each selected file from which no output would normally have been displayed; scanning stops at the first match.
`-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.)
`-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.
g-opts	See the *Global Options* section.

Usage Notes

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

- By default, `p4 grep` searches at most 10000 revisions. This limit is controlled by the `dm.grep.maxrevs` configurable.

p4 group

Synopsis

Add or delete users from a group, or set the `maxresults`, `maxscanrows`, `maxlocktime`, and `timeout` limits for the members of a group.

Syntax

```
p4 [g-opts] group [ -a ] groupname
p4 [g-opts] group -d [ -a ] groupname
p4 [g-opts] group -o groupname
p4 [g-opts] group -i [ -a ]
```

Description

A *group* is a list of Perforce users. Use groups to set access levels in the `p4 protect` form, limit the maximum amount of data that can be accessed from the server by particular users within a single command, and to set the timeout period for `p4 login` tickets.

To delete a group, use `p4 group -d groupname`, or call `p4 group groupname` and remove all the users from the resulting form.

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 server from a single command. The default value is `unset`. See the *Usage Notes* below for more details.
`MaxScanRows:`	Writable	The maximum number of rows that members of this group can scan from the server from a single command. The default value is `unset`. See the *Usage Notes* below for more details.
`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 the *Usage Notes* below for more details.

Field Name	Type	Description
`Timeout:`	Writable	The duration (in seconds) of the validity of a session ticket created by `p4 login`. The default value is 43200 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`.
`Users:`	Writable, multi-line	The Perforce usernames of the group members. Each user name must be typed on its own line, and should be indented.
`Subgroups:`	Writable, multi-line	Names of other Perforce 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.
`Owners:`	Writable, multi-line	Names of other Perforce users. Group owners without `super` access are permitted to administer this group, provided that they use the `-a` flag. 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.

Options

`-a`	Allow a (non-superuser) group owner to administer the group. The user must be listed in the `Owner:` field of the group.
`-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.

`-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 the *Global Options* section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`super` (`list` for `p4 group -o` or `-a`)

- Ticket `Timeout` 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 `timeout` value of `43200`. 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 2592000 seconds.
- As the number of files in the depot grows, certain commands can significantly slow down the server 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 Perforce superuser can limit the amount of data that the server returns to the client by setting the `MaxResults` value for groups of users. The superuser can also limit the amount of data scanned by the server (whether returned to the client 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.

  If any of the `MaxResults`, `MaxScanRows`, or `MaxLockTime` limits are violated, the server request fails and the user is asked to limit his query.

  If a user belongs to multiple groups, the server 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` and `MaxLockTime` limits are computed in the same way.)

The speed of most server hardware should make it unnecessary to ever set a `MaxResults` value below 10000, a `MaxScanRows` value below 50000, or a `MaxLockTime` value below 1000.

- 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 three limiting values.

Related Commands

To modify users' access levels	`p4 protect`
To view a list of existing groups	`p4 groups`

p4 groups

Synopsis

List groups of users.

Syntax

```
p4 [g-opts] groups [ -m max ] [ [ [ -i ] user | group ] | [ -v group ] ]
```

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

`-m max`	List only the first *`max`* groups.
`-i [ user | group ]`	Display groups to which the *`user`* or *`group`* is an indirect member (that is, by means of inclusion in a subgroup).
`-v [ group ]`	Display verbose output: include `MaxResults`, `MaxScanRows`, `MaxLockTime`, and `Timeout` values.
`g-opts`	See the *Global Options* section.

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.
- The `-v` and `-i` options are mutually exclusive.

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 Perforce access levels for the members of a particular group	`p4 protect`

p4 have

Synopsis

List files and revisions that have been synced to the client workspace

Syntax

```
p4 [g-opts] have [file...]
```

Description

List those files and revisions that have been copied to the client workspace with `p4 sync`. If file patterns are provided, the list is limited to those files that match one of the patterns, and to those files that are 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 (i.e., in *local syntax*).

Options

`g-opts`	See the *Global Options* section.

Usage Notes

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

- Some Perforce 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 have been most recently synced from the depot. It does *not* include files that exist in your client workspace but not in the depot.

  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 haven't submitted them, 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
  ```

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`

p4 help

Synopsis

Provide on-line help for Perforce.

Syntax

```
p4 [g-opts] help
p4 [g-opts] help keyword
p4 [g-opts] help command
```

Description

`p4 help` displays a help screen describing the named *`command`* or *`keyword`*. It's very similar to this manual, but the text is written by the developers.

`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 Chapter in this Manual
`p4 help simple`	Provides short descriptions of the eight most basic Perforce commands.	(none)
`p4 help commands`	Lists all the Perforce commands	*Table of Contents*
`p4 help charset`	Describes how to control Unicode translation	`P4CHARSET` description.
`p4 help environment`	Lists the Perforce environment variables and their meanings	*Environment and Registry Variables*
`p4 help filetypes`	Lists the Perforce filetypes and their meanings	*File Types*
`p4 help jobview`	Describes Perforce jobviews	`p4 jobs` description
`p4 help revisions`	Describes Perforce revision specifiers	*File Specifications*
`p4 help usage`	Lists the six flags available with all Perforce commands	*Global Options*
`p4 help views`	Describes the meaning of Perforce views	*Views*

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 Perforce configuration	`p4 info`

p4 info

Synopsis

Display information about the current client and server.

Syntax

`p4 [g-opts] info [-s]`

Description

The `p4 info` command displays information about the Perforce client and server.

Here's an example of the output from `p4 info`:

```
User name: joe
Client name: joes_client
Client host: joes_workstation
Client root: /usr/joe/projects
Current directory: /usr/joe/projects/source
Client address: 192.168.0.123:1818
Server address: p4server:1666
Server root: /usr/depot/p4d
Server date: 2008/06/28 12:11:47 -0700 PDT
Server uptime: 752:41:33
Server version: P4D/FREEBSD/2010.1/246375 (2010/05/25)
Server license: P4Admin <p4adm> 20 users (expires 2011/01/01)
Server license-ip: 10.0.0.2
Case handling: sensitive
```

To obtain the version of the Perforce client program (`p4`), use `p4 -V`.

Options

g-opts	See the *Global Options* section.
`-s`	Shortened output: exclude information (for example, the workspace root) that requires a database lookup.

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 Perforce's help files	`p4 help`
To view version information for your Perforce client program	`p4 -V`

p4 integrate

Synopsis

Open files for branching or merging.

Syntax

```
p4 [g-opts] integrate [options] fromFile[revRange] toFile
p4 [g-opts] integrate [options] -b branch [toFile[fromRevRange]...]
p4 [g-opts] integrate [options] -b branch -s fromFile[revRange] [toFile...]
    options: -c changelist# -d -Dflag -f -h -i -I -o -n -r -t -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 simplest form of this command is `p4 integrate fromFile toFile`; this lets the Perforce server know that changes in *`fromFile`* need to be propagated to *`toFile`*, and has the following effects:

- If *`toFile`* doesn't 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 common ancestor of *`fromFile`* as *`base`*.
- If both *`toFile`* and *`fromFile`* exist, but *`toFile`* shares no common ancestor with *`fromFile`*, the integration is rejected. Use the `-i` flag to force a baseless merge.
- 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.

(Some of the available flags modify this behavior. See the *Options* section for details.)

The process is complete when you `p4 submit` *`toFile`* to the depot.

To specify multiple files, use wildcards in *`fromFile`* and *`toFile`*. Any wildcards used in *`fromFile`* must match identical wildcards in *`toFile`*. Perforce compares the *`fromFile`* pattern to the *`toFile`* pattern, creates a list of *`fromFile`*/*`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`*.

Options

Because some of the more recent integration flags add complexity to the integration process, we've divided the options into *Basic Integration Flags* and *Advanced Integration Flags*

Basic Integration Flags

`-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`*.
`-n`	Display the integrations this command would perform without actually performing them.
`-v`	Open files for branching without copying *`toFiles`* into the client workspace. Without this flag, `p4 integrate` copies newly-branched *`toFiles`* into the client workspace from *`fromFiles`*. When the `-v` (*virtual*) flag is used, With this flag, you can save time by not copying *`toFiles`* to the client workspace. Instead, you can fetch them with `p4 sync` when you need them.
`-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.
`g-opts`	See the *Global Options* section.

Advanced Integration Flags

`-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: first, integration is performed from *`fromFile`* to *`toFile`*; then integration is performed from *`toFile`* to *`fromFile`*. This variation of `p4 integrate` was written to provide some needed functionality to graphical Perforce client programs; 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.
`-d`	The `-d` flag enables integrations around deleted revisions. Using `-d` implies `-Dt`, `-Ds`, and `-Di`. If the target file has been deleted and the source file has changed, (or the source file has changed and subsequently been deleted), using `-d` re-branches the changes from the source file on top of the target file. If the source file has been deleted and the target file has changed, using `-d` deletes the target file. If the source file has been deleted and re-added, using `-d` integrates all outstanding revisions of the file, including those revisions prior to the file's deletion. If you do not use the `-d` flag, outstanding edits cannot be mixed with a deleted file.

`-Dt`
`-Ds`
`-Di`
	The `-Dt` flag allows integration around a deleted target file; if the target file is deleted, it is restored with the branched source file.
	The `-Ds` flag allows integration around a deleted source file; if the source file has been deleted, any modified target file is also deleted.
	The `-Di` flag ignores the fact that a source file was deleted and re-added when searching for an integration base.

`-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.
`-i`	Perform the integration even if *`toFile`* and *`fromFile`* share no common ancestor, using the first revision as the *`base`*.
`-I`	Equivalent to `-i`, the `-I` flag exists for compatibility purposes.
`-o`	The `-o` flag outputs the base file name and revision to be used in subsequent resolves, if a resolve is needed.
`-t`	Propagate the source file's filetype to the target file. (Newly-branched files always use the source file's filetype, but without `-t`, the target file retains its previous filetype.)

Usage Notes

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

- *`FromFiles`* are often called *source files*, and *`toFiles`* are often called *target files*.
- Any *`toFiles`* that `p4 integrate` needs to operate on must be included in the `p4 client` workspace view.
- 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`.
- `p4 integrate` can be abbreviated as `p4 integ`. (This abbreviation is used the examples below).

- Whenever a *toFile* is integrated from a *fromFile*, Perforce 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.

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`
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

Synopsis

Show integrations that have been submitted.

Syntax

```
p4 [g-opts] integrated [ -r ] [ -b branch ] file...
```

Description

The `p4 integrated` command shows the integration history of the selected files, in the 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; and
- *integrate-action* describes what the user did during the `p4 resolve` process, and is one of the following:

Integrate Action	What the User Did During the `p4 Resolve` Process
`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*.
`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.

Integrate Action	What the User Did During the `p4 Resolve` Process
`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 (Perforce 2001.1 and later).
`edit into`	*`file`* was integrated into *`partner-file`*, and *`partner-file`* was reopened for `edit` before submission (Perforce 99.2 and later).
`add into`	*`file`* was integrated into previously nonexistent *`partner-file`*, and *`partner-file`* was reopened for `add` before submission (Perforce 99.2 and later).

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`* flag is used, only files integrated from the source to target files in the branch view are shown.

If the optional `-r` flag is provided, the mappings in the branch view are reversed. This flag requires the use of the -b *`branch`* flag.

Options

`g-opts`	See the *Global Options* section.

Usage Notes

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

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 job

Synopsis

Create or edit a defect, enhancement request, or other job specification.

Syntax

```
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 isn't 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 Perforce 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. Since the Perforce superuser can change the fields in the job form with `p4 jobspec`, the fields that make up a job may vary from one Perforce server 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 the *Usage Notes* below for more information).

Form Fields

These are the fields as found in the default job form. Since the fields that describe a job can be changed by the Perforce superuser, the form you see at your site may be very different.

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. Alternately, you can name the job anything at all by replacing the text in this field.
`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 created.
`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 flag. Allows Perforce 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 the *Global Options* section.

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 Perforce superuser has eliminated field ID# `102` (the `Status:` field) with `p4 jobspec`, Perforce is unable to close jobs when the changelists to which they are linked are submitted. Please see the `p4 jobspec` page and the *Perforce System Administrator's Guide* for more information.

- After a job has been created or changed, Perforce 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`.

Related Commands

To list all jobs, or a subset of jobs	`p4 jobs`
To attach a job to an existing changelist	`p4 fix`
To view a list of connections between jobs and changelists	`p4 fixes`
To add or delete a job from a pending changelist	`p4 change`
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 jobs

Synopsis

List jobs known to the Perforce server.

Syntax

```
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 on the server. You can limit the output of the command by specifying various criteria with flags and arguments. If you specify a file pattern, the jobs listed will be limited to those linked to changelists affecting particular files. The `-e` flag 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` flag, 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) flag is used.

If any of the `date`, `user`, `status`, or `description` fields have been removed by the Perforce 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`*. Please see the *Usage Notes* below for a discussion of job views.
`-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` flag, the last *`max`* jobs are included.
`-r`	Display jobs in reverse alphabetical order by job name.

`-R`	Rebuild the job table and reindex each job. Reindexing 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 the *Global Options* section.

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 jobviews 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 jobview '*`word1 word2 ... wordN`*' can be used to find jobs that contain all of *`word1`* through *`wordN`* in any of the job's fields.

Spaces between search terms in jobviews act as boolean AND operations. To find jobs that contain any of the terms (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.

Search results can be narrowed by matching values within specific fields with the jobview syntax "*`fieldname=value`*". The *`value`* must be a single token, including both alphanumeric characters and punctuation.

The wildcard "`*`" allows for partial word matches. The jobview "*`fieldname`*`=string*`" matches "`string`", "`stringy`", "`stringlike`", and so on.

Date fields can be matched by expressing the jobview 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).

To search for words containing characters that are job search expression operators, escape the characters with a backslash (\) character.

The behavior of these operators depends on the type of job field you're comparing against:

Field Type	Use of Comparison Operators in Jobviews
`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, since 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 jobview is "`ShortDesc<filter`", the job matches the jobview, 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.

Other Usage Notes

- The `p4 user` form has a `JobView:` field that allows a jobview to be linked to a particular user. After a user enters a jobview into this field, any changelists he creates automatically list jobs that match the jobview in this field. The jobs that are fixed by the changelist can be left in the form, and the jobs that aren't 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 Perforce 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're using Perforce's default 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 jobview "`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

- Jobviews 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 jobview that matches this "deleted field" value.
- The jobview NOT operator (^) can be used only after an AND within the jobview. Thus, the jobviews "`gui ^name=joe`" and "`gui&^name=joe`" are valid, while the jobviews "`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 jobview and logically ANDed with all jobs NOT containing the string "unwanted".

  Likewise, because the jobview "`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"`.

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 on your server (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 jobview that includes jobs matching the jobview in all new changelists	`p4 user`

p4 jobspec

Synopsis

Edit the jobs template.

Syntax

```
p4 [g-opts] jobspec
p4 [g-opts] jobspec [-i]
p4 [g-opts] jobspec -o
```

Description

The `p4 jobspec` command presents the Perforce administrator with a form in which job fields can be edited, created, deleted, and refined.

Do not confuse the names of the fields in the `p4 jobspec` form with the names of the fields within a job. The fields in the `p4 jobspec` form are used to store information *about* the fields in the `p4 jobs` form.

Form Fields

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 persistence`*.
	• `code`: a unique integer that identifies the field internally to Perforce. The code must be between `106` and `199`. Codes `101` to `105` are reserved for Perforce use; see the *Usage Notes* below for more details.
	• `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`.

Field Name	Description

`Fields: (cont'd)`	• `length`: recommended length for display boxes in GUI clients accessing this field. Use a value of `0` to let a Perforce client program choose its own value.
	• `persistence`: 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.
	  • `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 persistence 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 the *Usage Notes* below for special considerations for field 102.

Field Name	Description
`Comments:`	Textual comments that appear at the top of each `p4 job` form. Each line must begin with the comment character `#`. See the *Usage Notes* below for special considerations for these comments if your users need to enter jobs through P4V, the Perforce Visual Client.

Options

`-o`	Write the jobspec form to standard output.
`-i`	Read the jobspec form from standard input.
`g-opts`	See the *Global Options* section.

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` flag

- Do not attempt to change, rename, or redefine fields 101 through 105. These fields are used by Perforce and should not be deleted or changed. Only use `p4 jobspec` to add new fields (106 and above) to your jobs.

  Field 101 is required by Perforce and cannot be renamed nor deleted.

  Fields 102 through 105 are reserved for use by Perforce client programs. Although it is possible to rename or delete these fields, it is highly undesirable to do so. Perforce client programs may 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 Perforce Visual Client, to display tooltips. Please make your comments complete and understandable.

- 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 jobspecs chapter of the *System Administrator's Guide* for an example of a customized jobspec.

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 label

Synopsis

Create or edit a label specification and its view.

Syntax

```
p4 [g-opts] label [ -f -t template ] labelname
p4 [g-opts] label -o [ -t template ] labelname
p4 [g-opts] label -d [ -f ] labelname
p4 [g-opts] label -i [ -f ]
```

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.

Form Fields

Field Name	Type	Description
`Label:`	Read-only	The label name as provided in the invoking command.
`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.
`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`.
`Description:`	Writable, optional	An optional description of the label's purpose.

Field Name	Type	Description
`Options:`	Writable	`locked` or `unlocked`. If the label is `locked`, the list of files tagged with the label cannot be changed with `p4 labelsync`.
`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. Please see the *Views* chapter for more information.

Options

`-d [-f]`	Delete the named label if it's `unlocked`. The `-f` flag forces the deletion even if the label is `locked`. (Deleting a `locked` label requires `admin` or `super` access.)
`-i`	Read the label definition from standard input without invoking the editor.
`-o`	Write the label definition to standard output without invoking the editor.
`-f`	Allow the `Update:` field's date to be set. Can be used with either the `-i` flag or the `-t` flag for the same purpose.
`-t` *`template`*	Copy label *`template`*'s view and options into the `View:` and `Options:` fields of this label.
`g-opts`	See the *Global Options* section.

Usage Notes

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

Related Commands

To tag revisions in your client workspace with a label	`p4 labelsync`
To list all labels known to the system	`p4 labels`
To create a label and tag files with the label	`p4 tag`

p4 labels

Synopsis

Display list of defined labels.

Syntax

```
p4 [g-opts] labels [-t] [-u user] [-e namefilter -m max] [file[revrange]]
```

Description

`p4 labels` lists all the labels known to the Perforce server 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 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 namefilter` option to limit the output to labels whose name matches the *namefilter* pattern.

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

Options

`-t`	Display the time as well as the date of the last update to the label.
`-u user`	List only labels owned by *user*.
`-e namefilter`	List only labels matching *namefilter*.
`-m max`	List only the first *max* labels.
g-opts	See the *Global Options* section.

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`.

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`
To create or edit a label specification	`p4 label`
To add, delete, or change the files included in a label	`p4 labelsync`
To view a list of files included in a label	`p4 files @`*`labelname`*

p4 labelsync

Synopsis

Synchronize a label with the contents of the current client workspace.

Syntax

`p4 [g-opts] labelsync [-a] [-d] [-n] [-q] -l labelname [file[revRange]...]`

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 workspace view is ignored; the specified revision is used instead of the revision existing in the workspace. If the file argument includes a revision range, then 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.
`-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 the *Global Options* section.

Usage Notes

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

- 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`.

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 license

Synopsis

Update or display the license file.

Syntax

```
p4 [g-opts] license [ -o ]
p4 [g-opts] license [ -i ]
```

Description

The `p4 license` command allows Perforce superusers to update or display the Perforce license file. This command requires that there is already a valid license file in the Perforce server root directory.

Use `p4 license` to add licensed users to a Perforce server without having to shut down the server and manually copy the license file into the server root.

Most new license files obtained from Perforce can be installed with `p4 license`, (or by copying over the existing license file) except for when the server IP address or port has changed. If either the server IP address or port number has changed, you must stop the Perforce Server, manually copy the license file into place, and restart the Server.

Without a valid license, the Perforce Server limits itself to either two users and five client workspaces (and unlimited files), or to an unlimited number of users and workspaces (but with a limit of 1000 files).

Options

`-o`	Display the current license file on the standard output.
`-i`	Read in a new license file from the standard input.
`g-opts`	See the *Global Options* section.

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 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 lock

Synopsis

Lock an opened file against changelist submission.

Syntax

```
p4 [g-opts] lock [-c changelist#] [file ...]
```

Description

Locking files prevents all 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#` flag is used, all open files matching the given file pattern in changelist `changelist#` are locked.

Options

`-c changelist#`	Lock only files included in changelist `changelist#`
`g-opts`	See the *Global Options* section.

Usage Notes

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

Related Commands

To unlock locked files	`p4 unlock`
To display all your open, locked files (UNIX)	`p4 opened | grep "*locked*"`

p4 lockstat

Synopsis

Report lock status of database tables.

Syntax

`p4 [g-opts] lockstat`

Description

The `p4 admin lockstat` command reports any database tables that are currently locked for a read or write operation.

Options

g-opts	See the *Global Options* section.

Usage Notes

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

p4 logger

Synopsis

Report changed jobs and changelists.

Syntax

```
p4 [g-opts] logger [-c sequence#] [-t countername]
```

Description

The `p4 logger` command is meant for use in external programs that call Perforce.

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; as this clears the log regardless of which counter name is specified, only one user can make use of this option.
`g-opts`	See the *Global Options* section.

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 intended for use by end users. It exists to support propagation of information to an external defect tracking system.

Related Commands

To list users who have subscribed to review particular files	`p4 reviews`
To set or read the value of a Perforce 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

Synopsis

Log in to a Perforce server by obtaining a ticket.

Syntax

```
p4 [g-opts] login [ -a -p ] [ user ]
p4 [g-opts] login [ -s ]
```

Description

The `p4 login` command authenticates a user and creates a ticket that represents a session with a Perforce server. Once authenticated, a user can access the Perforce server until either the ticket expires or until the user issues the `p4 logout` command.

By default, tickets are valid for 12 hours.

To obtain a ticket valid for all IP addresses (for instance, to use Perforce simultaneously on more than one machine), use `p4 login -a`. Users with tickets that are valid for all IP addresses still consume only one Perforce license.

Options

`-a`	Obtain a ticket that is valid for all IP addresses.
`-p`	Display the ticket, rather than storing it in the local ticket file.
`-s`	Display the status of the current ticket, if one exists.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- The default timeout value of 43200 seconds (12 hours) is defined on a per-group basis in the `p4 group` form.
- To create tickets that do not expire, set the timeout value to `unlimited` in the `p4 group` form.
- 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.

- Perforce 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.

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 machines.

Related Commands

To end a login session	`p4 logout`
To display tickets	`p4 tickets`

p4 logout

Synopsis

Log out of a Perforce server by removing or invalidating a ticket.

Syntax

```
p4 [g-opts] logout [ -a ]
```

Description

Log a user out of Perforce by removing a ticket on the user's workstation, or by invalidating the ticket on the server.

If you use `p4 logout -a`, the ticket remains in the ticket file, but is invalidated on the server: all users of the ticket are logged out simultaneously.

Options

`-a`	Log out all users of the ticket by invalidating the ticket on the server.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- 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.

Examples

`p4 logout`	Log out of Perforce by removing the local session ticket.
`p4 logout -a`	Log out of Perforce by removing the local session ticket and instructing the Perforce server 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 logstat

Synopsis

Report size of journal, error log, and/or audit log files.

Syntax

`p4 [g-opts] logstat`

Description

The `p4 admin logstat` command reports the sizes of the journal, error log (if it exists), and audit log (if it exists).

Options

g-opts	See the *Global Options* section.

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

Synopsis

Display the last block(s) of the error log.

Syntax

```
p4 [g-opts] logtail [ -b blocksize ] [ -s start_offset ] [ -m maxBlocks ]
```

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 "`... offset next_offset`", where *`next_offset`* is the offset in the logfile from which the next block of data is to be retrieved.

Options

`-s start`	The offset (from the beginning of the file), in bytes.
`-b blocksize`	The block size, in bytes. The default is 8192 bytes.
`-m maxBlocks`	The maximum number of blocks to output.
`g-opts`	See the *Global Options* section.

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 administering Perforce, see the *Perforce System Administrator's Guide*.

Examples

`p4 logtail -b 1024 -m 2`	Display the last two kilobytes of the log file, as two separate blocks of 1024 bytes each.

p4 monitor

Synopsis

Display Perforce process information

Syntax

```
p4 [g-opts] monitor show [ -a ] [ -e ] [ -l ]
p4 [g-opts] monitor terminate id
p4 [g-opts] monitor clear id
p4 [g-opts] monitor clear all
```

Description

`p4 monitor` allows a system administrator to observe what Perforce-related processes are running on the Perforce server machine. Each line of output consists of the following fields:

```
pid status owner hh:mm:ss command [args]
```

where *`pid`* is the process ID under UNIX (or thread ID under Windows), status is `R` or `T` depending on whether the process is running or marked for termination, *`owner`* is the Perforce user name of the user who invoked the command, *`hh:mm:ss`* is the time elapsed since the command was called, and *`command`* and *`args`* are the command and arguments as received by the Perforce server.

To list current process information, use `p4 monitor show`. All processes are listed, but only the command (for example, `sync`, `edit`, `submit`) is shown, without arguments. This form of `p4 monitor` requires `list` level access.

To show the list of arguments associated with each command, use the `-a` (arguments) flag or `-l` (long) flag. For additional information from the user environment, use the `-e` (environment) flag. These options require `admin` level access.

To mark a process for termination, use `p4 monitor terminate id`. This command requires `super` level access.

To remove an entry from the monitor table, use `p4 monitor clear id`. You can clear the entire table with `p4 monitor clear all`. Both of these commands require `super` level access.

To use `p4 monitor`, you must enable monitoring on the Perforce Server by setting the `monitor` configurable with `p4 configure`. You can control server process monitoring by setting the `monitor` counter to 0 (disable monitoring), 1 (enable monitoring of active processes), or 2 (enable monitoring of both active and idle processes).

Options

g-opts	See the *Global Options* section.
-a	Show all arguments associated with the process (for example, `edit file.c`, or `sync -f //depot/src/...`). Perforce user names are truncated to 10 characters, and each line is limited to a total of 80 characters of output.
-e	Show environment information including Perforce client application (if known), host IP address, and client workspace name.
-l	Show all arguments in long form; that is, without truncating user names or the list of command line arguments.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list, super`

- Processes marked as running continue to run to completion even if removed from the monitor table with `p4 monitor clear`.
- If a command terminates prematurely on the server side, it may be erroneously listed as running. Superusers can clear such processes with `p4 monitor clear`.
- The `p4 monitor terminate` command will not mark a process for termination unless the process has already been running for at least ten seconds.
- Some commands (for instance, `p4 submit`) invoke multiple processes. For example, `dm_CommitSubmit` or `dm_SubmitChange` may appear in the output of `p4 monitor` as two separate phases of the `p4 submit` command.
- Some commands, such as `p4 obliterate`, cannot be terminated.
- If you have enabled idle process monitoring for your server (by setting the `monitor` counter to 2), idle processes appear with a *`status`* of `R`, but with a *`command`* of `IDLE`.

Examples

`p4 monitor show`	Show Perforce processes 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 server 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 server monitoring	`p4 configure monitor=1`
To turn off server monitoring	`p4 configure monitor=0`

p4 move

Synopsis

Move (rename) a file from one location to another.

Syntax

```
p4 [g-opts] move [-c change] [-f] [-k] [-n] [-t filetype] fromFile toFile
```

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.

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.
`-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 Server.
`-n`	Preview the move that would be performed, without actually moving files.
`g-opts`	See the *Global Options* section.

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/d1/... //depot/d2/...`	Moving open files from directory `d1` to directory `d2`.

p4 obliterate

Synopsis

Removes files and their history from the depot.

Syntax

`p4 [g-opts] obliterate [ -y ] file[revRange] ...`

Warning

Use `p4 obliterate` ***with caution.*** This is the one of only two commands (along with the archive-purging option of `p4 archive`) in Perforce that actually remove file data.

The `p4 obliterate` command actually deletes the server's copy of a file's data, precluding any possibility of recovery. (By contrast, the `p4 delete` command merely marks the latest revision as deleted, but leaves earlier revisions intact in the depot.)

Description

`p4 obliterate` can be used by Perforce administrators to permanently remove files from the depot. All information about the files is wiped out, including the files' revisions, the files' metadata, and any records in any labels or client workspace records that refer directly to those files. Once `p4 obliterate` completes, it appears to the server as if the affected file(s) had never existed. Copies of files in client workspaces are left untouched, but are no longer recognized as being under Perforce control.

`p4 obliterate` requires at least one file pattern as an argument. To actually perform the obliteration, the `-y` flag is required; without it, `p4 obliterate` 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.

Options

`-y` *`filespec`*	Perform the obliterate operation. Without this flag, `p4 obliterate` merely reports what it would do.
`g-opts`	See the *Global Options* section.

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 Perforce 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. With archive depots, file history is preserved and available to all users, and file contents may be moved to offline or near-line storage. (This feature was introduced in Perforce 2010.2.)
- Obliterating files can alter the behavior of user commands. Syncing to an obliterated revision will remove the file from your client workspace, syncing to the head revision will either remove the file from your client workspace (if all revisions were obliterated), or provide you with the most recent non-obliterated revision of the file.
- Obliterating files in revision ranges can also change the behavior of scripts, as revision numbers of files may "skip" obliterated revisions. For instance, the output of `p4 filelog` after obliterating revisions `#2` and `#3` might look like this:

  ```
  ... #4 change 1276 edit on 2001/04/18 by user@dev1 (binary) 'Fixed'
  ... #1 change 1231 add on 2001/04/12 by user@dev1 (binary) 'First try'
  ```

  In this case, a developer using the `#4` in the first line of the output to assume the existence of four change descriptions in the output of `p4 filelog` would be in trouble.

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

Synopsis

List files that are open in pending changelists.

Syntax

```
p4 [g-opts] opened [-a -c change -C workspace -u user -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, or to show open files in all pending changelists, and to limit the number of files displayed.

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`, or `integrate`), and the type of the file. The output for each file looks like this:

```
depot-file#rev - action chnum 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; and
- *`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.

Options

`-a`	List opened files in any client workspace.
`-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`*.
`-m` *`max`*	List only the first *`max`* open files.

`-u user`	List only those files that were opened by `user`.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- Perforce does not prevent users from opening already open files; 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`.
- 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`.

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.

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 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 create a new, numbered changelist	`p4 change`
To view a list of changelists that meet particular criteria	`p4 changes`

p4 passwd

Synopsis

Change a user's Perforce password on the server.

Syntax

`p4 [g-opts] passwd [-O oldpassword] [-P newpassword] [user]`

Description

By default, user records are created without passwords, and any Perforce user can impersonate another by setting `P4USER` or by using the *globally-available* `-u` flag. To prevent another user from impersonating you, use `p4 passwd` to set your password to any string that doesn't contain the comment character `#`.

After you have set a password, you can authenticate with the password by providing it to the Perforce server program whenever you run any Perforce command. You can provide passwords to the Perforce server in one of three ways:

- Set the environment or registry variable `P4PASSWD` to the password value;
- Create a setting for `P4PASSWD` within the `P4CONFIG` file;
- Use the `-P password` flag on the Perforce client command line, for example:

  ```
  p4 -u ida -P idaspassword sync
  ```

Each of these three methods overrides the methods above it. Some of these methods may not be permitted depending on your server's security level.

On Windows clients connecting to servers at security levels 0 and 1, `p4 passwd` stores the password by using `p4 set` to change the local registry variable. (The registry variable holds only the encrypted MD5 hash, not the password itself.) On Windows clients connecting to servers at security levels 2 and 3, password hashes are neither stored in, nor read from, the registry.

You can improve security by using ticket-based authentication instead of password-based authentication. To authenticate with tickets instead of passwords, 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.

For more about how user authentication works, see the *System Administrator's Guide*.

Certain combinations of server security level and Perforce client software releases require users to set "strong" passwords. A password is considered strong if it is at least eight characters long, and at least two of the following are true:

- Password contains uppercase letters
- Password contains lowercase letters
- Password contains non-alphabetic characters.

For example, the passwords `a1b2c3d4`, `A1B2C3D4`, `aBcDeFgH` are considered strong. For information about how higher security levels work, see the *System Administrator's Guide*.

Options

`-O` *`oldpassword`*	Avoid prompting by specifying the old password on the command line. This option is not supported if your server is using security level 2 or 3.
`-P` *`newpassword`*	Avoid prompting by specifying the new password on the command line. This option is not supported if your server is using security level 2 or 3.
`user`	Superusers can provide this argument to change the password of another user.
`g-opts`	See the *Global Options* section.

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 passwd` command never sends plaintext passwords over the network; a challenge/response mechanism is used to send the encrypted password to the server.
- Passwords can contain spaces; command line use of such passwords requires quotes. For instance, to pass the password `my passw`, to Perforce, use `p4 -P "my passw" command`.
- If a user forgets his or her password, a Perforce superuser can reset it by specifying the username on the command line: `p4 passwd username`
- Passwords are truncated to 16 characters on all platforms.
- To delete a password, set the password value to an empty string. Depending on your server's security level, your server may not permit you to set a null password.

- If you are using ticket-based authentication, changing your password automatically invalidates all of your tickets and logs you out; that is, changing your password is equivalent to `p4 logout -a`.

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`

p4 ping

Synopsis

Test network performance.

Syntax

```
p4 [g-opts] ping [-f][-p pausetime][-c count][-t transmittime][-i
iterations][-s sendsize][-r receivesize]
```

Description

`p4 ping` simulates Perforce network traffic by sending messages from the Perforce Server to the Perforce client program 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.
`-t` *transmittime*	Transmit data for *transmittime* (maximum 6000) seconds.
`-i` *iterations*	Repeat the test for the specified number of *iterations*.
`-f`	Flood mode: the server transmits continuously, sending the next message without waiting for the Perforce client to confirm receipt of the prior message..
`-p` *pausetime*	Pause for *pausetime* seconds between tests, up to 120 seconds. To disable pausing, specify a pausetime of 0.
`-s` *sendsize*	Size of the server-to-client message, up to a maximum value of 10,000,000 bytes.
`-r` *receivesize*	Size of the client-to-server message, up to a maximum value of 10,000,000 bytes.
g-opts	See the *Global Options* section.

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 print

Synopsis

Print the contents of a depot file revision.

Syntax

```
p4 [g-opts] print [ -a ] [ -o outfile ] [ -q ] file[revRange] ...
```

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, and by default, only the highest revision in that range is listed. (To output each file at every revision within a specified revision range, use `p4 print -a`.)

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) flag.

Multiple file patterns can be included; all files matching any of the patterns are printed.

Options

`-a`	For each file, print all revisions within a specified revision range, rather than only the highest revision in the range.
`-q`	Suppress the one-line file header normally added by Perforce.
`-o` *`outfile`*	Redirect output to the specified output file on the local disk, preserving the same file type, attributes, and/or permission bits as the original file in the depot.
`g-opts`	See the *Global Options* section.

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 handy for files such as UNIX 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 protect

Synopsis

Control users' access to files, directories, and commands.

Syntax

```
p4 [g-opts] protect
p4 [g-opts] protect -o
p4 [g-opts] protect -i
```

Description

Use `p4 protect` to control Perforce permissions. You can use `p4 protect` to:

- Control which files particular users can access;
- Manage which commands particular users are allowed to use;
- Combine the two, allowing one user to write one set of files but only be able to read other files;
- Grant permissions to groups of users, as defined with `p4 group`;
- Grant or deny specific rights to users by using the `=read`, `=open`, `=write`, and `=branch` rights, without having to re-grant lesser permissions.
- Limit access to particular IP addresses, so that only users at these IP addresses can run Perforce.

In general, one typically grants an access level to a user or group, after which, if finer-grained control is required, one or more specific rights can then be denied.

The permission levels and access rights are:

Permission Level / Right	What the User Can Do
`list`	The user can access all Perforce metadata, but has no access to file contents. The user can run all the commands that describe Perforce objects, such as `p4 files`, `p4 client`, `p4 job`, `p4 describe`, `p4 branch`, etc.
`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.

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.
`=open`	If this right is denied, users cannot open files with `p4 add`, `p4 edit`, `p4 delete`, or `p4 integrate`.
`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 cannot submit open files.
`=branch`	If this right is denied, users cannot use files as a source for `p4 integrate`.
`review`	This permission is meant for external programs that access Perforce. 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 server 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`, and so on.

Form Fields

When you run `p4 protect`, Perforce displays a form with a single field, `Protections:`. Each permission is specified in its own indented line under the `Protections:` header, and has five values:

Column	Description
Access Level	One of the access levels `list`, `read`, `open`, `write`, `review`, or `super`, or one of the rights of `=read`, `=open`, `=write`, and `=branch`, as defined above
User or Group	Does this protection apply to a user or a group? The value of this field must be `user` or `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.

Column	Description
Host	The IP address. CIDR notation is supported. The `*` wildcard can also be used to refer to all IP addresses, but only when you are not using CIDR notation.
Depot File Path	The depot file path this permission is granted on, in Perforce *depot syntax*. The file specification can contain Perforce *wildcards*. To exclude this mapping from the permission set, use a dash (`-`) as the first character of this value.

When exclusionary mappings are not used, a user is granted the highest permission level listed in the union of all the mappings that match the user, the user's IP address, and the files the user is trying to access. In this case, the order of the mappings is irrelevant.

When exclusionary mappings are used, order is relevant: the exclusionary mapping overrides any matching protections listed above it in the table. No matter what access level is being denied in the exclusionary protection, all the access levels for the matching users, files, and IP addresses are denied.

If you use exclusionary mappings to deny access to an area of the depot to members of `group1`, but grant access to the same area of the depot to members of `group2`, a user who is a member of both `group1` and `group2` is either granted or denied access based on whichever line appears last in the protections table.

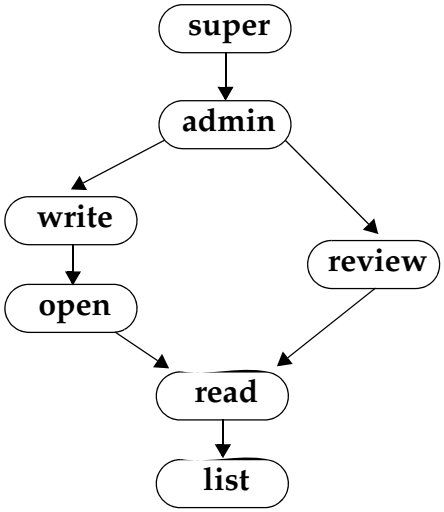
Options

`-i`	Read the form from standard input without invoking an editor.
`-o`	Write the form to standard output without invoking an editor.
`g-opts`	See the *Global Options* section.

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, as illustrated in this chart:



- 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:

```
admin       user      joe      *              //...
=write     user      joe       *              -//depot/build/...
=open      user      joe       *              -//depot/build/...
```

  In this example, user `joe` can perform administrative functions, which may include reading or listing files in `//depot/build/...`, but he is prohibited from opening files for edit (or submitting any changes he might have open.) He can, however, continue to create and modify files outside of the protected `//depot/build/...` area.

- Access levels determine which commands you can use. The following table lists the minimum access level required for each command. For example, because `p4 add` requires at least `open` access, you can run `p4 add` if you have `open`, `write`, `admin`, or `super` access.

Command	Access Level	Notes
`add`	`open`	
`admin`	`super`	
`annotate`	`read`	
`archive`	`admin`	
`branch`	`open`	The `-f` flag to override existing metadata or other users' data requires `admin` access.
`branches`	`list`	
`browse`	`none`	
`change`	`open`	The `-f` flag to override existing metadata or other users' data requires `admin` access.
`changes`	`list`	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
`client`	`list`	The `-f` flag to override existing metadata or other users' data requires `admin` access.
`clients`	`list`	
`configure`	`super`	
`copy`	`list`	`list` access to the source files; `open` access to the destination files.
`counter`	`review`	`list` access to at least one file in any depot is required to view an existing counter's value; `review` access is required to change a counter's value or create a new counter.
`counters`	`list`	
`dbschema`	`super`	
`dbstat`	`super`	
`delete`	`open`	
`depot`	`super`	The `-o` flag to this command, which allows the form to be read but not edited, requires only `list` access.

Command	Access Level	Notes
depots	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
describe	read	The -s flag to this command, which does not display file content, requires only list access.
diff	read	
diff2	read	
dirs	list	
edit	open	
export	super	
filelog	list	
files	list	
fix	open	
fixes	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
fstat	list	
grep	read	
group	super	The -o flag to this command, which allows the form to be read but not edited, requires only list access. The -a flag to this command requires only list access, provided that the user is also listed as a group owner.
groups	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
have	list	
help	none	
info	none	
integrate	open	The user must have open access on the target files and read access on the source files.
integrated	list	

Command	Access Level	Notes
job	open	The -o flag to this command, which allows the form to be read but not edited, requires only list access. The -f flag to override existing metadata or other users' data requires admin access.
jobs	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
jobspec	admin	The -o flag to this command, which allows the form to be read but not edited, requires only list access.
label	open	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot. The -f flag to override existing metadata or other users' data requires admin access.
labels	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
labelsync	list	
license	super	
lock	write	
lockstat	super	
login	list	
logout	list	
logstat	super	
logtail	super	
monitor	list	super access is required to terminate or clear processes, or to view arguments.
move	open	
obliterate	admin	
opened	list	
passwd	list	
ping	admin	
print	read	

Command	Access Level	Notes
protect	super	
protects	list	super access is required to use the -a, -g, and -u flags.
pull	super	
reopen	open	
replicate	super	
resolve	open	
resolved	open	
restore	admin	
revert	list	
review	review	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
reviews	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
set	none	
shelve	open	admin access is required to forcibly delete shelved files with p4 shelve -f -d
sizes	list	
submit	write	
sync	read	
tag	list	
tickets	none	
triggers	super	
typemap	admin	The -o flag to this command, which allows the form to be read but not edited, requires only list access.
unlock	open	The -f flag to override existing metadata or other users' data requires admin access.
unshelve	open	
user	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

Command	Access Level	Notes
`users`	`list`	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
`verify`	`admin`	
`where`	`list`	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

- When a new Perforce server is installed, anyone who wants to use Perforce is allowed to, and all Perforce users are superusers. The first time anyone runs `p4 protect`, the invoking user is made the superuser, and everyone else is given `write` permission on all files. Run `p4 protect` immediately after installation.
- In the course of normal operation, you'll primarily grant users `list`, `read`, `write`, and `super` access levels. The `open` and `review` access levels are used less often.
- Those commands that list files, such as `p4 describe`, will only list those files to which the user has at least `list` access.
- Some commands (for instance, `p4 change`, when editing a previously submitted changelist) take a `-f` flag that requires `admin` or `super` access.
- The `open` access level gives the user permission to change files but not submit them to the depot. Use this 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 aren't allowed to make permanent changes to the codeline.
- The `review` access level is meant for review daemons that need to access counter values.
- If you write a review daemon that requires both `review` and `write` access, but shouldn't have `super` access, grant the daemon both `review` and `write` access on two separate lines of the protections table.
- 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 Server, or by a virtual user named `remote`.
- For further information, see the *Protections* chapter of the *System Administrator's Guide*.

Examples

Suppose that user `joe` is a member of groups `devgroup` and `buggroup`, as set by `p4 group`, and the protections table reads as follows:

```
super     user      bill       *                  //...
write     group     devgroup   *                  //depot/...
write     group     buggroup   *                  -//depot/proj/...
write     user      joe        192.168.100.0/24   //...
```

Joe attempts a number of operations. His success or failure at each is described below:

From IP address...	Joe tries...	Results
`10.14.10.1`	`p4 print //depot/misc/...`	Succeeds. The second line grants Joe `write` access on these files; `write` access includes `read` access, and this protection isn't excluded by any subsequent lines.
`10.14.10.1`	`p4 print //depot/proj/README`	Fails. The third line removes all of Joe's permissions on any files in this directory. (If the second protection and the third protection had been switched, then the subsequent protection would have overridden this one, and Joe would have succeeded).
`192.168.100.123`	`p4 print //depot/proj/README`	Succeeds. Joe is sitting at an IP address from which he is granted this permission in the fourth line.
`192.168.100.123`	`p4 verify //depot/misc/...`	Fails. `p4 verify` requires `super` access; Joe doesn't have this access level no matter which IP address he's coming from.

Related Commands

To create or edit groups of users	`p4 group`
To list all user groups	`p4 groups`

p4 protects

Synopsis

Display protections in place for a given user, group, or path.

Syntax

```
p4 [g-opts] protects [ -a | -u user | -g group | -h host ] [-m] [ 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` flag to display lines for all users, or `-u user`, `-g group`, or `-h host` flags to display lines for a specific user, group, or host IP address.

Use the `-m` flag to display a one-word summary of the maximum applicable access level.

Options

`-a`	Displays protection lines for all users. This option requires `super` access.
`-g group`	Displays protection lines that apply to the named group. This option requires `super` access.
`-h host`	Displays protection lines that apply to the specified host IP address. This option requires `super` access.
`-u user`	Displays protection lines that apply to the named user. 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.)
`g-opts`	See the *Global Options* section.

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`, `-g`, `-u`

Related Commands

To edit the protections table	`p4 protect`

p4 pull

Synopsis

Replicate server metadata and versioned files from one Perforce Server to another.

Syntax

```
p4 [g-opts] pull [ -u | -l ] [ -i interval ]
```

Description

The `p4 pull` command instructs the current server (which must be a replica server) to retrieve either journal records or file contents from its target server as specified by `P4TARGET`.

The `-i` flag specifies a polling interval (in seconds) between updates. If `-i` is not specified, `p4 pull` runs for one polling interval and then exits.

To update file content as well as journal records, use the `-u` flag.

To display a list of pending file content transfers, use the `-l` flag.

Options

`-u`	Update files as well as journal records
`-l`	List pending file content transfers
`-i interval`	Specify a polling interval for content retrieval
g-opts	See the *Global Options* section.

Usage Notes

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

- Except for testing purposes, `p4 pull` is rarely run from the command line. Instead, use the `p4 configure` command to configure the server to start the `p4 pull` processes every time the replica server starts.
- A replica server can run multiple `p4 pull` commands against the same master server. For example, to replicate both metadata and file contents, run one `p4 pull` (without the `-u` flag) to replicate metadata, and at least one `p4 pull` (with the `-u` flag) to replicate versioned files.

- In most situations, server replication with `p4 pull` is preferable to `p4 replicate`.
- For more about configuring Perforce to run in a replicated environment, see the *Perforce System Administrator's Guide*.

Related Commands

To configure a Perforce 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`

p4 rename

Synopsis

Renaming files under Perforce.

Syntax

```
p4 [g-opts] move fromFile toFile
p4 [g-opts] integrate fromFile toFile
p4 [g-opts] delete fromFile
p4 [g-opts] submit fromFile toFile
```

Description

Use `p4 move` to move or rename files.

Prior to release 2009.1, moving files was accomplished by using `p4 integrate` to copy *`fromFile`* into a new *`toFile`*, using `p4 delete` to delete *`fromFile`*, and then using `p4 submit` to store these file changes in the depot.

You can rename multiple files with either method by including matching wildcards in *`fromFile`* and *`toFile`*.

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`*

Examples

`p4 edit //depot/d1/...` `p4 move //depot/d1/... //depot/d2/...`	Moving files from `d1` to `d2`. Files must be open for edit before they can be moved.

`p4 integ //depot/d1/... //depot/d2/...`
`p4 delete //depot/d1/...`
`p4 submit`
	Renaming a set of files prior to 2009.1:
	• `p4 integrate` copies all the files in the `d1` directory to the `d2` directory.
	• `p4 delete` deletes all files in `d1`.
	• `p4 submit` sends these changes to the depot in a single changelist.

Related Commands

Moving a file	`p4 move`
To copy a file and keep it under Perforce's control	`p4 integrate`
To delete a file from the depot	`p4 delete`
To submit changes to the depot	`p4 submit`

p4 reopen

Synopsis

Move opened files between changelists or change the files' type.

Syntax

`p4 [g-opts] reopen [-c changelist#] [-t filetype] file...`

Description

`p4 reopen` has two different but related uses:

- Use `p4 reopen -c changelist# file` to move an open file from its current pending changelist to pending changelist *changelist#*.
- Use `p4 reopen -c default` to move a file to the default changelist.
- Use `p4 reopen -t filetype` to change the type of a file.

If file patterns are provided, all open files matching the patterns are moved or retyped. The two flags can be combined to move a file and change its type in the same operation.

Options

`-c changelist# file`	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.
`-t filetype file`	When submitted, store file as type *filetype*. All subsequent revisions will be of that file type until the type is changed again. See the *File Types* section for a list of file types.
g-opts	See the *Global Options* section.

Usage Notes

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

Examples

`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.

Related Commands

To submit a changelist to the depot	`p4 submit`
To create a new changelist	`p4 change`
To remove a file from all pending changelists	`p4 revert`
To list opened files	`p4 opened`
To list all the files included in a changelist	`p4 opened -c` *`changelist#`*
To list all pending changelists	`p4 changes -p pending`
To open a file for edit under a particular pending changelist and as a particular type	`p4 edit -c` *`changelist#`* `-t` *`type`*
To open a file for add under a particular pending changelist and as a particular type	`p4 add -c` *`changelist#`* `-t` *`type`*
To implement 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 -t +l //`*`depotname`*`/...`

p4 replicate

Synopsis

Poll for journal changes on one Perforce Server for forwarding to another Perforce Server.

Syntax

```
p4 replicate [-j token] [-s statefile] [-i interval] [-k -x -R] [-J
prefix] [-o output] [command]
```

Description

This command polls for new journal entries from a Perforce 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

`-j token`	Specify a journal number or position token of the form *journalnum/byteoffset* from which to start replicating metadata. If this flag is specified, it overrides any state file specification.
`-s statefile`	Specify a state file which tracks the most recent journal position.
`-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 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.
`-x`	Exit the `p4 replicate` command when journal rotation is detected.
`-R`	The `-R` flag 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`.
`-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.
g-opts	See the *Global Options* section.

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 server metadata (but not archived files), or when you need to perform filtering operations on server metadata. In most situations, server replication with `p4 pull` is preferable to `p4 replicate`.
- For more information, see the *System Administrator's Guide* and the following Perforce Knowledge Base article:

  `http://kb.perforce.com/article/1099`

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 resolve

Synopsis

Resolve conflicts between file revisions.

Syntax

`p4 [g-opts] resolve [-af -am -as -at -ay -dflag -f -n -o -t -v] [file...]`

Description

Use `p4 resolve` to combine the contents of two files or file revisions into a single file revision in your workspace. 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 submits` `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 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.

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.

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 Perforce 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.

The interactive `p4 resolve` dialog presents the following options. Note that the dialog options are not the same as the command line flags.

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`e`	edit merged	Edit the preliminary merge file generated by Perforce.	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

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`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 base theirs yours merge` 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
`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.	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`ae`	accept edit	If you edited the file (i.e., 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
`a`	accept	Keep Perforce'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 Perforce's generated merge file, since Perforce 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, Perforce 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.

Options

`-am`
`-af`
`-as`
`-at`
`-ay`
	Skip the resolution dialog, and resolve the files automatically as follows:
	• `-am`: Automatic Mode. Automatically accept the Perforce-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 flag with caution, as the file in the client workspace will be overwritten!
	• `-as`: Safe Accept. If either *yours* or *theirs* is different from *base*, (and the changes are in common) accept that revision. 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.

`-d`*`flag`*	When merging files, ignore specified differences in whitespace or line-ending convention. (If you use these flags, 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.
`-n`	List the files that need resolving without actually performing the resolve.
`-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.
`g-opts`	See the *Global Options* section.

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:
  - Integrating the file with `p4 integrate`.
  - 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 only benefit of scheduling files for resolve with `p4 sync` instead of a failed submit is that the submit will not fail).

  When `p4 resolve` is run with no file arguments, it operates on all files in the client workspace that have been scheduled for resolve.
- 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.

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 resolved

Synopsis

Display a list of files that have been resolved but not yet submitted.

Syntax

`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 the *Global Options* section.

Usage Notes

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

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

Synopsis

Restore old archived revisions from an archive depot.

Syntax

```
p4 [g-opts] restore [ -n ] -D depot file[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.

Options

`-n`	Do not restore files; report on revisions that would be restored.
`-D depot`	Specify an archive depot from which files are to be restored.
g-opts	See the *Global Options* section.

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` flag.

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`

p4 revert

Synopsis

Discard changes made to open files.

Syntax

```
p4 [g-opts] revert [ -a -n -k -c changelist# ] file...
```

Description

Use `p4 revert` to discard changes made to open files, reverting them to the revisions last `p4 sync`ed from the depot. This command also removes the reverted files from the pending changelists with which they're associated.

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`, Perforce leaves the client workspace files intact. When you revert files you've opened with `p4 integrate`, Perforce removes the files from the client workspace.

Options

`-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: • open for edit but have unchanged content and unchanged filetype; or • open for integrate via `p4 integrate` and have not yet been resolved with `p4 resolve`.
`-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.
`-k`	Keep workspace files; the file(s) are removed from any changelists, and the server records the files as being no longer open, but the file(s) are unchanged in the client workspace.
`-c changelist#`	Reverts only those files in the specified changelist.
`g-opts`	See the *Global Options* section.

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 Perforce 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 workspace view.

  You don't need to specify a file argument when using the `-a` flag.

- 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

`p4 revert //...`	Revert every file you have open, in every one of your pending changelists, to its pre-opened state.
`p4 revert -c default //...`	Revert every file open in the default changelist to its pre-opened state.
`p4 revert -n *.txt`	Preview a reversion of all open `.txt` files in the current directory, but don't actually perform the revert.
`p4 revert -c 31 *.txt`	Revert all `.txt` files in the current directory that were open in changelist 31.
`p4 revert -a`	Revert all unchanged files. This command is often used before submitting a changelist.

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 Perforce thinks you have, overwriting any unopened, writable files in the process.	`p4 sync -f`

p4 review

Synopsis

List all submitted changelists above a provided changelist number.

Syntax

`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 changelist# username <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 Perforce 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 Perforce. The Perforce change review daemon, which is described in the *Perforce System Administrator's Guide*, and is available from our Web site, uses `p4 review`.

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 the *Global Options* section.

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 Perforce.
- 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 Perforce 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

Synopsis

List all the users who have subscribed to review particular files.

Syntax

```
p4 [g-opts] reviews [-c changelist#] [file...]
```

Description

The `p4 reviews` command is intended for use in external programs that call Perforce.

Users subscribe to review files by providing file patterns in the `Reviews:` field in their `p4 user` form.

`p4 reviews -c changelist#` lists each user who has subscribed to review any files included in the submitted changelist *`changelist#`*. 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 changelist#`	List all users who have subscribed to reviews any files included in submitted changelist *`changelist#`*.
`g-opts`	See the *Global Options* section.

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 changelist# file...` 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 users` form, which has a very specific meaning.

It is possible to enter values in the `Reviews:` field that mean something originally unintended by Perforce in order 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 anytime a new job is submitted. Since there's nothing built into Perforce that allows users to subscribe to review jobs, we co-opt a single line of the `Reviews:` field: Perforce sends 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 Perforce counter	`p4 counter`
To read full information about a particular changelist	`p4 describe`

p4 set

Synopsis

Set Perforce variables in the Windows registry.

Syntax

```
p4 [g-opts] set [ -s ] [ -S svcname ] [ var=[value] ]
```

Description

The Perforce client and server require the use of certain system variables.

On Windows, you can set the values of these variables in the registry with `p4 set`; on other operating systems, Perforce uses environment variables for the same purpose.

To set the value of a registry variable for the current user, use `p4 set var=value`. Windows administrators can use `p4 set -s var=value` to set the registry variable's default values for all users on the local machine.

Windows administrators running the Perforce server as a service can set variables used by the service (for instance, `P4JOURNAL` and others) with `p4 set -S svcname var=value`.

To unset the value for a particular variable, leave `value` empty.

To view a list of the values of all Perforce variables, use `p4 set` without any arguments. On UNIX, this displays the values of the associated environment variables. On Windows, this displays either the MS-DOS 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).

`p4 set` can be used on non-Windows operating systems to view the values of variables, but if you try to use `p4 set` to set variables on non-Windows operating systems, Perforce displays an error message.

Options

`-s`	Set the value of the registry variables for the local machine. Without this flag, `p4 set` sets the variables in the `HKEY_CURRENT_USER` hive; when you use the `-s` flag, the variables are set in the `HKEY_LOCAL_MACHINE` hive. These locations are 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 administrator privileges to do this.
`g-opts`	See the *Global Options* section.

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 Perforce variables in the *Environment Variables* section of this manual.
- Changes to registry values under Windows affect the local machine only; an administrator setting `P4JOURNAL` for a Perforce Windows service must be present at the machine running the service.
- On Windows, you can override the values of the registry keys in any of three ways:
  - Environment variables with the same names have precedence;
  - Values within `P4CONFIG` files have precedence over both of these;
  - For the Perforce service, configurables set with `p4 configure` override all environment variables, including registry entries set with `p4 set -S`;
  - The *global option* flags, specified on the command line, have the highest precedence.
- If you're working in a UNIX-like environment on a Windows machine (e.g. Cygwin), use environment variables instead of `p4 set`. (In such cases, the Perforce Command-Line Client behaves just as though it were in a UNIX environment.)

Examples

`p4 set`	On all platforms, display a list of Perforce variables without changing their values.
`p4 set P4MERGE=`	On Windows, unset the value of `P4MERGE`.
`p4 set P4PORT=tea:1666`	On Windows, set a registry variable telling Perforce client programs to connect to a Perforce server at host `tea`, port `1666`. The variable would be set only for the current local user. .
`p4 set -s P4PORT=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 NT service `p4svc`, instruct `p4s.exe` to listen on port 1666 for incoming connections from Perforce client programs. 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

Synopsis

Store files from a pending changelist in the depot, without submitting them.

Syntax

```
p4 shelve [files ...]
p4 shelve -i [-f | -r ]
p4 shelve -r -c changelist
p4 shelve -c changelist [-f] [file ...]
p4 shelve -d -c changelist [-f] [file ...]
```

Description

Shelving is the process of temporarily storing work in progress on a Perforce Server without submitting a changelist. Shelving is useful when you need to perform multiple development tasks (such as interruptions from higher-priority work, testing across multiple platforms) on the same set of files, 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.

After shelving files, you can revert or modify them in your client workspace, and 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 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.

Options

`-f`	Force the overwriting of any existing shelved files in a pending changelist with the contents of their client workspace copies. Perforce administrators can use this flag with `-d` to force the discarding of shelved files in a specified changelist.
`-i`	Read a changelist description from standard input. Input must be in the same format used by the `p4 shelve` form. (When modifying an existing changelist with shelved files, this option also implies `-c`).
`-c` *`changelist`*	Specify the pending changelist in which shelved files are to be created, discarded, or modified. Only the user and client workspace that owns the pending changelist can add or modify its shelved files. (Administrators can use `-f` to discard files.)
`-d`	Discard the shelved files in the specified changelist so that they are no longer available for `p4 unshelve` operations. Only the user and workspace that owns the pending changelist can discard its shelved files. (Administrators can use `-f` to discard files.)
`-r`	Replace all shelved files in the changelist with the files that are opened in your workspace. (To replace the files in the changelist with files open at a specific change number, use the `-c` or `-i` options.) Only the user and workspace associated with a pending changelist can replace its shelved files.
`-n`	Preview the results of the shelve operation without actually shelving the files (that is, without overwriting any existing shelved files.)
`g-opts`	See the *Global Options* section.

Usage Notes

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

- In order to add a file to a pre-existing shelve, the file must first be opened in the shelve's changelist.

Related Commands

To restore shelved files into a workspace	`p4 unshelve`

p4 sizes

Synopsis

Display size information for files in the depot.

Syntax

```
p4 [g-opts] sizes [-a] [-S] [-s|-z] [-b blocksize] [-m max] file[revRange]
...
```

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 `-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 Perforce server, 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.
`-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.
`-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.
`-z`	When calculating size information, exclude lazy copies.
`g-opts`	See the *Global Options* section.

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.
`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 Perforce Server 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 submit

Synopsis

Send changes made to open files to the depot.

Syntax

```
p4 [g-opts] submit [-r] [-s] [-f submitoption]
p4 [g-opts] submit [-r] [-s] [-f submitoption] files
p4 [g-opts] submit [-r] [-f submitoption] -d description
p4 [g-opts] submit [-r] [-f submitoption] -d description file
p4 [g-opts] submit [-r] [-f submitoption] -c changelist#
p4 [g-opts] submit -i [-r] [-s] [-f submitoption]
```

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 in the client workspace copy until the changelist is sent to the depot with `p4 submit`.

By default, files are opened within the default changelist, but new numbered changelists can be created with `p4 change`. To submit the default changelist, use `p4 submit`; to submit a numbered changelist, use `p4 submit -c` *`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 or registry 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.

`p4 submit` works atomically: either all the files listed in the changelist are saved in the depot, or none of them are. `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, or require resolution and remain unresolved.

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.

To supply a changelist description from the command line, use the `-d` flag. No change description dialog is presented. The `-d` flag works only with the default changelist, not with numbered changelists.

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 Perforce 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`, and `submitted` when its contents have been stored in the depot with `p4 submit` .
`Description:`	Writable	Textual description of changelist. This value *must* be changed.
`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

`-c changelist#`	Submit changelist number *`changelist#`*. Changelists are assigned numbers either manually by the user with `p4 change`, or automatically by Perforce when submission of the default changelist fails.
`-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 jobs to be added, nor for the default changelist to be modified.
`-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 Perforce. • `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. • `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. • `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 at that used by the `p4 submit` form.

`-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.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- A file's location within the depot is determined by intersection of its locations in the client workspace with the client view as set within the `p4 client` form.
- The atomic nature of `p4 submit` allows files to be grouped in changelists according to their purpose. For example, a single changelist might contain changes to three files that fix a single bug.
- When used with a numbered changelist, `p4 submit` does not display a form. To change the description information for a numbered changelist, use `p4 change -c` *`changelist#`*.
- A single file pattern can be specified as a parameter to a `p4 submit` of the default changelist. This file pattern limits which files in the default changelist are included in the submission; files that don't match the file pattern are moved to the next default changelist.

  The file pattern parameter to `p4 submit` can only be used when submitting the default changelist.

Examples

`p4 submit`	Submit the default changelist. The user's revisions of the files in this changelist are stored in the depot.
`p4 submit -c 41`	Submit changelist 41.
`p4 submit *.txt`	Submit only those files in the default changelist that have a suffix of `.txt`. Move all the other files in the default changelist to the next default changelist.
`p4 submit -d "header files" *.h`	Submit only those files in the default changelist that have a suffix of `.h`, with a description of `header files`. No changelist form is displayed. Move all the other files in the default changelist to the next default changelist.

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 sync

Synopsis

Copy files from the depot into the workspace.

Syntax

```
p4 [g-opts] sync [-f] [-k] [-n] [-q] [-m max] [file[revRange]...]
p4 [g-opts] sync [-n] [-p] [-q] [-m max] [file[revRange]...]
```

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 only if it meets all of the following criteria:

- The file must be visible through the *client workspace view*;
- It must not already be opened by `p4 edit`, `p4 delete`, `p4 add`, or `p4 integrate`;
- It must not already exist in the client workspace at its latest revision (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 selected by 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.

The newly synced files are not available for editing until opened with `p4 edit` or `p4 delete`. Newly synced files are read-only; `p4 edit` and `p4 delete` make the files writable. Under normal circumstances, do not use your operating system's commands to make the files writable; instead, use Perforce to do this for you.

Options

`-f`	Force the sync. Perforce 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 flag does not affect open files, but it *does* override the `noclobber` client option.
`-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.
`-m max`	Sync only the first *`max`* files specified.
`-n`	Display the results of the sync without actually performing the sync. This lets you make sure that the sync does what you think it does before you do it.
`-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.
`-q`	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- If the client view has changed since the last sync, the next sync removes from the client workspace those files that are no longer visible through the client 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 client view 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 Perforce's control, `p4 sync` will not normally overwrite those files. If the `clobber` option in the `p4 client` form has been turned on, however, these files will be overwritten.

- `p4 flush` is an alias for `p4 sync -k`. All of the warnings that apply to `p4 flush` also apply to `p4 sync -k`.
- The `-m max` flag is useful when combined with the `-n` flag 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 conceivably be thousands of lines long, all of which would be discarded.

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 unlabeled 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 @2001/06/24`	Bring the workspace into sync with the depot as of midnight, June 24, 2001. (That is, include all changes made during June 23.)
`p4 sync status%40june1st.txt`	Sync a filename containing a Perforce 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 300.
`p4 sync file.c#none`	Sync to the nonexistent revision of `file.c`; the file is deleted from the workspace.
`p4 sync ...#none`	Sync to the nonexistent revision of all files; all files in the workspace (that are under Perforce 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 tag

Synopsis

Tag files with a label.

Syntax

```
p4 [g-opts] tag [ -d -n ] -l labelname file[revRange]...
```

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.
`-n`	Display what `p4 tag` would do without actually performing the operation.
`-l labelname`	Specify the label to be applied to file revisions
`g-opts`	See the *Global Options* section.

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.

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`*.

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`
To create a label and tag files with the label	`p4 tag`

p4 tickets

Synopsis

Display all tickets granted to a user by `p4 login`.

Syntax

`p4 [g-opts] tickets`

Description

The `p4 tickets` command lists all tickets stored in the user's ticket file.

Options

g-opts	See the *Global Options* section.

Usage Notes

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

- 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.

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

Synopsis

Edit a list of scripts to be run conditionally whenever changelists are submitted, forms are updated, or when integrating Perforce with external authentication mechanisms.

Syntax

```
p4 [g-opts] triggers
p4 [g-opts] triggers -i
p4 [g-opts] triggers -o
```

Description

Perforce *triggers* are user-written scripts that are called by a Perforce 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. Upon failure, the script's standard output (not error output) is used as the text of the failed operation's error message.

Perforce supports a wide range of trigger types, divided into six categories.

- *Changelist submission triggers* (`change-submit`, `change-content`, and `change-commit`) are fired when users submit changelists.
- *Shelve triggers* (`shelve-submit`, `shelve-commit`, and `shelve-delete`) are similar to changelist submission triggers, but fire when users shelve files or delete shelved files.
- *Fix triggers* (`fix-add` and `fix-delete`) are fired when fixes are added or deleted from changelists by means of the `p4 fix`, `p4 submit`, or `p4 change` commands.
- *Form triggers* (`form-save`, `form-out`, `form-in`, `form-commit`, and `form-delete`) are fired when users generate or modify data in form specifications.
- *Authentication triggers* (`auth-check`, `auth-set`, and `service-check`) are used for situations where administrators wish to integrate Perforce with external authentication mechanisms such as LDAP or Active Directory.
- *Archive triggers* (`archive`) are used to work with content archived outside the Perforce-managed repository.

Use the `change-submit` trigger type to create triggers that fire after changelist creation, but before files are transferred to the server. Because `change-submit` triggers fire before files are transferred to the server, submit triggers cannot access file contents. Submit triggers are useful for integration with reporting tools or systems that do not require access to file contents.

Use the `change-content` trigger type to create triggers that fire after changelist creation and file transfer, but prior to committing the submit to the database.

Use the `change-commit` trigger type to create triggers that fire after changelist creation, file transfer, and changelist commission to the database. Use commit triggers for processes that assume (or require) the successful submission of a changelist.

Even when a `change-submit` or `change-content` trigger script succeeds, the submit can fail because of subsequent trigger failures, or for other reasons. Use `change-submit` and `change-content` triggers only for validation, and use `change-commit` triggers or daemons for operations that are contingent on the successful completion of the submit.

To configure Perforce to run trigger scripts when users fix jobs, use *fix triggers*: these are triggers of type `fix-add` and `fix-delete`. Use fix triggers to execute commands when users add or remove fixes from changelists.

To configure Perforce to run trigger scripts when users edit specification forms, use *form triggers:* these are triggers of type `form-save`, `form-in`, `form-out`, `form-commit`, and `form-delete`. Use form triggers to generate customized specification values for users, validate data submitted in forms, to notify other users of attempted changes to the values stored in specification forms, and to otherwise interact with process control and management tools.

To use an external password authentication manager (such as LDAP or Active Directory) for Perforce users, use *authentication triggers* (`auth-check` and `auth-set`). Use the `%user%` variable to pass the user's username in the command for the script. Passwords typed by the user as part of the authentication process are supplied to authentication scripts as standard input; never on the command line. For further information, see the *System Administrator's Guide*.

The `service-check` trigger works exactly like an `auth-check` trigger, but applies only to users whose `Type:` has been set to `service`. The `service-check` trigger type is used by Perforce administrators who want to use LDAP to authenticate other Perforce servers in replicated and other multiserver environments.

Archive triggers (`archive`) are a special case, and are reserved for storing, managing, or generating content archived outside of the Perforce repository.

Triggers are run in the order listed in the table; if a trigger script fails for a specified type, subsequent trigger scripts also associated with that type are not run.

To use the same trigger script with multiple file patterns, list the same trigger multiple times in the trigger table. Use exclusionary mappings to prevent files from activating the trigger script; the order of the trigger entries matters, just as it does when exclusionary mappings are used in views. If a particular trigger name and type is listed multiple times, only the script corresponding to the first use of the trigger name and type is activated.

Form Fields

The `p4 triggers` form contains a single `Triggers:` field. Like other Perforce forms, indent each row under the `Triggers:` field with tabs. Each row holds four values:

Field	Meaning
name	The user-defined name of the trigger. A run of the same trigger name on contiguous lines is treated as a single trigger, so that multiple *path*s can be specified. In this case, only the *command* of the first such trigger line is used.
type	Trigger types are divided into six subtypes: changelist submission triggers, shelve triggers, fix triggers, form triggers, authentication triggers, and archive triggers. Changelist submission triggers: • `change-submit`: Execute a changelist trigger after changelist creation, but before file transfer. Trigger cannot access file contents. • `change-content`: Execute a changelist trigger after changelist creation and file transfer, but before file commit. To obtain file contents, use commands such as `p4 diff2`, `p4 files`, `p4 fstat`, and `p4 print` with the revision specifier `@=`*change*, where *change* is the changelist number of the pending changelist as passed to the script in the `%changelist%` variable. • `change-commit`: Execute a changelist trigger after changelist creation, file transfer, and changelist commit. Shelve triggers: • `shelve-submit`: Execute a pre-shelve trigger after changelist has been created and files locked, but prior to file transfer. • `shelve-commit`: Execute a post-shelve trigger after files are shelved. • `shelve-delete`: Execute a shelve trigger prior to discarding shelved files. Fix triggers: The special variable `%jobs%` is available for expansion; it expands to one argument for every job listed on the `p4 fix` command line (or in the `Jobs:` field of a `p4 change` or `p4 submit` form), and must therefore be the last argument supplied to the trigger script. • `fix-add`: Execute fix trigger prior to adding a fix. • `fix-delete`: Execute fix trigger prior to deleting a fix.

Field	Meaning
	Form triggers: • `form-save`: Execute a form trigger after the form contents are parsed, but before the contents are stored in the Perforce database. The trigger cannot modify the form specified in `%formfile%` variable. • `form-out`: Execute form trigger upon generation of form to end user. The trigger can modify the form. • `form-in`: Execute form trigger on edited form before contents are parsed and validated by the Perforce server. The trigger can modify the form. • `form-delete`: Execute form trigger after the form contents are parsed, but before the specification is deleted from the Perforce database. Trigger cannot modify form. • `form-commit`: Execute form trigger after the form has been committed for access to automatically-generated fields such as jobname, dates, etc. For job forms, this trigger is run by `p4 job` as well as `p4 fix` (after the status is updated). The `form-commit` trigger has access to the new job name created by `p4 job`; any `form-in` and `form-save` triggers are run before the job name is created. For job forms, this trigger is also run by `p4 change` (if a job is added or deleted by editing the `Jobs:` field of the changelist), and `p4 submit` (for any jobs present in the `Jobs:` field of the changelist). In these cases, the special variable `%action%` is available for expansion on the job `form-commit` trigger command line. The trigger cannot modify the form. Authentication triggers: • `auth-check`: Execute an authentication check trigger to verify a user's password against an external password manager during login, or when setting a new password If an `auth-check` trigger is present, the Perforce `security` counter (and any associated password strength requirement) is ignored, as authentication is now controlled by the trigger script. • `auth-set`: Execute an authentication set trigger to send a new password to an external password manager. • `service-check`: Execute a trigger to verify the password of a service user, rather than a standard user. Service check triggers work in the same way that `auth-check` triggers do. You must restart the Perforce server after adding an `auth-check` or `service-check` trigger.

Field	Meaning
	Archive triggers: • `archive`: Execute the script when a user accesses any file with a filetype containing the `+X` filetype modifier. The script is run once per file requested. For `read` operations, scripts should deliver the file to the user on standard output. For `write` operations, scripts receive the file on standard input.
`path`	For changelist and shelve triggers (`change-submit`, `change-content`, `change-commit`, `shelve-submit`, `shelve-commit`, and `shelve-delete`), a file pattern in depot syntax. When a user submits a changelist that contains any files that match this file pattern, the script linked to this trigger is run. Use exclusionary mappings to prevent triggers from running on specified files. For fix triggers (`fix-add` or `fix-delete`), use `fix` as the path value. For form triggers (`form-save`, `form-out`, `form-in`, `form-commit`, or `form-delete`), the name of the type of form, (one of `branch`, `change`, `client`, `depot`, `group`, `job`, `label`, `protect`, `spec`, `triggers`, `typemap`, or `user`). For authentication triggers (`auth-check` or `auth-set`), use `auth` as the path value.

Field	Meaning
command	The command for the Perforce server to run when a matching *path* applies for the trigger type. Specify the command in a way that allows the Perforce server account to locate and run the command. The command must be quoted, and can take the variables specified below as arguments. For `change-submit` and `change-content` triggers, changelist submission continues if the trigger script exits with 0, or fails if the script exits with a nonzero value. For `change-commit` triggers, changelist submission succeeds regardless of the trigger script's exit code, but subsequent `change-commit` triggers do not fire if the script exits with a nonzero value. For `form-in`, `form-out`, `form-save`, and `form-delete` triggers, the data in the specification becomes part of the Perforce database if the script exits with 0. Otherwise, the database is not updated. The `form-commit` trigger type never rejects a change; it exists primarily so that scripts can access a job number (from the `%formname%` variable) during the process of job creation. For `fix-add` and `fix-delete` triggers, fix addition or deletion continues if the trigger script exits with 0, or fails if the script exits with a nonzero value. For `auth-check` and `service-check` triggers (fired by `p4 login` from standard users and service users respectively), the user's typed password is supplied to the trigger command as standard input. If the trigger executes successfully, the Perforce ticket is issued. The user name is available as `%user%` to be passed on the command line. For `auth-set` triggers, (fired by `p4 passwd`, but only after also passing an `auth-check` trigger check) the user's old password and new password are passed to the trigger as standard input. The user name is available as `%user%` to be passed on the command line.

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 the *Global Options* section.

Usage Notes

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

Warning! Never use a Perforce command in an `out` trigger that fires the same `out` trigger, or infinite recursion will result. For example, never run `p4 job -o` from within an `out` trigger script that fires on `job` specifications.

Warning! If you write a trigger that fires on trigger forms, and the trigger fails in such a way that the `p4 triggers` command no longer works, the only recourse is to remove the `db.triggers` file in the server root directory.

- To pass arguments to the trigger script, use the following variables:

Argument	Description	Available for type
`%action%`	Either null or a string reflecting an action taken to a changelist or job. For example, "`pending change 123 added`" or "`submitted change 124 deleted`" are possible `%action%` values on `change` forms, and "`job000123 created`" or "`job000123 edited`" are possible `%action%` values for `job` forms.	`form-commit`
`%changelist%` `%change%`	The number of the changelist being submitted. The abbreviated form `%change%` is equivalent to `%changelist%`. A `change-submit` trigger is passed the pending changelist number; a `change-commit` trigger receives the committed changelist number.	`change-submit`, `change-content`, `change-commit`, `fix-add`, `fix-delete`, `form-commit`, `shelve-commit`
`%changeroot%`	The root path of files submitted	`change-commit`
`%client%`	Triggering user's client workspace name.	all but `archive`
`%clienthost%`	Hostname of the client.	all but `archive`
`%clientip%`	The IP address of the client.	all but `archive`
`%jobs%`	A string of job numbers, expanded to one argument for each job number specified on a `p4 fix` command or for each job number added to (or removed from) the `Jobs:` field in a `p4 submit`, or `p4 change` form.	`fix-add` `fix-delete`
`%oldchangelist%`	If a changelist is renumbered on submit, this variable contains the old changelist number.	`change-commit`
`%serverhost%`	Hostname of the Perforce server.	all but `archive`

Argument	Description	Available for type
`%serverip%`	The IP address of the server.	all but `archive`
`%serverport%`	The IP address and port of the Perforce server, in the format *`ip_address:port`*.	all but `archive`
`%serverroot%`	The `P4ROOT` directory of the Perforce server.	all but `archive`
`%user%`	Perforce username of the triggering user.	all but `archive`
`%formfile%`	Path to temporary form specification file. To modify the form from an `in` or `out` trigger, overwrite this file. The file is read-only for triggers of type `save` and `delete`.	`form-commit`, `form-save`, `form-out`, `form-in`, `form-delete`
`%formname%`	Name of form (for instance, a branch name or a changelist number).	`form-commit`, `form-save`, `form-out`, `form-delete`
`%formtype%`	Type of form (for instance, `branch`, `change`, and so on).	`form-commit`, `form-save`, `form-out`, `form-in`, `form-delete`
`%op%`	Operation: `read`, `write`, or `delete`	`archive`
`%file%`	Path of archive file based on depot's `Map:` field. If the `Map:` field is relative to `P4ROOT`, the `%file%` is a server-side path relative to `P4ROOT`. If the `Map:` field is an absolute path, the `%file%` is an absolute server-side path.	`archive`
`%rev%`	Revision of archive file	`archive`

- If your trigger script needs to know what files were (or are about to be) submitted in the changelist, use the command `p4 opened -ac changelist`.
- Pre-submit trigger scripts cannot access submitted file contents from the server, because at the time a pre-submit trigger runs, file contents have not yet been transferred to the server.
- Perforce commands in trigger scripts are always run by a specific Perforce user. If no user is specified, an extra Perforce license for a user named `SYSTEM` (or on UNIX, the user that owns the `p4d` process) is assumed. To prevent this from happening:
  - Pass a `%user%` argument to the script that calls each Perforce command to ensure that each command is called by. For example, if Joe submits a changelist that activates trigger script `trigger.pl`, and `trigger.pl` calls the `p4 changes` command, the script can run the command as `p4 -u %user% changes`.
  - Set `P4USER` for the account that runs the trigger script to the name of an existing user. (If your Perforce server is installed as a service under Windows, note that Windows services cannot have a `P4USER` value; on Windows, you must therefore pass a user value to each command as described above.)
- For the four form trigger types (`form-in`, `form-out`, `form-save`, and `form-delete`), the `%formname%` variable is unset on job creation. This limitation is due to the fact that a job's name is unknown to the server until after job creation.

  To access a job's name *during* job creation, use the `form-commit` trigger; it is the only trigger type that has access to a job's name (in the `%formname%` variable) during the process of job creation.

  After job creation, subsequent user changes to a job correctly set `%formname%` for use by form trigger scripts.
- Trigger types were renamed in Release 2005.2. The following old trigger type names will continue to work but are deprecated:

Old trigger type	New trigger type (as of 2005.2)
`submit`	`change-submit`
`content`	`change-content`
`commit`	`change-commit`
`out`	`form-out`
`in`	`form-in`
`save`	`form-save`
`delete`	`form-delete`

- As of Release 2007.3, trigger standard output is passed to the client program on both success and failure of the trigger script. Prior to this, standard output was only passed to the client program upon failure of a trigger.
- You must specify the name of the trigger script or executable in ASCII, even when the server is running in Unicode mode and passes arguments to the trigger script in UTF8.

Examples

Suppose that the trigger table consists of the following entries:

```
Triggers:
    trig1 change-submit //depot/dir/... "/usr/bin/s1.pl %changelist%"
    trig2 change-submit //depot/dir/file "/usr/bin/s2.pl %user%"
    trig1 change-submit -//depot/dir/z* "/usr/bin/s1.pl %user%"
    trig1 change-submit //depot/dir/zed "/usr/bin/s3.pl %client%"
```

Both the first and third lines call the script `/bin/s1.pl %changelist%`, because the first occurrence of a particular trigger name determines which script is run when the trigger name is subsequently used.

No triggers are activated if someone submits file `//depot/dir/zebra`, because the third line excludes this file. If someone submits `//depot/dir/zed`, the `trig1` script `/usr/bin/s1.pl %changelist%` is run: although the fourth line overrides the third, only the first script associated with the name `trig1` is called.

For more detailed examples, see the *System Administrator's Guide*.

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 typemap

Synopsis

Modify the file name-to-type mapping table.

Syntax

```
p4 [g-opts] typemap
p4 [g-opts] typemap -i
p4 [g-opts] typemap -o
```

Description

The `p4 typemap` command allows Perforce administrators to set up a table linking Perforce 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 the Perforce client.

By default, Perforce automatically determines if a file is of type `text` or `binary` based on an analysis of the first 8192 bytes of a file. If the high bit is clear in each of the first 8192 bytes, Perforce 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` flag, it's easy to overlook this, particularly in cases where files' types were usually (but not always) detected correctly. The most common examples of this are associated with PDF files (which sometimes begin with over 8192 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.

Users can override any file type mapping defined in the typemap table by explicitly specifying the file type on the Perforce command line.

Form Fields

The `p4 typemap` form contains a single `TypeMap:` field, consisting of pairs of values linking file types to file patterns specified in depot syntax:

Column	Description
filetype	Any valid Perforce file type. For a list of valid file types, see the *File Types* section.
pattern	A file pattern in depot syntax. When a user adds a file matching this pattern, its default filetype will be the file type specified in the table.

Options

`-i`	Reads the typemap table from standard input without invoking the user's editor.
`-o`	Writes the typemap table to standard output without invoking the user's editor.
g-opts	See the *Global Options* section.

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` flag

- 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 the *File Types* section.

- If you use the `-t` flag 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 server 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 Perforce 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 the *File Types* section.

By default, Perforce 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/...
```

Related Commands

To add a new file with a specific type, overriding the typemap table	`p4 add -t` *`type file`*
To change the filetype of an opened file, overriding any settings in the typemap table	`p4 reopen -t` *`type file`*

p4 unlock

Synopsis

Release the lock on a file.

Syntax

```
p4 [g-opts] unlock [-c changelist#] [-f] file...
```

Description

The `p4 unlock` command releases locks created by `p4 lock`.

If the file is open in a pending changelist other than `default`, then you must use the `-c` flag to specify the pending changelist. If no changelist is specified, `p4 unlock` unlocks files in the default changelist.

Administrators can use the `-f` option to forcibly unlock a file opened by another user.

If no file name is given, all files in the designated changelist are unlocked.

Options

`-c changelist#`	Unlock files in pending changelist `changelist#`
`-f`	Superuser force flag; allows unlocking of files opened by other users.
`g-opts`	See the *Global Options* section.

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 unshelve

Synopsis

Restore shelved files from a pending change into a workspace

Syntax

```
p4 unshelve -s changelist [-f] [-n] [-c changelist] [file ...]
```

Description

The `p4 unshelve` command retrieves files that are shelved in a pending changelist into a pending changelist on the invoking user's workspace. Access to shelved files from a pending changelist is controlled by the user's permissions on the files.

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.)

Options

`-s` *`changelist`*	Specify the pending changelist number that contains the originally-shelved files.
`-c changelist`	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.
`g-opts`	See the *Global Options* section.

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 user

Synopsis

Create or edit Perforce user specifications and preferences.

Syntax

```
p4 [g-opts] user [-f] [username]
p4 [g-opts] user -d [-f] username
p4 [g-opts] user -o [username]
p4 [g-opts] user -i [-f]
```

Description

By default, any system user becomes a valid Perforce user the first time he uses any Perforce command that can update the depot or its metadata. Perforce automatically creates a user spec with default settings for the invoking user. Use the `p4 user` command to edit these settings or to create new user records. (By default, new users are created automatically. After installing Perforce, a Perforce superuser can control this behavior with the `p4 configure` command.)

When called without a *`username`*, `p4 user` edits 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 or registry variable.

Perforce superusers can create new users or edit existing users' specifications with the `-f` (force) flag: `p4 user -f username`.

The user who gives a Perforce 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 Perforce superuser, and uses the syntax `p4 user -f username`, user *`username`* is edited.
- If the `-u username` flag is used on the command line (for instance, `p4 -u joe submit`), the command runs as that user (a password may be required);
- If the above hasn't been done, but the file pointed to by the `P4CONFIG` environment or registry variable contains a setting for `P4USER`, then the command runs as that user.
- If neither of the above has been done, but the `P4USER` environment or registry variable has been set, then 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.

Form Fields

Field Name	Type	Description
`User:`	Read-only	The Perforce username under which `p4 user` was invoked. By default, this is the user's system username.
`Type:`	Writable	Type of user: `standard` or `service`.
`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 Perforce command.
`FullName:`	Writable	The user's full name.
`JobView:`	Writable	A description of the jobs to appear automatically on all new changelists (described in the *Usage Notes* below).
`Password:`	Writable	The user's password (described in the *Usage Notes* below).
`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 the *Usage Notes* below).

Options

`-d` *`username`*	Deletes the specified user. Only user *`username`*, or the Perforce superuser, can run this command.
`-f`	Superuser force flag; allows the superuser to modify or delete the specified user, or to change the last modified date.
`-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 the *Global Options* section.

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` flag can be used by non-superusers only to delete the user specification that invoked the `p4 user` command. Perforce superusers can delete any Perforce 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 Perforce user can impersonate another by setting `P4USER` or by using the *globally available* `-u` flag. 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, see the *System Administrator's Guide*.

  If you edit a password in the `p4 user` form, do not use the comment character `#` within the password; Perforce interprets everything following that character on the same line as a comment, and does not store it as part of the password.
- 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 Perforce 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
  ```

  then the change review daemon sends `joe` email whenever any `README` file has been submitted, and whenever any file under `//depot/main` has been submitted.
- There is a special setting for job review when used with the Perforce 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.

Examples

`p4 user joe`	View the user specification of Perforce user `joe`.
`p4 user`	Edit the user specification for the current Perforce user.
`p4 user -d sammy`	Delete the user specification for the Perforce user `sammy`.
`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 Perforce user named `joe2` if the caller is a Perforce superuser, and `joe2` doesn't already exist as a Perforce user. If user `joe2` already exists, allow a Perforce superuser to modify the user's settings.

Related Commands

To view a list of all Perforce 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

Synopsis

Print a list of all known users of the current server.

Syntax

```
p4 [g-opts] users [-l] [-a] [-m max] [ user... ]
```

Description

`p4 users` displays a list of all the users known to the current Perforce server. For each user, the information displayed includes their Perforce user name, their email address, their real name, and the date and time the user last accessed the server.

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

`-m max`	List only the first *`max`* users.
`-l`	Login information: includes time of last password change and login ticket expiry, if applicable. You must be a Perforce superuser to use this option.
`-a`	Include service users in list.
`g-opts`	See the *Global Options* section.

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 add or edit information about a particular user	`p4 user`
To edit information about the current client workspace	`p4 client`

p4 verify

Synopsis

Verify that the server archives are intact.

Syntax

```
p4 [g-opts] verify [ -m maxRevs ] [-q ] [ -u | -v | -z ] file[revRange]...
```

Description

`p4 verify` reports the revision specific information and an MD5 digest (fingerprint) of the revision's contents.

If invoked without arguments, `p4 verify` computes and displays the MD5 digest of each revision. If a revision is missing from the archive and therefore can't be reproduced, the revision's output line ends with `MISSING!` If the digests differ, the output line for the corrupt file ends with `BAD!`

Options

`-q`	Run quietly; verify the integrity of files for which MD5 digests have previously been generated, and only display output if there are errors.
`-u`	Store the filesize and MD5 digest of each file in the Perforce database if and only if no filesize and/or digest has been previously stored. Subsequent uses of `p4 verify` will compare the computed version against this stored version.
`-v`	Store the MD5 digest of each file in the Perforce database, even if there's already a digest stored for that file, overwriting the existing digest. (The `-v` flag is used only to update the saved digests of archive files which have been deliberately altered outside of Perforce control by a Perforce system administrator.)
`-z`	Skip revisions that have already been computed in the current pass; this option speeds verifications in the cases of revisions which exist via lazy copies.
`-m maxRevs`	Limit `p4 verify` to *`maxRevs`* revisions.
`g-opts`	See the *Global Options* section.

Usage Notes

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

- If `p4 verify` returns errors, contact Perforce technical support.
- It is good administrative practice to regularly verify the integrity of your depot files with `p4 verify -q //...`

  For details, see the *Perforce System Administrator's Guide*.
- As of Release 2005.1, Perforce Servers track file length metadata on a per-revision basis. Newly submitted files have file length metadata added to the database automatically. (You must still run `p4 verify -u` at least once following an upgrade to 2005.1, in order to update file length metadata for any pre-2005.1 files for which file lengths were not stored.)

  Administrators of very large sites (such as those with tens of millions of revisions) may encounter memory constraints immediately following an upgrade to 2005.1 if they attempt to update file length metadata for the entire repository at once. If this is the case, use the `-m` *`maxRevs`* flag to limit the number of revisions updated per command; `p4 verify -u -m 1000000 //...` limits file length metadata recomputation to a million files at a time, enabling an administrator to divide file length metadata recomputation over several calls to `p4 verify`.

p4 where

Synopsis

Show where a particular file is located, as determined by the client view.

Syntax

```
p4 [g-opts] where [file...]
```

Description

`p4 where` uses the client view and client 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 the *Global Options* section.

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 client 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 `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

Synopsis

Create or edit a client workspace specification and its view.

Syntax

```
p4 [g-opts] workspace [-f -t template] [workspacename]
p4 [g-opts] workspace -o [-t template] [workspacename]
p4 [g-opts] workspace -d [-f] workspacename
p4 [g-opts] workspace -i [-f]
```

Description

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

p4 workspaces

Synopsis

List all client workspaces currently known to the system.

Syntax

`p4 [g-opts] workspaces [ -u user ] [ -e namefilter -m max ]`

Description

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

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: `ksh`, `sh`, `bash`	`P4CLIENT=value ; export P4CLIENT`
UNIX: `csh`	`setenv P4CLIENT value`
VMS	`def/j P4CLIENT "value"`
Mac OS X (`bash`)	`P4CLIENT=value ; export P4CLIENT`
Windows	`p4 set P4CLIENT=value` Windows administrators running Perforce as a service can set variables for use by a specific service with `p4 set -S svcname var=value`, or set variables for all users on the local machine with `p4 set -s var=value`. (See the `p4 set` command for more details on setting Perforce's registry variables in Windows).

Perforce's environment variables can be grouped into the following four categories:

- *Crucial*: The variable must almost always be set on the client; default values are rarely sufficient. Understanding these variables is crucial for users and administrators alike.
- *Useful*: Setting this variable can provide additional functionality to the user, but is not required for most Perforce operations.
- *Esoteric*: The default value is normally sufficient; it rarely needs to be changed.
- *Server*: The variable is set by the Perforce system administrator on the machine running the Perforce server. Some of these variables are used by Perforce clients as well; in these cases, the variable is listed twice.

Crucial Variables	Useful Variables	Esoteric Variables	Server Variables
P4CLIENT	P4CONFIG	P4PAGER	P4AUDIT
P4PORT	P4DIFF	PWD	P4JOURNAL
P4PASSWD	P4EDITOR	TMP, TEMP	P4LOG
P4USER	P4MERGE	P4TICKETS	P4PORT
	P4CHARSET	P4LANGUAGE	P4ROOT
		P4COMMANDCHARSET	P4DEBUG
		P4DIFFUNICODE	P4NAME
		P4MERGEUNICODE	P4DESCRIPTION
		P4CLIENTPATH	P4ZEROCONF

P4AUDIT

Description

Location of the server 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, the server adds a line to the audit log file every time file content is transferred from the server to the client. On an active server, the audit log file will grow very quickly.

Lines in the audit log appear in the form:

```
date time user@client clientIP command file#rev
```

For example:

```
2006/05/09 09:52:45 karl@nail 192.168.0.12 diff //depot/src/x.c#1
2006/05/09 09:54:13 jim@stone 127.0.0.1 sync //depot/inc/file.h#1
```

If a command is run on the machine that runs the Perforce Server, the *`clientIP`* is shown as `127.0.0.1`.

For commands which arrive through a Perforce Proxy, the IP address is reported in the form *`proxyIP`*`/`*`clientIP`*, and the command is reported as *`command`*`-proxy`.

For more information, see the *System Administrator's Guide*.

P4AUTH

Description

A hostname and port number of an optional Perforce central authorization server (that is, a Perforce server for which this Perforce server derives its protections table).

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

Perforce server examples
`perforce.example.com:1818`
`192.168.0.123:1818`

Notes

The format of `P4AUTH` on is *`host`*`:`*`port`*, or *`port`* by itself if both the Perforce server and the central authorization server are running on the same host.

Port numbers must be in the range `1024` through `32767`.

For more about managing multiserver environments, see the *System Administrator's Guide*.

P4BROKEROPTIONS

Description

Set Perforce 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 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 flags for which there are no corresponding environment variables, or when you are doing so within the context of a Windows service.

For more information on the Perforce Broker, see the *Perforce System Administrator's Guide*.

P4CHARSET

Description

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 charset cmd`	Yes

Value if not Explicitly Set

Operating System	Value
All	None. If the Perforce server is operating in unicode mode and `P4CHARSET` is unset, Perforce client programs return an error message.

Notes

`P4CHARSET` only affects files of type `unicode` and `utf16`; non-unicode files are never translated.

For servers operating in the default (non-Unicode mode), `P4CHARSET` must be left unset (or set to `none`) on client machines. If `P4CHARSET` is set, but the server is not operating in internationalized mode, the server returns the following error message:

```
Unicode clients require a unicode enabled server.
```

For servers operating in Unicode mode, `P4CHARSET` must be set on client machines. If `P4CHARSET` is unset, but the server is operating in Unicode mode, client programs 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*:

```
http://www.perforce.com/perforce/doc.current/user/i18nnotes.txt
```

For a complete list of valid `P4CHARSET` values, issue the command `p4 help charset`.

P4CHANGE

Description

A hostname and port number of an optional Perforce central changelist server (that is, a Perforce server for which this Perforce server derives the most recent unused changelist).

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

Perforce server examples
`perforce.example.com:1818`
`192.168.0.123:1818`

Notes

The format of `P4CHANGE` on is *`host`*`:`*`port`*, or *`port`* by itself if both the Perforce server and the central changelist server are running on the same host.

Port numbers must be in the range `1024` through `32767`.

For more about managing multiserver environments, see the *System Administrator's Guide*.

P4COMMANDCHARSET

Description

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` *`commandcharset cmd`*	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*:

```
http://www.perforce.com/perforce/doc.current/user/i18nnotes.txt
```

For a complete list of valid `P4COMMANDCHARSET` values, issue the command `p4 help charset`.

P4CLIENT

Description

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 clientname cmd`	Yes

Value if not Explicitly Set

Operating System	Value
Windows	Value of `COMPUTERNAME` environment variable
All others	Name of host machine

Examples

```
cinnamon
computer1
WORKSTATION
```

P4CLIENTPATH

Description

A list of directories to which Perforce client programs are permitted to write.

Any attempt by a Perforce server 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
```

P4CONFIG

Description

Contains a file name without a path. The file(s) it points to are used to store other Perforce environment or registry variables. The current working directory (returned by `PWD`) and its parents are searched for the file. If the file exists, then the variable settings within the file are used.

The variable settings in the file must sit alone on each line and be in the form *`variable=value`*.

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=ida:3548
```

Notes

`P4CONFIG` makes it trivial to switch Perforce 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 Perforce settings will change to the settings in the configuration files automatically as you move from directories in one workspace to another.

You can set the following variables from within the `P4CONFIG` file:

- `P4CHARSET`
- `P4CLIENT`
- `P4DIFF`
- `P4EDITOR`
- `P4HOST`
- `P4LANGUAGE`
- `P4MERGE`
- `P4PASSWD`
- `P4PORT`
- `P4TICKETS`
- `P4USER`

P4DEBUG

Description

Set Perforce server or proxy trace flags.

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 not set, this variable is not used.

Examples

```
server=1
server=2
server=3
```

Notes

In most cases, the Perforce server trace flags are useful only to administrators working with Perforce Technical Support to diagnose or investigate a problem.

The preferred way to set trace flags for the Perforce server (or proxy) is to set them on the `p4d` (or `p4p`) command line. For technical reasons, this does not work for sites running Perforce servers or proxies as services under Windows. Administrators at such sites can use `p4 set` to set the trace flags within `P4DEBUG`, allowing the service to run with the flags enabled.

Some server debug levels require specific server release levels.

Setting server debug levels on a Perforce server (`p4d`) has no effect on the debug level of a Perforce Proxy (`p4p`) process, and vice versa.

For further information, see the *Perforce System Administrator's Guide*.

P4DESCRIPTION

Description

In a Zeroconf environment, a comment describing a Perforce server.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4d -0 -Id` *`description`*	N/A

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
Art assets are stored on this Perforce Server
```

Notes

P4DIFF

Description

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, Perforce's internal diff routine is used.

Examples

```
diff
diff -b
windiff.exe
```

Notes

The value of `P4DIFF` can contain flags to the called program, for example, `diff -u`.

The commands `p4 describe`, `p4 diff2`, and `p4 submit` all use a diff program built into the Perforce server program `p4d`. This cannot be changed.

P4DIFFUNICODE

Description

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 `P4DIFF` if the file being diffed is of type `unicode` or `utf16`, and the character set is passed as the first argument to the command. For details, see the *Release Notes*:

```
http://www.perforce.com/perforce/doc.current/user/relnotes.txt
```

P4EDITOR

Description

The editor invoked by those Perforce 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
UNIX and OS X	If `EDITOR` is set to any value, then the value of `EDITOR`; otherwise, `vi`.
Windows	If `SHELL` is set to any value, then `vi`; otherwise, `notepad`
VMS	If `POSIX$SHELL` is set, then `vi`; otherwise, `edit`.
Macintosh	If `EDITOR_SIGNATURE` is set, then the program with that four-character creator; otherwise, `SimpleText`.

Examples

```
/usr/bin/vi
emacs
SimpleText
```

Notes

The regular Perforce 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`.

P4HOST

Description

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

Perforce 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 server 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.

P4JOURNAL

Description

A file that holds the Perforce 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 Perforce server root.

Setting `P4JOURNAL` to `off` will disable journaling. This is not recommended.

For further information, see the *Perforce System Administrator's Guide*.

P4LANGUAGE

Description

This environment variable is reserved for system integrators.

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

Description

Name and path of the file to which Perforce 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 should be specified relative to the Perforce server root.

For further information, see the *Perforce System Administrator's Guide*.

P4MERGE

Description

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 (or registry variable on Windows, as set by `p4 set`) 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 Perforce. 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

Description

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. For details, see the *Release Notes*:

```
http://www.perforce.com/perforce/doc.current/user/relnotes.txt
```

P4NAME

Description

A unique identifiable name for a Perforce server.

`P4NAME` is used in both Zeroconf environments and by `p4 configure` 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 -0 -In` *`name`*	N/A

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
Artists' Store
masterserver
failoverserver
buildserver
```

Notes

P4PAGER

Description

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

Description

Supplies the current Perforce user's password for any Perforce client 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

Perforce 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 sends your password to the Perforce server. 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 Perforce 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

Description

For the Perforce 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 Perforce 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 Perforce Proxy (`p4p`).

Only the account running `p4p` needs to have read/write permissions in this directory.

For more information on setting up a Perforce Proxy, see the *Perforce System Administrator's Guide*.

P4PFSIZE

Description

For the Perforce 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

For more information on setting up a Perforce Proxy, see the *Perforce System Administrator's Guide*.

P4POPTIONS

Description

Set Perforce 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 `"Perforce Proxy"` service, the Proxy will listen to port 1999 and communicate with the Perforce Server at `mainserver:1666`.

Most installations do not need to use `P4POPTIONS`, because there are already environment variables associated with most `p4p` flags; in the example shown above, you can use `P4PORT` and `P4TARGET`. Use `P4POPTIONS` when you need to call `p4p` with flags for which there are no corresponding environment variables, and when you are doing so within the context of a Windows service.

For more information on setting up a Perforce Proxy, see the *Perforce System Administrator's Guide*.

P4PORT

Description

For the Perforce server, and Perforce Proxy, the port number on which it listens.

For Perforce clients, the host and port number of the Perforce server or proxy with which to communicate.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	`p4 -p` *`host:port cmd`*	Yes

Value if not Explicitly Set

Program	Value
Perforce server	`1666`
Perforce proxy	`1666`
Perforce client	`perforce:1666`

Examples

Perforce client examples	Perforce server examples
`1818`	`1818`
`squid:1234`	`1234`
`example.com:1234`	`1234`
`192.168.0.123:1818`	`1818`

Notes

The format of `P4PORT` on the Perforce client is *`host:port`*, or *`port`* by itself if both the Perforce client and server 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 Perforce server ignores requests from any IP addresses other than the one specified in `P4PORT`.

To use the default value `perforce` with a Perforce server, define `perforce` as an alias to the host running the server in `/etc/hosts` on UNIX, or in `%SystemRoot%\system32\drivers\etc\hosts` on Windows, or use DNS.

If your network environment and Perforce Server have been configured to support Zeroconf services, you can set `P4PORT` to the value of the service name.

P4ROOT

Description

Directory in which the Perforce server stores its files and subdirectories.

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 Perforce 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 Perforce server (`p4d`).

Only the account running `p4d` needs to have read/write permissions in this directory.

For more information on setting up a Perforce server, see the *Perforce System Administrator's Guide*.

P4TARGET

Description

For the Perforce Proxy and replica servers, the name and port number of the target Perforce server (that is, the Perforce server for which P4P acts as a proxy, or for the master server from which the replica server retrieves its copy of the master's metadata, versioned files, or both).

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
Perforce Proxy	`perforce:1666`
Replicated environments	None

Examples

Perforce server examples
`1818`
`master:11111`
`perforce.example.com:1234`
`192.168.0.123:1818`

Notes

The format of `P4TARGET` on is *`host:port`*, or *`port`* by itself if both the Perforce server and the proxy (or replica) are running on the same host.

Port numbers must be in the range `1024` through `32767`.

For more about the Perforce Proxy or replicated environments, see the *System Administrator's Guide*.

P4TICKETS

Description

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	No	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.

P4USER

Description

Current Perforce 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 Perforce username is the same as the OS username.

If a particular Perforce user does not have a password set, then any other Perforce user can impersonate this user by using the `-u` flag with their Perforce client commands. To prevent this, users should set their password with the `p4 user` or `p4 passwd` command.

If a user has set their Perforce password, you can still run commands as that user (if you know the password) with `p4 -u` *`username`* `-P` *`password command`*.

Perforce superusers can impersonate users without knowing their passwords. For more information, see the *Perforce System Administrator's Guide*.

P4ZEROCONF

Description

If set, `p4d` attempts to advertise itself as a zeroconf service upon startup.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4d -0`	N/A

Value if not Explicitly Set

Operating System	Value
All	None

PWD

Description

The directory used to resolve relative filename arguments to Perforce client 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 isn't 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 Perforce 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 unsettings you've placed in your startup files; the confusion only occurs when variables are exported to subshells.)

TMP, TEMP

Description

The directory to which Perforce clients and servers 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	On Perforce clients: the current working directory. On Perforce servers: `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 will be written in the directories described in the table above.

Additional Information

This section describes features of Perforce that you'll use with multiple commands. We've included information on the following topics:

- *Flags* that can be used with any Perforce command,
- How to use Perforce *file specifications* in depot syntax, client syntax, and local syntax,
- Perforce *file types*, and
- How to create and use *views* to describe client workspaces, branches, and labels.

For an in-depth treatment of these and other topics from a conceptual level, please see the *Introduction to Perforce*, which is available at our web site: `http://www.perforce.com`.

Global Options

Synopsis

Global options for Perforce commands; these options can be supplied on the command line before any Perforce command.

Syntax

```
p4 [-cclient -ddir -Hhost -pport -Ppass -uuser -xfile -Ccharset -Qcharset
-Llanguage] [-G] [-s] [-z tag] cmd [args ...]
p4 -V
p4 -h
```

Options

`-c client`	Overrides any `P4CLIENT` setting with the specified client name.
`-d dir`	Overrides any `PWD` setting (i.e. current working directory) and replaces it with the specified directory.
`-G`	Causes all output (and batch input for form commands with `-i`) to be formatted as marshalled Python dictionary objects. This is most often used when scripting.
`-H host`	Overrides any `P4HOST` setting and replaces it with the specified hostname.
`-p port`	Overrides any `P4PORT` setting with the specified port number.
`-P pass`	Overrides any `P4PASSWD` setting with the specified password.
`-s`	Prepends a descriptive field (for example, `text:`, `info:`, `error:`, `exit:`) to each line of output produced by a Perforce command. This is most often used when scripting.
`-u user`	Overrides any `P4USER`, `USER`, or `USERNAME` setting with the specified user name.
`-x file`	Instructs Perforce to read arguments, one per line, from the specified file. If file is a single hyphen (`-`), then standard input is read.
`-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`.

`-V`	Displays the version of the `p4` client program 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 Perforce command. Options specified after the Perforce 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; a user adding several files at once could create a text file with the names of these files and invoke `p4 -x textfile add` to add them all at once.

  The `-x` option can be extremely powerful - as powerful as whatever generates its input. For example, a UNIX developer wishing to edit any file referring to an included `file.h` file, for instance, could `grep -l file.h *.c | cut -f1 -d: | p4 -x - edit`.

  In this example, the `grep` command lists occurrences of `file.h` in the `*.c` files, the `-l` option tells `grep` to list each file only once, and the `cut` command splits off the filename from `grep`'s output before passing it to the `p4 -x` 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 will find the `-G` option extremely useful for scripting. For instance, to get a dictionary of all 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 may not be intuitively obvious what keys are used by the client program. 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--%d--' % num
        dict =  marshal.load(sys.stdin, 'rb')
        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, it is necessary to call `marshal.load()` to read the data in binary ("`rb`") mode.

- Some uses of the global options are absurd.

  For example, `p4 -c` *`anotherclient`* `help` provides exactly the same output as `p4 help`.

Examples

`p4 -p` *`new_server`*`:1234 sync`	Performs a sync using server *`new_server`* and port `1234`, regardless of the settings of the `P4PORT` environment variable or registry setting.
`p4 -c` *`new_client`* `submit -c 100`	The first `-c` is the global option to specify the client 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 then be easily detected by examining the command's output for lines beginning with "`error:`"

File Specifications

Synopsis

Any file can be specified within any Perforce command in client syntax, depot syntax, or local syntax. Client workspace names and depot names share the same namespace; there is no way for the Perforce server to confuse a client name with 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.)

Perforce 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 Perforce syntax.

Perforce 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 is provided below

Syntax	Example
Local syntax	`/staff/user/usercws/file.c`
Depot syntax	`//depot/source/module/file.c`
Client syntax	`//usercws/file.c`

Wildcards

The Perforce system allows the use of three 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.

For example:

Expression	Matches
`J*`	Files in the current directory starting with `J`
`*/help`	All files called `help` in current subdirectories
`./...`	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/...`	All files under `/usr/bruno`
`//bruno_ws/...`	All files in the workspace or depot that is named `bruno_ws`
`//depot/...`	All files in the depot
`//...`	All files in all depots

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 sync` *`file`*`#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``@`*`labelname`*	Label name: The revision of *`file`* in the label *`labelname`*.

Modifier	Meaning
file@clientname	Client name: The revision of *file* last taken into client workspace *clientname*.
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/dd hh: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 (i.e., 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.

Some Perforce 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-Perforce) 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 Perforce commands can use revision ranges to modify file arguments. Revision ranges are two separate revision specifications, separated by a comma. For example, `p4 changes file#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.

  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 changes file#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

To support internationalization, Perforce permits the use of printable non-ASCII characters in filenames, label names, client workspace names, and other identifiers.

The pathname component separator (`/`) is not permitted in filenames, depot names, or client workspace names, but can appear in label names, job names, or user names. The recursive subdirectory wildcard (`...`) is not permitted in file names, label names, or other identifiers.

Character	Reason
`...`	Perforce wildcard: matches anything, works at the current directory level and includes files in all directory levels below the current level.
`/`	Perforce separator for pathname components.

To refer to files containing the Perforce 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 Perforce 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

Synopsis

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 which 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 always 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 need be provided - 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 within mappings are provided in the usual Perforce 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. For instance, in the view defined by the following two mappings:

```
//depot/... //cws/...
//depot/dir1/... //cws/dir2/...
```

the entire depot is mapped to the client workspace, but the file `//depot/dir1/file.c` is mapped to `//cws/dir2/file.c`. If the order of the lines in the view is reversed, however:

```
//depot/dir1/... //cws/dir2/...
//depot/... //cws/...
```

then the file `//depot/dir1/file.c` is mapped to `//cws/dir1/file.c`, as the first mapping (mapping the file into `//cws/dir2`) is overridden by the second mapping

(which maps the entire depot onto the client workspace). A later mapping in a view always overrides an earlier mapping.

Spaces in path and file names

If a path or file name in a workspace view, branch view, or label view contains spaces, make sure to quote the path:

```
//depot/dir1/... "//cws/dir 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 Perforce wildcards), expand the characters to their ASCII equivalents as follows:

Character	ASCII expansion
`@`	`%40`
`#`	`%23`
`*`	`%2A`
`%`	`%25`

Client Views

Client views are used to map files in the depot to files in client workspaces, and vice versa. A client workspace is an area in which users perform their work; files are synced 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 mapping is reversed and the files are copied from the client workspace back 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/... //cws/...`
Full client workspace mapped to part of depot	`//depot/dir1/... //cws/...`
Some files in the depot are mapped to a different part of the client workspace	`//depot/... //cws/...` `//depot/rel1/... //cws/release1/...`

Client View	Sample Mapping
Some files in the depot are excluded from the client workspace	`//depot/dir1/... //cws/...` `-//depot/dir1/exclude/... //cws/dir1/exclude/...`
Files in the client workspace are mapped to different names than their depot names.	`//depot/dir1/old.* //cws/renamed/new.*`
Portions of filenames in the depot are rearranged in the client workspace	`//depot/dir1/%%1.%%2 //cws/dir1/%%2.%%1`
The files do not map the same way in each direction. The second line takes precedence, and the first line is ignored.	`//depot/dir1/... //cws/build/...` `//depot/dir2/... //cws/build/...`
An overlay mapping is used to map files from more than one depot directory into the same place in the workspace.	`//depot/dir1/... //cws/build/...` `+//depot/dir2/... //cws/build/...`

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.

Branch Views

Branching of the source tree allows multiple sets of files to evolve along different paths. The creation of a branch view allows Perforce 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 `p4 branch` *`newbranch`*. This will bring up a screen (similar to the one associated with `p4 client`) and allow 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 workspace view intending to use those files. You can do this by adding the newly-mapped branch of the depot to your current client workspace view and performing a `p4 sync` 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 doesn't 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

Synopsis

Perforce supports six base file types:

- `text` files,
- compressed `binary` files,
- native `apple` files on the Macintosh,
- Mac `resource` forks,
- symbolic links (`symlinks`), and
- `unicode` and `utf16` files.

File type modifiers are then applied to the base types allowing for support of RCS keyword expansion, file compression on the server, and more.

When adding files, Perforce first 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. If a match is *not* found, Perforce examines the first bytes of the file based on the `filesys.binaryscan` configurable (by default, 8192 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*.)

Perforce 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.

Perforce administrators can use the filesys.binaryscan and filetype.maxtextsize configurables (`p4 configure`) to change the default limits of 8192 bytes for text/binary detection, and the 10 MB RCS text file size limit respectively.

Base filetypes

The base Perforce file types are:

Keyword	Description	Comments	Server Storage
`text`	Text file	Treated as text on the client. Line-ending translations are performed automatically on Windows and Macintosh clients.	deltas in RCS format
`binary`	Non-text file	Accessed as binary files on the client. Stored compressed within the depot.	full file, compressed
`symlink`	Symbolic link	UNIX clients (and the BeOS client) access these as symbolic links. Non-UNIX clients treat them as (small) text files.	deltas in RCS format
`apple`	Multi-forked Macintosh file	AppleSingle storage of Mac data fork, resource fork, file type and file creator. For full details, please see the Mac client release notes.	full file, compressed, AppleSingle format.
`resource`	Macintosh resource fork	The only file type for Mac resource forks in Perforce 99.1 and before. Still supported, but the `apple` file type is preferred. For full details, please see the Mac client release notes.	full file, compressed

Keyword	Description	Comments	Server Storage
`unicode`	Unicode file	Perforce servers operating in unicode mode support the `unicode` file type. These files are translated into the local character set specified by `P4CHARSET`. Perforce servers not in unicode mode do not support the `unicode` file type. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format
`utf16`	Unicode file	If the Perforce server is operating in unicode mode, files are translated into the local character set as specified by `P4CHARSET`. If the Perforce server is operating in non-unicode mode, files are transferred as UTF-8, and translated to UTF-16 (with byte order mark, in the byte order appropriate for the client machine) in the client workspace. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format

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.
`+ko`	Old-style keyword expansion	Expands only the `$Id$` and `$Header$` keywords: This pair of modifiers exists primarily for backwards compatibility with versions of Perforce prior to 2000.1, and corresponds to the `+k` (`ktext`) modifier in earlier versions of Perforce.
`+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. Supported keywords are: • `$Id$` • `$Header$` • `$Date$` • `$DateTime$` • `$Change$` • `$File$` • `$Revision$` • `$Author$`
`+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`	Server stores the full compressed version of each file revision	Default server storage mechanism for `binary` files and newly-added `text` files larger than 10MB.

Modifier	Description	Comments
`+D`	Server stores deltas in RCS format	Default server storage mechanism for `text` files.
`+F`	Server stores full file per revision, uncompressed	Useful for large binaries, or for long ASCII files that aren't read by users as text, such as PostScript files.
`+S`	Only the head revision is stored on the server	Older revisions are purged from the depot upon submission of new revisions. Useful for executable or `.obj` files.
`+Sn`	Only the most recent *n* revisions are stored on the server, 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 `+Sn` file's *n* to a number less than its current value. Earlier revisions unaffected; see the Usage Notes for details.
`+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 server runs an `archive` trigger to access the file. See the *System Administrator's Guide* for details.

A file's type is normally preserved between revisions, but can be overridden or changed with the `-t` flag 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

The following table lists recommended Perforce file types and modifiers for common file extensions.

File Type	Perforce file type	Description
`.asp`	`text`	Active server page file
`.avi`	`binary+F`	Video for Windows file
`.bmp`	`binary`	Windows bitmap file
`.btr`	`binary`	Btrieve database file
`.cnf`	`text`	Conference link file
`.css`	`text`	Cascading style sheet file
`.doc`	`binary`	Microsoft Word document
`.dot`	`binary`	Microsoft Word template
`.exp`	`binary+w`	Export file (Microsoft Visual C++)
`.gif`	`binary+F`	GIF graphic file
`.gz`	`binary+F`	Gzip compressed file
`.htm`	`text`	HTML file
`.html`	`text`	HTML file
`.ico`	`binary`	Icon file
`.inc`	`text`	Active Server include file
`.ini`	`text+w`	Initial application settings file
`.jpg`	`binary`	JPEG graphic file
`.js`	`text`	JavaScript language source code file
`.lib`	`binary+w`	Library file (several programming languages)
`.log`	`text+w`	Log file
`.mpg`	`binary+F`	MPEG video file
`.pdf`	`binary`	Adobe PDF file
`.pdm`	`text+w`	Sybase Power Designer file
`.ppt`	`binary`	Microsoft Powerpoint file
`.xls`	`binary`	Microsoft Excel file

For more about mapping file names to Perforce filetypes, see the `p4 typemap` command.

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 $`
`$Header$`	Synonymous with `$Id$`	`$Header: //depot/path/file.txt#3 $`
`$Date$`	Date of last submission in format *`YYYY/MM/DD`*	`$Date: 2000/08/18 $`
`$DateTime$`	Date and time of last submission in format *`YYYY/MM/DD hh:mm:ss`* Date and time are as of the local time on the Perforce server at time of submission.	`$DateTime: 2000/08/18 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; 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 `+Sn` (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 `+Sn` modifier was used will remain unaffected. (Syncing to a non-head revision submitted *after* the `+Sn` modifier was used will delete the file from your workspace. Such revisions are displayed as `purge` operations in the output of `p4 filelog`.)
- 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 server at time of submission, and *not* the time on the client 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.

  The `+m` modifier on a file allows this to happen; if set, Perforce will ignore the `modtime` ("file's timestamp at time of submission") or `nomodtime` ("date and time on the client at time of sync") option setting of the client workspace when syncing the file, and always restore the file's original timestamp at the time of submit.

- Versions of Perforce prior to 99.1 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`
`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
`resource`	Macintosh resource fork	`resource`	none
`uresource`	Uncompressed Macintosh resource fork	`resource`	`+F`
`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`

Index

T