Perforce 2004.2 Command Reference

September 2004

Table of Contents

About This Manual

Synopsis

This is the *Perforce 2004.2 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`
Files	`p4 add`, `p4 delete`, `p4 diff`, `p4 diff2`, `p4 dirs`, `p4 edit`, `p4 files`, `p4 fstat`, `p4 lock`, `p4 print`, `p4 rename`, `p4 revert`, `p4 unlock`
Changelists	`p4 change`, `p4 changes`, `p4 describe`, `p4 filelog`, `p4 opened`, `p4 reopen`, `p4 review`, `p4 submit`
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 counter`, `p4 counters`, `p4 depot`, `p4 depots`, `p4 logger`, `p4 monitor`, `p4 obliterate`, `p4 reviews`, `p4 triggers`, `p4 typemap`, `p4 verify`
Security	`p4 group`, `p4 groups`, `p4 login`, `p4 logout`, `p4 passwd`, `p4 protect`, `p4 tickets`, `p4 user`, `p4 users`
Environment	`p4 set`, *Environment and Registry Variables*, `P4CHARSET`, `P4CLIENT`, `P4CONFIG`, `P4DEBUG`, `P4DIFF`, `P4EDITOR`, `P4HOST`, `P4JOURNAL`, `P4LANGUAGE`, `P4LOG`, `P4MERGE`, `P4PAGER`, `P4PASSWD`, `P4PORT`, `P4ROOT`, `P4USER`, `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 *Perforce 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] [-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 linked to the default changelist. Use `-c` to specify a different changelist.

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 1024 bytes of the file to determine whether it is `text` or `binary`, and the files are stored in the depot accordingly. Text file revisions are stored in reverse delta format; binary file revisions are stored as full files.

To explicitly specify a file type, overriding both the typemap table and Perforce's default file type detection mechanism, use the `-t` *`filetype`* flag.

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

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.
`-t` *`filetype`*	Adds the file as the specified *`filetype`*. Please see the *File Types* chapter for a list of Perforce file types.
`-f`	Use the `-f` flag to force inclusion of wildcards in filenames. See the *File Specifications* chapter for details.
`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%2540.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 218.

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 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
```

Description

The `p4 admin` command allows Perforce superusers to perform administrative tasks whether they are on the host running the Perforce server or not.

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`* respectively, where *`n`* is a sequence number. If no prefix is specified, the default filenames `checkpoint.`*`n`* and `journal.`*`n`* are used. Use the `-z` option to save the checkpoint and journal files in compressed form.

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.

Options

`-z`	For `p4 admin checkpoint`, save the checkpoint and saved journal file in compressed (gzip) format, appending the `.gz` suffix to the 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
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.
- 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*, 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*, where *n* is a sequence number.

p4 annotate

Synopsis

Print file lines along with their revisions.

Syntax

`p4 [g-opts] annotate [ -a -c -q ] 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.
`-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.

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 branch

Synopsis

Create or edit a branch specification 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 specification, integrate files by typing `p4 integrate -b` *`branchname`*; the branch specification 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 specification. 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 specification was last accessed.
`Update:`	read-only	The date the branch specification was last changed.

Field Name	Type	Description
`Options:`	mandatory	Either `unlocked` (the default) or `locked`. If `locked`, only the `Owner:` can modify the branch spec, and the spec 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 specification. 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 specification (the `Update:` field is writable when using the `-f` flag).
`-i`	Read the branch specification from standard input without invoking an editor.
`-o`	Write the branch specification 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 may contain multiple mappings. See the *Views* chapter for more information on specifying views.

- 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 specifications	`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 specifications.

Syntax

```
p4 [g-opts] branches
```

Description

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

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 or edit a branch specification	`p4 branch`

p4 change

Synopsis

Create or edit a changelist specification.

Syntax

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

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.

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.

You can use `p4 change` *`changelist#`* can be used to edit the description of a pending changelist, and to view the fields of a submitted 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.

Form Fields

Field Name	Type	Description
`Change:`	Read-only	Contains the change number if editing an existing changelist, or new if creating a new changelist.
`Client:`	Read-only	Name of current client workspace.
`User:`	Read-only	Name of current Perforce user.

Field Name	Type	Description
`Status:`	Read-only value	`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, and cannot be changed after submission, except by the Perforce superuser.
`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 may be deleted from or added to this list.
`Files:`	List	The list of files being submitted in this changelist. Files may 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, but the superuser can delete changelists under other circumstances with the addition of the `-f` flag.
`-f`	Force flag. Allows the description of a submitted changelist to be edited. Editing a submitted changelist requires `admin` or `super` access.
`-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`. This option works in conjunction with the `-s` option to `p4 fix`, and is intended for use by Perforce Defect Tracking Integration (P4DTI).
`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 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 changes

Synopsis

List submitted and pending changelists.

Syntax

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

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`. The description is limited to the first 31 characters unless you provide the `-l` (long) flag.

If you provide file patterns as arguments, the changelists listed are those that affect files matching the patterns. Only `submitted` changelists are reported in this instance; `pending` changelists (by definition) have not yet affected any files in the depot. (If you try to view `pending` changelists while specifying a file pattern, you will get an error message.)

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` 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.

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`	Provide long output that includes the full descriptions of each changelist.
`-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`

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`.

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.
`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. Can also be used by Perforce superusers to delete or modify clients that they don't own.
`-d [-f]` *`clientname`*	Delete the specified client workspace, if the client is owned by the invoking user or it is unlocked. (The `-f` flag allows Perforce superusers 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.	`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	`noclobber`
`[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 the client specification (To make a `locked` client specification truly effective, you should also set a the client's owner's password with `p4 passwd`.) If `locked`, only the owner is able to use, edit, or delete the client spec. Perforce administrators can override the lock by using the `-f` (force) flag with `p4 client`.	`unlocked`

Option	Choice	Default
`[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.
`[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 client 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 line endings: `LF` only.
`mac`	Macintosh-style: `CR` only.
`win`	Windows-style: `CR`, `LF`.
`share`	Shared mode: Line endings are `LF` with any `CR/LF` pairs translated to `LF`-only style before storage or syncing with the depot. When you sync your client workspace, line endings will be `LF`. If you edit the file on a Windows machine, and your editor inserts `CR`s before each `LF`, the extra `CR`s will not appear in the archive file. The most common use of the `share` option is for users of Windows workstations who have UNIX home directories mounted as network drives; if they sync files from UNIX, but edit the files on the Windows machine, the `share` option eliminates any problems caused by Windows-based editors' insertion of extra carriage return characters at line endings.

- 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
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 uppercase 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`

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, with a root of `/usr/src`. The description of the workspace entered in the `p4 client` form is `Joe’s client`.

This command takes no arguments other than the *Global Options*.

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 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 counter

Synopsis

Access, set, 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 | monitor [ 1 | 0 ] ]
```

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`*.

The Perforce server uses three counters in the course of its regular operations: `change`, `job`, and `journal`. Superusers may 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.

You can enable or disable server process monitoring by setting the `monitor` counter to 1 or 0. You must stop and restart the Perforce server for this change to take effect. Once process monitoring is enabled, you can use `p4 monitor` to observe processes spawned by the Perforce server.

To configure password strength requirements or to require the use of the ticket-based authentication mechanism, set the `security` counter. See the *System Administrator's Guide* for details.

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.
`-f monitor [ 1 | 0 ]`	Activate or deactivate server process monitoring. See `p4 monitor` for details.

`-f security [ 0 | 1 | 2 | 3 ]`	Set the server security level.
	See the *System Administrator's Guide* for details.
`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
  ```
- Counters are represented internally as signed `int`s. (For most platforms, the largest value that can be stored in a counter is 2^{31} - 1, or 2147483647. A server running on a 64-bit platform can store counters up to 2^{63} - 1, or 9223372036854775807)

Related Commands

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 delete

Synopsis

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

Syntax

```
p4 [g-opts] delete [-c changelist#] 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.
`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`. However, it is possible to create multiple depots on a single server with the `p4 depot` command. Although it is normally not necessary to create multiple depots, there are two situations where this might be desirable:

- You'd like to have separate depots for separate projects stored on the same server.
- You want to access files on one Perforce server from another Perforce server.

In the former case, once a second depot has been created, it can be used exactly as the default depot `depot` is used. For example, to sync a file `README` in the `rel2` directory of the depot `new`, use `p4 sync //new/rel2/README`. It can also be used on the left-hand side of any client or branch view, exactly as the default depot `depot` is used.

In the latter case, referred to as the use of *remote depots*, the Perforce client's default Perforce server (i.e. the machine specified in `P4PORT`) allows the Perforce client to access a remote Perforce server, so the client doesn't need to know where the files are actually stored. Remote depots are restricted to read-only access. Thus, a Perforce client program can't `add`, `edit`, `delete`, or `integrate` files that reside in the depots on the other servers. For more information about remote depots, see the *Perforce User's Guide* and the *Perforce System Administrator's Guide*.

To create or edit a depot, use `p4 depot` *`depotname`* and edit the fields in the form.

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` or `remote`. Local depots are writable; remote depots are proxies for depots residing on other servers, and cannot be written to.
`Address:`	Writable	If the `Type:` is `local`, the address should be the word `subdir`. If the `Type:` is `remote`, the address should be the `P4PORT` address of the remote server.
`Map:`	Writable	If the `Type:` is `local`, the map should be the relative location of the depot subdirectory relative to the Perforce server's `P4ROOT`. It must contain the `...` wildcard; for example, a local depot `new` might have a `Map:` of `new/...` . If the `Type:` is `remote`, the map should be 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`	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 always accessed by a virtual user named `remote`, and by default, all files on any Perforce server may be accessed remotely. To limit or eliminate remote access to a particular server, use `p4 protect` to set permissions for user `remote` 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.

  The virtual user `remote` does not consume a Perforce license.
- 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`.
- Absolute paths in the `Map:` field on Windows must be specified with forward slashes (for instance, `d:/newdepot/`) in the depot form.

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`

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 a changelist and the changelist's files.

Syntax

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

Description

`p4 describe` displays the details of a changelist. The output includes the changelist number, the changelist's creator, the client workspace name, the date the changelist was created, and the changelist's description.

If the changelist has been `submitted`, the output also includes a list of affected files and the diffs of those files relative to the previous revision.

If the 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.)

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

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

Options

`-s`	Display a shortened output that excludes the files' diffs.
`-d`*`flag`*	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

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] [-sa | -sd | -se | -sr] [-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, 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)
`-sa`	Show only the names of opened files that are different from the revision in the depot, or are missing.
`-sd`	Show only the names of unopened files that are missing from the client workspace, but present in the depot.
`-se`	Show only the names of unopened files in the client workspace that are different than the revision in the depot.
`-sr`	Show only the names of opened files in the client workspace that are identical to the revision in the depot.

`-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] file1[rev] file2[rev]
p4 [g-opts] diff2 [-dflags -q -t] -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 specification 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`.
`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.
`-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 specification 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.

Examples

`p4 diff2 -ds file#1 file`	Compare the second 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 specification `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 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#] [-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.

If `p4 edit` is run on any files that are already opened for edit, these files are simply moved into the specified changelist, which must have a status of `pending`.

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.
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 -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 218.

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 filelog

Synopsis

Print detailed information about files' revisions.

Syntax

```
p4 [g-opts] filelog [-i] [-l] [-t] [-m maxrev] 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`, or `integrate`;

  If the action is `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.

Options

`-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 the full description of each revision.
`-t`	Display the time as well as the date.
`-m` *`maxrev`*	List only the first *`maxrev`* changes per file 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
Yes	No	`list`

- Since `p4 filelog`'s output can be quite large when called with highly non-restrictive file arguments (for example, `p4 filelog //depot/...` will print the revision history for every file in the depot), it 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 `+S`, or `tempobj`) 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] 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.
`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 `closed` immediately.

To change a job status to something other than `closed` when you submit a changelist, supply the `-s` option to `p4 fix`, `p4 submit`, or `p4 change`.

Because described work may be fixed over multiple changelists, one job may 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`. 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.
`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 as part of the Perforce Defect Tracking Integration (P4DTI). For more about P4DTI, see the P4DTI product information page at:

  ```
  http://www.perforce.com/perforce/products/p4dti.html
  ```

  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 `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 inprogress job002433`	Mark `job002433` as `inprogress`, 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] [-j jobname] [-c changelist#] [file[revRange]...]
```

Description

Once a job has been linked to a particular changelist with `p4 fix`, `p4 change`, or `p4 submit`, and once the changelist has been submitted, the job is said to have been *fixed* by the changelist. 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
```

You can limit the listed fixes by combining the following flags when calling `p4 fixes`:

- Use the `-c` flag to specify a particular *`changelist`*. Only the jobs fixed by that changelist are listed.
- Use the `-j` flag to specify a particular *`jobname`*. Only those changelists that fix that job are listed.
- Provide one or more file pattern arguments. Only those changelists that affected files that match the file patterns are listed. If a revision specifier or revision range is included, only changelists that affected files at the given revisions are listed.

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.
`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 changelists that included any files under `//depot/proj1`.
`p4 fixes -c 414`	Display all jobs fixed by 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 `ws` 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

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 [ -c | -e changelist# ] [ -Oflag -Rflag ] file[rev]...
```

Description

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

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`
`headAction`	action taken at head revision, if in depot	one of `add`, `edit`, `delete`, `branch`, or `integrate`
`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 modification time, if in depot. Time is measured in seconds since 00:00:00 UTC, January 1, 1970	`919283152` is a date in early 1999
`headType`	head revision type, if in depot	`text`, `binary`, `text+k`, etc. (see the chapter on *File Types*.)

Field Name	Description	Example/Notes
`haveRev`	revision last synced to workspace, if on workspace	`1`, `2`, `3`... *n*
`action`	open action, if opened in your workspace	one of `add`, `edit`, `delete`, `branch`, 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
`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

Field Name	Description	Example/Notes
`otherLock`*n*	for each user with the file locked, the workspace and user holding the lock	`user123@workstation9`
`otherAction`*n*	for each user with the file open, the action taken	one of `add`, `edit`, `delete`, `branch`, or `integrate`
`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.
`fileSize`	file length in bytes (requires `-Ol` option, may be expensive to compute)	`63488`

Options

`-c` *`changelist#`*	Display only files affected after the given changelist number. This operation is much faster than using a revision range on the affected files.
`-e` *`changelist#`*	Display only files affected by the given changelist number. This option is much faster than using a revision range on the affected files.
`-Ol`	Include a `fileSize` field displaying the length of the file. Note that 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.
`-Ru`	Limit output to open files that are unresolved.
`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`

- 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.

Examples

`p4 fstat file.c`	Displays information on `file.c`
`p4 fstat -Rc 20 *.c`	Displays information on all `.c` files affected since checking-in of files under changelist 20.
`p4 fstat -Os file.c`	No client workspace information lines (i.e. `clientFile`) are displayed

Related Commands

To read additional information about each file	`p4 files`
To display file information including change descriptions	`p4 filelog`

p4 group

Synopsis

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

Syntax

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

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 `unlimited`. 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 `unlimited`. See the *Usage Notes* below for more details.
`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).
`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.

Field Name	Type	Description
`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.

Options

`-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`

- 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 for groups of users.

  If either of the `maxresults` or `maxscanrows` 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 (ignoring any settings still at the default value of `unlimited`). If a particular user is not in any groups, her `maxresults` value is `unlimited`. (The user's `maxscanrows` value is computed in the same way.)

The speed of most server hardware should make it unnecessary to ever set a `maxresults` value below 10,000, or a `maxscanrows` value below 50,000.

The commands that are affected by the `maxresults` and `maxscanrows` values are:

Command	**Counted Entity**	**How Affected Users Can Reduce Command Output**
`p4 changes`	changes	Using `p4 changes -m` *`numchanges`*.
`p4 changes` *`files`*	file revisions	Use a more restrictive file pattern on the command line.
`p4 diff2`	files	Use a more restrictive file pattern on the command line.
`p4 filelog`	file revisions	Use a more restrictive file pattern on the command line.
`p4 files`	files	Use a more restrictive file pattern on the command line.
`p4 fixes`	fixes	The `-c` *`changenum`* or `-j` *`jobname`* flags restricts this command appropriately.
`p4 fixes` *`files`*	files	Use a more restrictive file pattern on the command line.
`p4 integrate`	files	Use a more restrictive file pattern on the command line.
`p4 integrated`	file revisions	Use a more restrictive file pattern on the command line.
`p4 jobs`	jobs	The `-e` *`jobquery`* flag restricts the output to those jobs that meet particular criteria.
`p4 jobs` *`files`*	file revisions	Use a more restrictive file pattern on the command line.
`p4 labelsync`	files	Use a more restrictive file pattern and the `-a` flag to build the label's file set in pieces.

Command	Counted Entity	How Affected Users Can Reduce Command Output
`p4 print`	files	Use a more restrictive file pattern on the command line.
`p4 sync`	files, as mapped through client view	Use a more restrictive file pattern on the command line.

- 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. Users in no groups have the default `timeout` value of `43200`.

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 [user]
```

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.

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`

- To see all the members of a particular group, use `p4 group -o` *`groupname`*. This variation of `p4 group` requires only `list` access.

Examples

`p4 groups bob`	Display the names of all groups of which user `bob` is a member.

Related Commands

To create or edit an existing group of users	`p4 group`
To view a list of all the members and specifications of a particular group	`p4 group -o` *`groupname`*
To set 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 Specification*
`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
```

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: 2000/07/28 12:11:47 -0700 PDT
Server version: P4D/FREEBSD/2000.1/16375 (2000/07/25)
Server license: P4Admin <p4adm> 20 users on freebsd (expires 2001/01/01)
```

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

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

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, *`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.

Multiple files can be specified by using wildcards in *`fromFile`* and *`toFile`*. If so, 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, Perforce won't copy *`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 P4Win, the Perforce Windows Client; 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`	Allow non-conforming adds and deletes. By default, a non-existent *`toFile`* is only opened for `branch` or `add` if *`fromFile`* conforms to the condition that its *`revRange`* starts with a *`branch`* or *`add`*. (When *`revRange`* is not given, this condition is always met, because the implied *`revRange`* is `#1` to `#head`.) The `-d` flag allows a non-existent *`toFile`* to be opened for `branch` or `add` even if the first revision of *`fromFile`* in *`revRange`* is an `edit` or an `integrate`. An existing *`toFile`* is only opened for `delete` if it conforms to the condition that all of its revisions are already accounted for in previous integrations to or from *`fromFile`*. In other words, *`toFile`* is only opened for `delete` if all of its changes either came from *`fromFile`* or have been merged into *`fromFile`*. The `-d` flag allows an existing *`toFile`* to be opened for `delete` even if it doesn't conform to these conditions.

`-Dt` `-Ds` `-Di`	The `-Dt` flag allows integration around a deleted target file; the source file is branched onto the deleted target. 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`.
- You can use `p4 integrate` to rename files. The method is described in the `p4 rename` description.
- `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 specification	`p4 branch`
To view a list of existing branch specifications	`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 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`*.
`copy from`	*`file`* was integrated from *`partner-file`*, accepting *`theirs`*.
`copy into`	*`file`* was integrated into *`partner-file`*, accepting *`theirs`*.
`ignored`	*`file`* was integrated from *`partner-file`*, accepting *`yours`*.
`ignored by`	*`file`* was integrated into *`partner-file`*, accepting *`yours`*.
`delete from`	*`file`* was integrated from *`partner-file`*, and *`partner-file`* had been previously deleted.
`delete into`	*`file`* was integrated into *`partner-file`*, and *`file`* had been previously deleted.

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

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`*.
`-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 may 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 straight text.

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 inequality 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 inequality 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 inequality 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`*`=*`"

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 has five parts, and is of the form *`code name type 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. • `datatype`: the datatype of the field. Possible values are: • `word`: a single arbitrary word • `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.

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.

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 P4Win, the Perforce Windows 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` 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. Please make your comments complete and understandable.
- The first line of each field's comment is also used by P4Win, the Perforce Windows Client, to display tooltips. The first line of each field's comment should be readable on its own.
- 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.

Only the `Owner:` of an `unlocked` label may 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	The label's owner. By default, the user who created the label. Only the owner of a label may update what 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`.
`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 file[`*`revrange`*`]`

Description

`p4 labels` lists all the labels known to the Perforce server in the form:

`Label `*`labelname date description`*

To see a list of labels containing specific revisions, specify the revision range.

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`

- To see a list of files included in 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] -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, then that 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 may 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, even if some of the files being labeled are deleted at their head revision.
`-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.
`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`

- 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 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 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.

The Perforce Defect Tracking Integration (P4DTI) uses `p4 logger`.

Options

`-c` *`sequence#`*	List all events happening after this sequence number.
`-t` *`countername`*	List all events after this counter number.
`-c` *`changelist#`* `-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 propagating 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 may 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, and only for the IP address of the workstation of the user that issued the `p4 login` command.

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	`none`

- The default timeout value of 43200 seconds (12 hours) is defined on a per-group basis 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 may obtain login tickets for users other than themselves without entering passwords. Non-superusers may obtain tickets for other users if and only if they correctly supply the other user's password.

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	`none`

Examples

`p4 logout`	Log out of Perforce by removing the local session ticket.
`p4 logout -a`	Log out of Perforce by invalidating the ticket, instructing the Perforce server to log this user out from any and all 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 monitor

Synopsis

Display Perforce process information

Syntax

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

Description

You must enable monitoring on the Perforce server for `p4 monitor` to work. This is done by setting the `monitor` counter with `p4 counter`, and restarting the server.

`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 `admin` 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 forms require `admin` level access.

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`, `admin`

- 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. You 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.

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 `admin` access.
`p4 monitor show -a`	Show arguments and commands, limited to 80 characters per line of output. Requires `admin` access.

`p4 monitor terminate 123`	Instruct the Perforce server to mark process 123 for termination. Requires `admin` access.
`p4 monitor clear all`	Clears the monitor table of all entries. Requires `admin` access.

Related Commands

To turn on server monitoring (requires server restart)	`p4 counter -f monitor 1`
To turn off server monitoring (requires server restart)	`p4 counter -f monitor 0`

p4 obliterate

Synopsis

Removes files and their history from the depot.

Syntax

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

Warning

The `p4 delete` command marks the latest revision as deleted, but leaves the file information intact in the depot. As such, recovery from the server data is always possible.

In contrast, `p4 obliterate` deletes the file data itself, precluding any possibility of recovery.

Use `p4 obliterate` ***with caution.*** This is the only command in Perforce that actually removes file data.

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.

The `-z` flag is used with branches; after branching a file from one area of the depot into another. When a branch is made, a "lazy copy" is performed - the file itself isn't copied; only a record of the branch is made. If you want to "obliterate" the lazy copy performed by the branch, thereby creating an *extra* copy of the file in the depot, use `p4 obliterate -z` on it. Note that this typically *increases* disk space usage.

Options

`-y` *`filespec`*	Perform the obliterate operation. Without this flag, `p4 obliterate` merely reports what it would do.
`-z` *`filespec`*	Undo “lazy copies” only; remove no files nor metadata. This option is most commonly used when working with branches in order to ensure that no other files in the database refer to the named 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	`admin`

- `p4 obliterate` is most often used to reclaim disk space from files that are no longer required, or to clean up mistakes made by users who, for instance, may have created a file hierarchy in the wrong place.
- 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.
- To undo lazy copies, the `-z` option also requires the `-y` option.

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`

p4 opened

Synopsis

List files that are open in pending changelists.

Syntax

```
p4 [g-opts] opened [-a] [-c changelist#] [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.

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` *`changelist#`*	List the files in pending changelist *`changelist#`*. To list files in the default changelist, use `p4 opened -c default`.
`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 clients.

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. For more about how ticket-based 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 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 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 MD5 hash of the password to the server.
- Passwords may 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`*
- The maximum password length is 1024 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 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`

- `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`;
- Limit access to particular IP addresses, so that only users at these IP addresses can run Perforce.

Perforce provides seven levels of access. The access levels are:

Access Level	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.
`open`	This gives the user permission to do everything she can do with `read` access, and gives her permission to `p4 add`, `p4 edit`, and `p4 delete` files. However, the user is not allowed to lock files or submit files to the depot.
`write`	The user can do all of the above, and can also write files with `p4 submit` and lock them with `p4 lock`.

Access Level	What the User Can Do
`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 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`, 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.
Host	The IP address. Use the `*` wildcard to refer to all IP addresses.
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 granted, not denied, access.

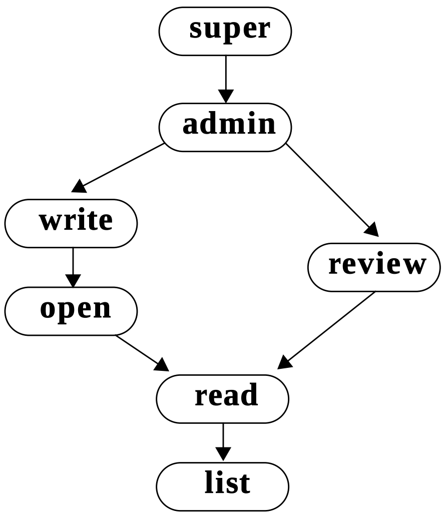
Options

`-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 access level includes all the access levels below it, as illustrated in this chart:



- Access levels determine which commands you may 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	Command	Access Level
`add`	`open`	`job` [b e]	`open`
`admin`	`super`	`jobs` [a]	`list`
`annotate`	`read`	`jobspec` [b]	`admin`
`branch` [e]	`open`	`label` [a e]	`open`
`branches`	`list`	`labels` [a b]	`list`
`change` [e]	`open`	`labelsync`	`open`
`changes` [a]	`list`	`lock`	`write`
`client` [e]	`list`	`login`	`none`
`clients`	`list`	`logout`	`none`
`counter` [c]	`review`	`obliterate`	`admin`
`counters`	`list`	`opened`	`list`
`delete`	`open`	`passwd`	`list`
`depot` [a b]	`super`	`print`	`read`
`depots` [a]	`list`	`protect` [a]	`super`
`describe`	`read`	`reopen`	`open`
`describe -s`	`list`	`resolve`	`open`
`diff`	`read`	`resolved`	`open`
`diff2`	`read`	`revert`	`open`
`dirs`	`list`	`review` [a]	`review`
`edit`	`open`	`reviews` [a]	`list`
`filelog`	`list`	`set`	`list`
`files`	`list`	`submit`	`write`
`fixes` [a]	`list`	`sync`	`read`
`fstat`	`list`	`tag`	`open`
`group` [a b]	`super`	`tickets`	`none`
`groups` [a]	`list`	`triggers`	`super`

Command	Access Level	Command	Access Level
`have`	`list`	`typemap`	`admin`
`help`	`none`	`unlock` [e]	`open`
`info`	`none`	`user` [a b]	`list`
`integrate` [d]	`open`	`users` [a]	`list`
`integrated`	`list`	`verify`	`admin`
		`where` [a]	`none`

[a] This command doesn't operate on specific files. Thus, permission is granted to run the command if the user has the specified access to at least one file in the depot.
[b] The `-o` flag to this command, which allows the form to be read but not edited, requires only `list` access.
[c] `list` access is required to view an existing counter's value; `review` access is required to change a counter's value or create a new counter.
[d] To run `p4 integrate`, the user needs `open` access on the target files and `read` access on the donor files.
[e] The -f flag to override existing metadata or other users' data requires `admin` access.

- 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.

  It is possible to deny yourself `super` access; if you accidentally deny yourself `super` access, you will subsequently be unable to run `p4 protect`. To get around this, remove the `db.protect` table under `P4ROOT` of the Perforce server.
- 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 a different server (as defined by `p4 depot`), create protections for user `remote`. Remote depots are always accessed by a virtual user named `remote`.
- For further information, consult 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.*   //...
```

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).

From IP address...	Joe tries...	Results
`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 rename

Synopsis

Renaming files under Perforce.

Syntax

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

Description

Although Perforce doesn't have a `rename` command, renaming a file can be 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 this 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 integrate -c 413 //depot/p2/... //depot/guiProj/...`
 	
`p4 delete -c 413 //depot/p2/...`
 	
`p4 submit -c 413`
	Renaming a set of files, in three steps:
	• `p4 integrate` copies all the files in the `p2` directory to the `guiProj` directory.
	• `p4 delete` deletes all files in the `p2` directory.
	• `p4 submit` makes these changes to the depot in a single atomic changelist.

Related Commands

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 may 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`*

p4 resolve

Synopsis

Resolve conflicts between file revisions.

Syntax

```
p4 [g-opts] resolve [-af -am -as -at -ay -db -dw -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. 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 choose whether to overwrite the file revision in the depot with the file in the client workspace, 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 (overwrite the file in the depot with the file in the workspace, or overwrite the file in the workspace with the file in the depot) are normally available, since merges don't generally work with binary files.

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 `p4 resolve` dialog presents the following options:

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 client.	yes
`et`	edit theirs	Edit the revision in the depot that the client revision conflicts with (usually the head revision). This edit is read-only.	yes
`dy`	diff yours	Show diffs between *`yours`* and *`base`*.	no
`dt`	diff theirs	Show diffs between *`theirs`* and *`base`*.	no

Option	Short Meaning	What it Does	Available by Default for Binary Files?
`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

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` options are chosen, or if the file is skipped with the `skip` option.

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, but not both, of *yours* and *theirs* is different from *base*, 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.

`-db` `-dw`	When merging files, ignore specified differences in whitespace. (If you use these flags, and the files differ in whitespace only, `p4 resolve` will use the text in the client 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)
`-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.

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 revert

Synopsis

Discard changes made to open files.

Syntax

```
p4 [g-opts] revert [ -n ] [ -a -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 files you've opened with `p4 delete` are reverted, 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 since they were opened. Specifically, the only files reverted are those whose client revisions are: • open for edit but have unchanged content; 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.
`-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	`open`

- `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 //depot/...`	Revert all open files to their pre-opened state.
`p4 revert -c default //...`	Revert all open files in the default changelist to their pre-opened state.
`p4 revert -n *.txt`	Preview a reversion of all open files with the suffix `.txt` in the current directory, but don't actually perform the revert.
`p4 revert -c 31 *.txt`	Revert all files in changelist 31 with the suffix `.txt` in the current directory to their pre-opened state.

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 will display 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	`list`

- 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;
  - The *global option* flags 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 submit

Synopsis

Send changes made to open files to the depot.

Syntax

```
p4 [g-opts] submit [-r] [-s] [files]
p4 [g-opts] submit [-r] -c changelist#
p4 [g-opts] submit -i [-r] [-s]
```

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, files open for `edit`, `add`, and `branch` are closed when submitted. Use the `-r` (reopen) flag if you want files reopened for `edit` after submission.

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.

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.
`Files:`	List	A list of files being submitted in this changelist. Files may be deleted from this list, but may not 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.
`-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`. This option works in conjunction with the `-s` option to `p4 fix`, and is intended for use by Perforce Defect Tracking Integration (P4DTI).
`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 may 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.

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] [-n] [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 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.

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

`-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.
`-f`	Force the sync. Perforce performs the sync even if the client workspace already has the file at the specified revision, and even if the file is not writable. This flag does not affect open files, but it *does* override the `noclobber` client 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
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, 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 be 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.

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 218.

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	`open`

- 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 `rel1`. If the label `rel1` 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`

- On Windows and UNIX, tickets are stored in `%USERPROFILE%\p4tickets.txt` and `$HOME/.p4tickets` respectively.

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 or forms are updated.

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.

There are six trigger types. The first three trigger types (`submit`, `content`, and `commit`) are fired when users submit changelists, and are referred to as *changelist submission triggers*. The remaining trigger types (`save`, `out`, and `in`) are fired when users generate or modify form specifications, and are referred to as *specification triggers*.

Use the `submit` trigger type to create triggers that fire after changelist creation, but before files are transferred to the server. Because `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 `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 `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.

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

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.

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

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. Each row in the field holds four values:

Field	Meaning
`name`	The user-defined name of the trigger.
`type`	There are six trigger types. The first three trigger types (`submit`, `content`, and `commit`) are fired when users submit changelists, and are referred to as *changelist submission triggers*. The remaining trigger types (`save`, `out`, and `in`) are fired when users generate or modify form specifications, and are referred to as *specification triggers*. • `submit`: Execute a changelist trigger after changelist creation, but before file transfer. Trigger may not access file contents. • `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. • `commit`: Execute a changelist trigger after changelist creation, file transfer, and changelist commit. • `save`: Execute specification trigger after its contents are parsed, but before its contents are stored in the Perforce database. Trigger may not modify form specified in `%formfile%` variable. • `out`: Execute specification trigger upon generation of form to end user. Trigger may modify form. • `in`: Execute specification trigger on edited form before contents are parsed and validated by the Perforce server. Trigger may modify form.

Field	Meaning
path	For changelist submission triggers (`submit`, `content`, or `commit`), 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 specification triggers (`save`, `out`, or `in`), the name of the type of form, such as `branch`, `client`, and so on. Triggers that fire on the `p4 triggers` command are ignored.
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 `submit` and `content` triggers, changelist submission continues if the trigger script exits with 0, or fails if the script exits with a nonzero value. For `commit` triggers, changelist submission succeeds regardless of the trigger script's exit code, but subsequent `commit` triggers do not fire if the script exits with a nonzero value. For `in`, `out`, and `save` triggers, the data in the specification becomes part of the Perforce database if the script exits with 0. Otherwise, the database is not updated.

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.

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

Argument	Description	Available for type
`%changelist%`	The number of the changelist being submitted. (The abbreviated form `%change%` is equivalent.)	`submit`, `content`, and `commit`
`%client%`	Triggering user's client workspace name.	all
`%clienthost%`	Hostname of the client.	all
`%clientip%`	The IP address of the client.	all
`%serverhost%`	Hostname of the Perforce server.	all
`%serverip%`	The IP address of the server.	all
`%serverport%`	The IP address and port of the Perforce server, in the format *`ip_address`*`:`*`port`*.	all
`%serverroot%`	The `P4ROOT` directory of the Perforce server.	all
`%user%`	Perforce username of the triggering user.	all
`%formfile%`	Path to temporary 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`.	`save`, `out`, and `in`
`%formname%`	Name of form (for instance, a branch name or a changelist number).	`save`, `out`, and `in`
`%formtype%`	Type of form (for instance, `branch`, `change`, and so on).	`save`, `out`, and `in`

- 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 three specification trigger types (in, out, and save), 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. After job creation, subsequent user changes to a job correctly set %formname% for use by specification trigger scripts.

Examples

Suppose that the trigger table consists of the following entries:

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

Both the first and fourth 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 1024 bytes of a file. If the high bit is clear in each of the first 1024 bytes, Perforce assumes it to be `text`; otherwise, it's `binary`.

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 1024 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`

- 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, or enforcing the preservation of original file modification times (the `+m` file type modifier) in directories of third-party DLLs.
- 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.

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 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. 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. (After installing Perforce, use `p4 protect` as a Perforce superuser to prevent automatic creation of new users.)

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.
`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).
`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.
`-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 may 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`

p4 users

Synopsis

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

Syntax

```
p4 [g-opts] users [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 may contain the `*` wildcard; in this case, all users matching the given pattern are reported on. (If using wildcards, be sure to quote the user argument, since the OS will likely attempt to expand the wildcard to match file names in the current directory).

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 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 [ -q -u -v ] file[revRange]...
```

Description

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

If invoked without arguments, `p4 verify` computes and displays the 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!`

To save MD5 fingerprints in the Perforce database, use `p4 verify -u`. Subsequent invocations of `p4 verify` compute checksums for the desired files and compare them against those stored by `p4 verify -u`. If the checksums differ, the output line for the corrupt file ends with `BAD!`

Once stored, a digest is not recomputed unless `p4 verify -v` flag is used to overwrite it. The `-v` flag is generally used only to update the saved digest of archive files which have been deliberately altered outside of Perforce control by a Perforce system administrator.

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 MD5 digest of each file in the Perforce database if and only if no 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.
`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 //...`

  Because subsequent verifications can only be performed against previously stored signatures, it is also good practice to regularly generate checksums with `p4 verify -u`.

  For more about good administrative practices, see the *Perforce System Administrator's Guide*.
- As of Release 2003.2, `p4 verify -u` is obsolescent, because Perforce Servers at Release 2003.2 and higher automatically generate and store MD5 checksums of files upon file submission. (You must still run `p4 verify -u` at least once following an upgrade to 2003.2, to generate signatures for any pre-2003.2 files for which signatures were not generated.)

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	`none`

- 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/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%40.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`

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 in each OS and shell:

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 MPW	`set -e P4CLIENT `*`value`*
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` chapter for more details on setting Perforce's registry variables in Windows).

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

- *Crucial*: The variable almost certainly needs to be set on the client; the 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 of this variable 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 categorized twice.

Crucial Variables	Useful Variables	Esoteric Variables	Server Variables
`P4CLIENT`	`P4CONFIG`	`P4PAGER`	`P4JOURNAL`
`P4PORT`	`P4DIFF`	`PWD`	`P4LOG`
`P4PASSWD`	`P4EDITOR`	`TMP, TEMP`	`P4PORT`
`P4USER`	`P4MERGE`	`P4LANGUAGE`	`P4ROOT`
	`P4CHARSET`		`P4DEBUG`

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.

Examples

```
iso8859-1
iso8859-15
eucjp
shiftjis
winansi
macosroman
```

Notes

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

For servers operating in the default (non-unicode mode), `P4CHARSET` must be left unset. 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. 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, see the *Release Notes*.

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
```

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`
- `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 NT 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.*

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.

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	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\p4winmrg.exe
c:\progra~1\Perforce\p4winmrg.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
```

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 via P4Win (the Perforce Windows Client) the value of the registry variable `P4PASSWD` is set for you. Setting the password in P4Win 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*.

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`
`perforce.squid.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.

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.

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

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, the name and port number of the target Perforce server (that is, the Perforce server for which P4P acts as a proxy).

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	`p4p -t` *`host`*`:`*`port`*	N/A

Value if not Explicitly Set

Program	Value
Perforce Proxy	`perforce:1666`

Examples

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

Notes

The format of `P4TARGET` on the Perforce Proxy is *`host`*`:`*`port`*, or *`port`* by itself if both the Perforce server is running on the same host (an unlikely configuration).

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

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

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*.

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 *Perforce User's Guide*, which is available at our web site: `http://www.perforce.com`.

Global Options

Synopsis

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

Syntax

```
p4 [-cclient -ddir -Hhost -pport -Ppass -uuser -xfile -Ccharset] [-G] [-s] 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.
`-L` *language*	This feature is reserved for system integrators.
`-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, 'r'))
  ```

  In some cases, it may not be intuitively obvious what keys the client program uses. 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)
          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 may be 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 may be specified by their absolute paths or relative to the current working directory. (Relative path components may 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.

Using revision specifiers

File specifiers may 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` may 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*.
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` may 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, `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 “unprintable” (non-ASCII) characters in filenames, label names, client workspace names, and other identifiers.

The pathname component separator (/) and recursive subdirectory wildcards (...) are 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 thus specified are then 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, client, or label name, and followed by the actual file name(s) within the depot or client.

File specifications within mappings may contain the usual Perforce wildcards of `*`, `...`, and the substring positional specifiers `%%1` through `%%9`.

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.

If a path listed in a client view contains spaces, make sure to quote the path:

```
//depot/dir1/... "//cws/dir one/..."
```

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 checked out to a client workspace, opened for editing, edited, and checked back into the depot.

When files are checked out, 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/...`

Some files in the depot are excluded from the client workspace	`//depot/dir1/... //cws/...`
	`-//depot/dir1/exclude/... //cws/dir1/...`

Client View	Sample Mapping
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 client 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.

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. This may be done 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 may be tagged 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.

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` 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 a file is opened for `add`, Perforce attempts to determine the type of the file automatically. If the file is a regular file or a symbolic link, its type is set accordingly. Perforce then examines the first 1024 bytes of the file to determine whether it is `text` or `binary`. If any non-text characters are found, the file is assumed to be `binary`; otherwise, the file is assumed to be `text`.

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 1024 or more characters of ASCII text, and might otherwise be mistaken for `text` files.

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

Keyword	Description	Comments	Server Storage
`apple`	Multi-forked Macintosh file	AppleSingle storage of Mac data fork, resource fork, file type and file creator. New to Perforce 99.2. 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 we recommend using the new `apple` file type instead. For full details, please see the Mac client release notes.	full file, compressed
`unicode`	Unicode file	Perforce servers operating in internationalized mode support a Unicode file type. These files are translated into the local character set. For details, see the *System Administrator's Guide*.	deltas in RCS format, stored as UTF-8

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.

Modifier	Description	Comments
`+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.
`+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.
`+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.

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`.

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
`.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

File Type	Perforce file type	Description
`.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
`.zip`	`binary+F`	ZIP compressed archive 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 on non-UNIX clients 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 client.
- Changing a file's type does not affect earlier revisions stored in the depot.

  For instance, changing a file's type by adding the `+S` (temporary object) modifier tells Perforce to store only the head revision of the file in the depot. If you change an existing file into a temporary object, subsequent revisions will purge the one stored at the old head revision, but revisions to the file stored in the depot *before* the `+S` modifier was used will remain unaffected. (Syncing to a non-head revision submitted *after* the `+S` 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. (Normally, Perforce updates the timestamp when a file is synced.) It allows 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	`ubinary`	`+FSw`
`ctempobj`	Temporary object (compressed)	`cbinary`	`+Sw`
`xtempobj`	Temporary executable object	`ubinary`	`+FSwx`
`xunicode`	Executable unicode	`unicode`	`+x`

Index