
Perforce 2007.3 Command Reference

December 2007

This manual copyright 1999-2007 Perforce Software.

All rights reserved.

Perforce software and documentation is available from <http://www.perforce.com>. You may download and use Perforce programs, but you may not sell or redistribute them. You may download, print, copy, edit, and redistribute the documentation, but you may not sell it, or sell any documentation derived from it. You may not modify or attempt to reverse engineer the programs.

Perforce programs and documents are available from our Web site as is. No warranty or support is provided. Warranties and support, along with higher capacity servers, are sold by Perforce Software.

Perforce Software assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

By downloading and using our programs and documents you agree to these terms.

Perforce and Inter-File Branching are trademarks of Perforce Software. Perforce software includes software developed by the University of California, Berkeley and its contributors.

All other brands or product names are trademarks or registered trademarks of their respective companies or organizations.

Table of Contents

About This Manual	7
p4 add	9
p4 admin	12
p4 annotate	15
p4 branch	17
p4 branches	20
p4 change	21
p4 changelists	25
p4 changelist	26
p4 changes	27
p4 client	30
p4 clients	38
p4 counter	40
p4 counters	43
p4 delete	44
p4 depot	46
p4 depots	50
p4 describe	51
p4 diff	53
p4 diff2	56
p4 dirs	60
p4 edit	62
p4 filelog	65
p4 files	68
p4 fix	70
p4 fixes	73
p4 flush	75
p4 fstat	78
p4 group	83
p4 groups	87
p4 have	89
p4 help	91

p4 info	93
p4 integrate	94
p4 integrated	100
p4 job	102
p4 jobs	105
p4 jobspec	110
p4 label	114
p4 labels	117
p4 labelsync	119
p4 license	121
p4 lock	122
p4 logger	123
p4 login	124
p4 logout	126
p4 monitor	128
p4 obliterate	131
p4 opened	134
p4 passwd	136
p4 print	139
p4 protect	141
p4 protects	150
p4 rename	152
p4 reopen	153
p4 resolve	155
p4 resolved	161
p4 revert	163
p4 review	165
p4 reviews	167
p4 set	169
p4 sizes	172
p4 submit	174
p4 sync	179
p4 tag	183
p4 tickets	185
p4 triggers	186
p4 typemap	195
p4 unlock	199

p4 user	200
p4 users	204
p4 verify	205
p4 where.....	207
p4 workspace.....	209
p4 workspaces.....	210

Environment and Registry Variables 211

P4AUDIT.....	213
P4CHARSET	214
P4COMMANDCHARSET.....	215
P4CLIENT	216
P4CONFIG.....	217
P4DEBUG.....	219
P4DIFF	220
P4DIFFUNICODE	221
P4EDITOR.....	222
P4HOST.....	223
P4JOURNAL.....	224
P4LANGUAGE	225
P4LOG	226
P4MERGE	227
P4MERGEUNICODE	228
P4PAGER	229
P4PASSWD	230
P4PCACHE.....	231
P4PFSIZE	232
P4OPTIONS	233
P4PORT	234
P4ROOT	235
P4TARGET.....	236
P4TICKETS	237
P4USER	238
PWD.....	239
TMP, TEMP	240

Additional Information 241

Global Options 243

File Specifications..... 247

Views 253

File Types 257

Index..... 265

About This Manual

Synopsis

This is the *Perforce 2007.3 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, <i>File Specifications, Views, Global Options, File Types</i>
Client workspace	p4 client, p4 clients, p4 flush, p4 have, p4 sync, p4 where, p4 workspace, p4 workspaces
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 sizes, p4 unlock
Changelists	p4 change, p4 changelist, p4 changes, p4 changelists, 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 license, 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 protects, p4 tickets, p4 user, p4 users
Environment	p4 set, <i>Environment and Registry Variables</i> , P4AUDIT, P4CHARSET, P4COMMANDCHARSET, P4CLIENT, P4CONFIG, P4DEBUG, P4DIFF, P4DIFFUNICODE, P4EDITOR, P4HOST, P4JOURNAL, P4LANGUAGE, P4LOG, P4MERGE, P4MERGEUNICODE, P4PAGER, P4PASSWD, P4PCACHE, P4PFSIZE, P4POPTIONS, P4PORT, P4ROOT, P4TARGET, P4TICKETS, 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 *P4 User's Guide*, available from our web site at: <http://www.perforce.com>.

Options

This manual is available in PDF and HTML.

Usage Notes

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

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

p4 add

Synopsis

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

Syntax

```
p4 [g-opts] add [-c changelist#] [-f -n] [-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 8192 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

<code>-c changelist</code>	Opens the files for add within the specified <i>changelist</i> . If this flag is not used, the files are linked to the default changelist.
<code>-t filetype</code>	Adds the file as the specified <i>filetype</i> . Please see the <i>File Types</i> chapter for a list of Perforce file types.
<code>-f</code>	Use the <code>-f</code> flag to force inclusion of wildcards in filenames. See the <i>File Specifications</i> chapter for details.
<code>-n</code>	Preview which files would be opened for add, without actually changing any files or metadata.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

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

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

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

Examples

<code>p4 add -t binary file.pdf</code>	Assigns a specific file type to a new file, overriding any settings in the typemap table
<code>p4 add -c 13 *</code>	Opens all the files within the user's current directory for add, and links these files to changelist 13.

<code>p4 add README ~/src/*.c</code>	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.
<code>p4 add -f *.c</code>	<p>Opens a file named *.c for add.</p> <p>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.</p> <p>For more information, see “Limitations on characters in filenames and entities” on page 250.</p>

Related Commands

To open a file for edit	<code>p4 edit</code>
To open a file for deletion	<code>p4 delete</code>
To copy all open files to the depot	<code>p4 submit</code>
To read files from the depot into the client workspace	<code>p4 sync</code>
To create or edit a new changelist	<code>p4 change</code>
To list all opened files	<code>p4 opened</code>
To revert a file to its unopened state	<code>p4 revert</code>
To move an open file to a different pending changelist	<code>p4 reopen</code>
To change an open file's file type	<code>p4 reopen -t filetype</code>

p4 admin

Synopsis

Perform administrative operations on the server.

Syntax

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

Description

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

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

To take a checkpoint, use `p4 admin checkpoint [prefix]`. This is equivalent to logging in to the server machine and taking a checkpoint with `p4d -jc [prefix]`. A checkpoint is taken and the journal is copied to a numbered file. If a *prefix* is specified, the files are named *prefix.ckp.n* or *prefix.jnl.n-1* respectively, where *n* is a sequence number. You can store checkpoints and journals in the directory of your choice by specifying the directory as part of the prefix. (Rotated journals are stored in the `P4ROOT` directory, regardless of the directory in which the current journal is stored.) If no *prefix* is specified, the default filenames `checkpoint.n` and `journal.n-1` are used.

The `p4 admin journal` command is equivalent to `p4d -jj`. For details, see the *System Administrator's Guide*. The files are created in the server root specified when the Perforce server was started.

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

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

Options

<code>-z</code>	For <code>p4 admin checkpoint</code> and <code>p4 admin journal</code> , save the checkpoint and saved journal file in compressed (gzip) format, appending the <code>.gz</code> suffix to the files.
<code>-h</code>	For <code>p4 admin dbstat</code> , display a histogram showing distances between leaf pages.
<code>-a</code>	For <code>p4 admin dbstat</code> , display statistics for all tables. For <code>p4 admin updatespecdepot</code> , update the spec depot with all current forms.
<code>dbtable</code>	For <code>p4 admin dbstat</code> , display statistics for only the specified tables (for instance, <code>db.have</code> , <code>db.user</code> , and so on.)
<code>-s type</code>	For <code>p4 admin updatespecdepot</code> , update the spec depot with forms of the specified <code>type</code> , where <code>type</code> is one of <code>client</code> , <code>depot</code> , <code>branch</code> , <code>label</code> , <code>typemap</code> , <code>group</code> , <code>user</code> , or <code>job</code> .
<code>g-opts</code>	See the <i>Global Options</i> 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.
- Because `p4 admin dbstat` locks out write access to the database while it scans the tables, use this command with care. You will most often use this command when working with Perforce technical support.
- The spec depot must exist before running `p4 admin updatespecdepot`.
- 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 <code>checkpoint.n</code> , and start a new journal named <code>journal</code> , copying the old journal file to <code>journal.n-1</code> , where <i>n</i> is a sequence number.
p4 admin checkpoint name	Create a checkpoint named <code>name.ckp.n</code> , and start a new journal named <code>journal</code> , copying the old journal file to <code>name.jnl.n-1</code> , where <i>n</i> is a sequence number.

p4 annotate

Synopsis

Print file lines along with their revisions.

Syntax

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

Description

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

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

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

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

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

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

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

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

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

Options

<code>-a</code>	All lines, including deleted lines and lines no longer present at the head revision, are included. Each line includes a starting and ending revision.
<code>-c</code>	Display the changelist number, rather than the revision number, associated with each line. If you use the <code>-a</code> option and the <code>-c</code> option together, each line includes a starting and ending changelist number.
<code>-dflags</code>	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the <i>Usage Notes</i> below for a listing of these flags.

<code>-i</code>	Follow file history across branches. If a file was created by branching, Perforce includes revisions up to the branch point. The use of <code>-i</code> option implies the <code>-c</code> option.
<code>-q</code>	Quiet mode; suppress the one-line header for each file.
<code>g-opts</code>	See the <i>Global Options</i> 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.
- The diff flags supported by `p4 annotate` are:

Flag	Name
<code>-db</code>	ignore changes made within whitespace
<code>-dw</code>	ignore whitespace altogether

Examples

<code>p4 annotate file.c</code>	Print all lines of <code>file.c</code> , each line preceded by the revision that introduced that line into the file.
<code>p4 annotate -c file.c</code>	Print all lines of <code>file.c</code> , each line preceded by the changelist number that introduced that line into the file.
<code>p4 annotate -a file.c</code>	Print all lines of <code>file.c</code> , including deleted lines, each line preceded by a revision range. The starting and ending revision for each line are included.
<code>p4 annotate -a -c file.c</code>	Print all lines of <code>file.c</code> , 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 <i>source files</i>) to another set of files in the depot (the <i>target files</i>). The view maps from one location in the depot to another; it can't refer to a client workspace. For example, the branch view <pre>//depot/main/... //depot/r2.1/...</pre> 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.
<i>g-opts</i>	See the <i>Global Options</i> 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.

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

```
//depot/project/dev/... "//depot/project/release 2.0/..."
```
- Branch views can also be used with `p4 diff2` with the syntax `p4 diff2 -b branchname fromFiles`. This will diff the files that match the pattern *fromFiles* against their corresponding *toFiles* as defined in the branch view.

Related Commands

To view a list of existing branch specifications	<code>p4 branches</code>
To copy changes from one set of files to another	<code>p4 integrate</code>
To view differences between two codelines	<code>p4 diff2</code>

p4 branches

Synopsis

List existing branch specifications.

Syntax

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

Description

- Print the list of all branch specifications currently known to the system.
- Use the `-m max` option to limit the output to the first `max` branch specifications.
- Use the `-u user` option to limit the output to branches owned by the named user.

Options

<code>-m max</code>	List only the first <code>max</code> branch specifications.
<code>-u user</code>	List only branches owned by <code>user</code> .
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 branch</code>
--	------------------------

p4 change

Synopsis

Create or edit a changelist specification.

Syntax

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

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.

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

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

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

Form Fields

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

Field Name	Type	Description
Status:	Read-only	pending, submitted, or new. Not editable by the user. The status is new when the changelist is created, pending when it has been created but has not yet been submitted to the depot with <code>p4 submit</code> , and submitted when its contents have been stored in the depot with <code>p4 submit</code> .
Description:	Writable, mandatory	Textual description of changelist. This value <i>must</i> be changed before submission.
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 <code>p4 user</code> form's <code>JobView:</code> 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 or pending fixes, but the superuser can delete changelists under other circumstances with the addition of the -f flag.
-f	Force flag. Allows the description, modification date, or user of a submitted changelist to be edited. Editing a submitted changelist requires <code>admin</code> or <code>super</code> access. Superusers and administrators may also overwrite read-only fields when using the -f flag.
-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 <code>p4 obliterate</code> .
-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 <code>p4 change</code> form.

<code>-s</code>	<p>Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of <code>closed</code>.</p> <p>On new changelists, the <code>fix</code> status is displayed as the special status <code>ignore</code>. (If the status is left unchanged, the job is not fixed by the submission of the changelist.)</p> <p>This option works in conjunction with the <code>-s</code> option to <code>p4 fix</code>, and is intended for use by Perforce Defect Tracking Integration (P4DTI).</p>
<code>-u</code>	Update a submitted changelist. Only the <code>Jobs:</code> and <code>Description:</code> fields may be updated, and only the submitter of the changelist may update the changelist. The <code>-u</code> and the <code>-f</code> flags are mutually exclusive.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- 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

<code>p4 change</code>	Create a new changelist.
<code>p4 change -f 25</code>	Edit previously submitted changelist 25. Administrator or superuser access is required.
<code>p4 change -d 29</code>	Delete changelist 29. This succeeds only if changelist 29 is pending and contains no files.

Related Commands

To submit a changelist to the depot	<code>p4 submit</code>
To move a file from one changelist to another	<code>p4 reopen</code>
To remove a file from all pending changelists	<code>p4 revert</code>
To list changelists meeting particular criteria	<code>p4 changes</code>
To list opened files	<code>p4 opened</code>
To list fixes linked to particular changelists	<code>p4 fixes</code>
To link a job to a a particular changelist	<code>p4 fix</code>
To remove a job from a particular changelist	<code>p4 fix -d</code>
To list all the files listed in a changelist	<code>p4 opened -c changelist#</code>
To obtain a description of files changed in a changelist	<code>p4 describe changelist#</code>

p4 changelists

Synopsis

List submitted and pending changelists.

Syntax

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

Description

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

p4 changelist

Synopsis

Create or edit a changelist specification.

Syntax

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

Description

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

p4 changes

Synopsis

List submitted and pending changelists.

Syntax

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

Description

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

By default, the format of each line is:

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

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

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

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

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

Revision specifications and revision ranges can be included in the file patterns. Including a revision range lists all changes that affect files within the range; providing a single revision specifier lists all changes from 1 to the specified revision.

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

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

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

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

Options

<code>-i</code>	Include changelists that affected files that were integrated with the specified files.
<code>-t</code>	Display the time as well as the date of each change.
<code>-l</code>	List long output, with the full text of each changelist description.
<code>-L</code>	List long output, with the full text of each changelist description truncated at 250 characters.
<code>-c <i>client</i></code>	List only changes made from the named client workspace.
<code>-m <i>max</i></code>	List only the highest numbered <i>max</i> changes.
<code>-s <i>status</i></code>	Limit the list to the changelists with the given status (pending or submitted)
<code>-u <i>user</i></code>	List only changes made from the named user.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

Examples

<code>p4 changes -m 5 //depot/project/...</code>	Show the last five submitted changelists that include any file under the <code>project</code> directory
<code>p4 changes -m 5 -c eds_elm</code>	Show the last five submitted changelists from client workspace <code>eds_elm</code> .
<code>p4 changes -m 5 -u edk</code>	Show the last five submitted changelists from user <code>edk</code> .
<code>p4 changes file.c@2000/05/01,2000/06/01</code>	Show any changelists that include file <code>file.c</code> , as mapped to the depot through the client view, during the month of May 2000.
<code>p4 changes -m 1 -s submitted</code>	Output a single line showing the changelist number of the last submitted changelist.

```
p4 changes @2001/04/01,@now
```

Display all changelists submitted from April 1, 2001 to the present.

```
p4 changes @2001/04/01
```

Display all changelists submitted *before* April 1, 2000.

Related Commands

To submit a pending changelist

```
p4 submit
```

To create a new pending changelist

```
p4 change
```

To read a detailed report on a single changelist

```
p4 describe
```

p4 client

Synopsis

Create or edit a client workspace specification and its view.

Syntax

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

Description

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

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

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

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

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

Form Fields

Field Name	Type	Description
Client:	Read-only	The client workspace name, as specified in the <code>P4CLIENT</code> 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	<p>The name of the host machine on which this client workspace resides. If included, operations on this client workspace can be run <i>only</i> from this host.</p> <p>The hostname must be provided exactly as it appears in the output of <code>p4 info</code> when run from that host.</p> <p>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 <code>-H</code> flag to any <code>p4</code> command, or with the <code>P4HOST</code> environment variable. For a similar mechanism that does provide security, use the IP address restriction feature of <code>p4 protect</code>.</p>
Description:	Writable, optional	A textual description of the client workspace. The default text is <code>Created by owner</code> .
Root:	Writable, mandatory	The directory (on the local host) relative to which all the files in the <code>View:</code> are specified. The default is the current working directory.
AltRoots:	Writable, optional	<p>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.</p> <p>This enables users to use the same Perforce client specification on multiple platforms with different directory naming conventions.</p> <p>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 <code>AltRoots:</code> field.</p> <p>For example, an engineer building products on multiple platforms might specify a main client root of <code>C:\Projects\Build</code> for Windows builds, and an alternate root of <code>/staff/userid/projects/build</code> for any work on UNIX builds.</p>

Field Name	Type	Description
Options:	Writable, mandatory	A set of seven switches that control particular client options. See the <i>Usage Notes</i> , below, for a listing of these options.
SubmitOptions:	Writable, mandatory	<p>Flags to govern the default behavior of <code>p4 submit</code>.</p> <ul style="list-style-type: none">• <code>submitunchanged</code> All open files (with or without changes) are submitted to the depot. This is the default behavior of Perforce.• <code>submitunchanged+reopen</code> All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist.• <code>revertunchanged</code> Only those files with content or type changes are submitted to the depot. Unchanged files are reverted.• <code>revertunchanged+reopen</code> Only those files with content or type changes are submitted to the depot and reopened in the default changelist. Unchanged files are reverted and <i>not</i> reopened in the default changelist.• <code>leaveunchanged</code> Only those files with content or type changes are submitted to the depot. Any unchanged files are moved to the default changelist.• <code>leaveunchanged+reopen</code> Only those files with content or type changes are submitted to the depot. Unchanged files are moved to the default changelist, and changed files are reopened in the default changelist. This option is similar to <code>submitunchanged+reopen</code>, except that no unchanged files are submitted to the depot.
LineEnd:	Writable, mandatory	A set of four switches that control carriage-return/linefeed (CR/LF) conversion. See the <i>Usage Notes</i> , 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 <i>Views</i> for more information.

Options

<code>-t clientname</code>	Copy client workspace <i>clientname</i> 's view and client options into the <code>View:</code> and <code>Options:</code> field of this client workspace. (i.e, use <i>clientname</i> 's <code>View:</code> as a template)
<code>-f</code>	Allows the last modification date, which is normally read-only, to be set. Superusers may use this flag to delete or modify locked workspaces owned by other users, and non-superusers may use this flag to delete or modify locked workspaces that they do own.
<code>-d clientname</code>	Delete the specified client workspace, if the client is owned by the invoking user or it is unlocked. (The <code>-f</code> flag allows allows Perforce superusers to delete locked client workspaces that they don't own, or by non-superusers to delete or modify locked clients that they do own.)
<code>-i</code>	Read the client description from standard input.
<code>-o</code>	Write the client specification to standard output.
<code>g-opts</code>	See the <i>Global Options</i> 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
<code>[no]allwrite</code>	If set, unopened files on the client are left writable.	<code>noallwrite</code>
<code>[no]clobber</code>	If set, a <code>p4 sync</code> overwrites ("clobbers") writable-but-unopened files in the client that have the same name as the newly-synced files	<code>noclobber</code>
<code>[no]compress</code>	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.)	<code>nocompress</code>

Option	Choice	Default
[no]crlf	Note: 2000.2 or earlier only! On Windows, if <code>crlf</code> is set, CR/LF translation is performed automatically when copying files between the depot and the client workspace.	<code>crlf</code>
[un]locked	Grant or deny other users permission to edit the client specification (To make a <code>locked</code> client specification truly effective, you should also set the client's owner's password with <code>p4 passwd</code> .) If <code>locked</code> , only the owner is able to use, edit, or delete the client spec. Perforce administrators can override the lock by using the <code>-f</code> (force) flag with <code>p4 client</code> .	<code>unlocked</code>
[no]modtime	For files <i>without</i> the <code>+m</code> (modtime) file type modifier: <ul style="list-style-type: none">• For Perforce clients at the 99.2 level or earlier, if <code>modtime</code> is set, the modification date (on the local filesystem) of a newly synced file is the date and time <i>at the server</i> when the file was submitted to the depot.• For Perforce clients at the 2000.1 level or higher, if <code>modtime</code> is set, the modification date (on the local filesystem) of a newly synced file is the datestamp <i>on the file</i> when the file was last modified.• If <code>nomodtime</code> is set, the modification date is the date and time <i>of sync</i>, regardless of Perforce client version. For files <i>with</i> the <code>+m</code> (modtime) file type modifier: <ul style="list-style-type: none">• For Perforce clients at the 99.2 level or earlier, the <code>+m</code> modifier is ignored, and the behavior of <code>modtime</code> and <code>nomodtime</code> 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 <i>on the file</i> when the file was submitted to the depot, <i>regardless</i> of the setting of <code>modtime</code> or <code>nomodtime</code> on the client.	<code>nomodtime</code> (i.e. date and time of sync) for most files. Ignored for files with the <code>+m</code> file type modifier.

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

- By default, any user can edit any workspace specification with `p4 client -c clientname`. To prevent this from happening, set the `locked` option and use `p4 passwd` to create a password for the client workspace owner.
- The `compress` option speeds up client/server communications over slow links by reducing the amount of data that has to be transmitted. Over fast links, the compression process itself may consume more time than is saved in transmission. In general, `compress` should be set for line speeds under T1, and should be left unset otherwise.
- The `LineEnd:` field controls the line-ending character(s) used for text files in the client workspace.

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

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

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

Option	Meaning
local	Use mode native to the client (default)
unix	UNIX-style (and Mac OS X) line endings: LF
mac	Macintosh pre-OS X: CR only
win	Windows-style: CR, LF.
share	<p>The <code>share</code> option normalizes mixed line-endings into UNIX line-end format. The <code>share</code> option does not affect files that are synced into a client workspace; however, when files are submitted back to the Perforce Server, the <code>share</code> option converts all Windows-style CR/LF line-endings and all Mac-style CR line-endings to the UNIX-style LF, leaving lone LFs untouched.</p> <p>When you sync your client workspace, line endings are set to LF. If you edit the file on a Windows machine, and your editor inserts CRs before each LF, the extra CRs do not appear in the archive file.</p> <p>The most common use of the <code>share</code> option is for users of Windows workstations who mount their UNIX home directories as network drives; if you sync files from UNIX, but edit the files on a Windows machine.</p>

For more information about how Perforce uses the line-ending settings, see Tech Note 63 on the Perforce web site:

<http://www.perforce.com/perforce/technotes/note063.html>

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

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

- Files with the `modtime (+m)` type are primarily intended for use by developers who need to preserve original timestamps on files. The use of `+m` in a file type overrides the client's `modtime` or `nomodtime` setting. For a more complete discussion of the `+m` modifier, see the *File Types* section.
- If you are using multiple or alternate client roots (the `AltRoots:` field), you can always tell which client root is in effect by looking at the `Client root:` reported by `p4 info`.
- To specify a Perforce client on Windows that spans multiple drives, use a `Root:` of `null`, and specify the drive letters in the client workspace view. For instance, the following client spec with a null client root maps `//depot/main/...` to an area of the `C:` drive, and other releases to the `D:` drive:

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

Use lowercase drive letters when specifying workspaces across multiple drives.

Examples

<code>p4 client</code>	Edit or create the client workspace specification named by the value of <code>P4CLIENT</code> or its equivalents.
<code>p4 client -t sue joe</code>	Create or edit client workspace <code>joe</code> , opening the form with the field values and workspace options in client workspace <code>sue</code> as defaults.
<code>p4 client -d release1</code>	Delete the client workspace <code>release1</code> .

Related Commands

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

p4 clients

Synopsis

List all client workspaces currently known to the system.

Syntax

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

Description

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

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

For example:

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

describes a client workspace named `paris`, last modified on February 19, 1999 with a root of `/usr/src`. The description of the workspace entered in the `p4 client` form is `Joe's client`.

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

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

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

Options

<code>-m <i>max</i></code>	List only the first <i>max</i> client workspaces.
<code>-u <i>user</i></code>	List only client workspaces owned by <i>user</i> .
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 client</code>
To see the name of the current client workspace and other useful data	<code>p4 info</code>
To view a list of Perforce users	<code>p4 users</code>

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|security ] value
```

Description

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

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

The Perforce server uses three counters in the course of its regular operations: `change`, `job`, and `journal`. Superusers 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 control server process monitoring by setting the `monitor` counter to 0 (disable monitoring), 1 (enable monitoring of active processes), or 2 (enable monitoring of both active and idle processes). You must stop and restart the Perforce server for any change in this counter to take effect. After you have enabled process monitoring, you can use `p4 monitor` to observe activity on the Perforce server.

To configure password strength requirements or to require the use of the ticket-based authentication mechanism, set the `security` counter to the desired level. You must stop and restart the Perforce server for this change to take effect. See the *System Administrator's Guide* for details.

Options

<code>-d countername</code>	Delete variable <code>countername</code> from the Perforce server.
<code>-f [change job journal]</code>	Force a change to one of three internal counters used by Perforce. Most installations rarely, if ever, need to use this flag.

<code>-f monitor [0 1 2]</code>	Server process monitoring off, monitor active processes only, or monitor both active and idle processes. See <code>p4 monitor</code> for details.
<code>-f security [0 1 2 3]</code>	Set the server security level. See the <i>System Administrator's Guide</i> for details.
<code>g-opts</code>	See the <i>Global Options</i> 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 <code>-f</code> 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 ints. (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)

Examples

<code>p4 counter mycounter 123</code>	Set the value of a counter <code>mycounter</code> to 123. If <code>mycounter</code> does not exist, it is created. Requires review access.
<code>p4 counter mycounter</code>	Display the value of <code>mycounter</code> . If <code>mycounter</code> does not exist, its value is displayed as 0. Requires list access.

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

<i>g-opts</i>	See the <i>Global Options</i> 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	<code>p4 counter</code>
--	-------------------------

p4 delete

Synopsis

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

Syntax

```
p4 [g-opts] delete [-c changelist#] [-n] 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

<code>-c change#</code>	Opens the files for <code>delete</code> within the specified changelist. If this flag is not provided, the files are linked to the default changelist.
<code>-n</code>	Preview which files would be opened for delete, without actually changing any files or metadata.
<code>g-opts</code>	See the <i>Global Options</i> 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

<code>p4 delete //depot/README</code>	Opens the file called <code>README</code> 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.
<code>p4 delete -c 40 file</code>	Opens <i>file</i> 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 <code>p4 submit</code> .

Related Commands

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

p4 depot

Synopsis

Create or edit a depot specification.

Syntax

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

Description

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

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

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

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

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

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

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

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

For more information about setting up a `spec` depot, see the *System Administrator's Guide*.

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

Form Fields

Field Name	Type	Description
Depot:	Read-Only	The depot name as provided in <code>p4 depot depotname</code> .
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	<code>local</code> , <code>remote</code> , or <code>spec</code> . Local depots are writable; remote depots are proxies for depots residing on other servers, and cannot be written to. The <code>spec</code> depot, if present, archives edited forms.
Address:	Writable	If the <code>Type:</code> is <code>remote</code> , the address should be the <code>P4PORT</code> address of the remote server. If the <code>Type:</code> is <code>local</code> or <code>spec</code> , this field is ignored.
Suffix:	Writable	If the <code>Type:</code> is <code>spec</code> , this field holds an optional suffix for generated paths to objects in the <code>spec</code> depot. The default suffix is <code>.p4s</code> . You do not need a suffix to use the <code>spec</code> depot, but supplying a file extension to your Perforce server's versioned specs enables users of GUI client software to associate Perforce specifications with a preferred text editor. If the <code>Type:</code> is <code>local</code> or <code>remote</code> , this field is ignored.
Map:	Writable	If the <code>Type:</code> is <code>local</code> or <code>spec</code> , set the map to point to the relative location of the depot subdirectory relative to the Perforce server's <code>P4ROOT</code> . The map must contain the <code>...</code> wildcard; for example, a local depot <code>new</code> might have a <code>Map:</code> of <code>new/...</code> . If the <code>Type:</code> is <code>remote</code> , set the map to point to a location in the remote depot's physical namespace, for example, <code>//depot/new/rel2/...</code> . This directory will be the root of the local representation of the remote depot.

Options

<code>-d depotname</code>	Delete the depot <i>depotname</i> . The depot must not contain any files; the Perforce superuser can remove files with <code>p4 obliterate</code> . If the depot is remote, <code>p4 obliterate</code> must still be run: no files are deleted, but any outstanding client or label records referring to that depot are eliminated.
<code>-i</code>	Read a depot specification from standard input.
<code>-o depotname</code>	Write a depot specification to standard output.
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 depots</code>
To populate a new depot with files	<code>p4 add</code>
To add mappings from an existing client workspace to the new depot	<code>p4 client</code>
To remove all traces of a file from a depot	<code>p4 obliterate</code>
To limit remote access to a depot	<code>p4 protect</code>

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

<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 depot</code>
To remove all traces of a file from a depot	<code>p4 obliterate</code>

p4 describe

Synopsis

Provides information about changelists and the changelists' files.

Syntax

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

Description

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

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

If a changelist is pending, it is flagged as such in the output, and the list of open files is shown. (Diffs for pending changelists are not displayed because the files have yet to be submitted to the server.)

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

<code>-s</code>	Display a shortened output that excludes the files' diffs.
<code>-dflag</code>	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the <i>Usage Notes</i> below for a flag listing.
<code>g-opts</code>	See the <i>Global Options</i> 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 <code>p4 describe -s</code>

The diff flags supported by `p4 describe` are:

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

Related Commands

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

p4 diff

Synopsis

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

Syntax

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

Description

`p4 diff` runs a diff program on the Perforce client, comparing files in the client workspace to revisions in the depot.

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

If the file argument includes wildcards, all open files that match the file pattern are diffed. If no file argument is provided, all open files are diffed against their depot counterparts.

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

Options

<code>-f</code>	Force the diff (if no revision is specified, against the head revision), even when the client file is not open for <code>edit</code> .
<code>-dflags</code>	Pass flags to the underlying diff routine (see the <i>Usage Notes</i> below for details)
<code>-m max</code>	Limit output to diffs (or status) of only the first <i>max</i> files.
<code>-sa</code>	Show only the names of opened files that are different from the revision in the depot, or are missing.
<code>-sb</code>	Show only the names of files opened for integrate that have been resolved, but that have been modified after being resolved.
<code>-sd</code>	Show only the names of unopened files that are missing from the client workspace, but present in the depot.
<code>-se</code>	Show only the names of unopened files in the client workspace that are different than the revision in the depot.

<code>-sr</code>	Show only the names of opened files in the client workspace that are identical to the revision in the depot.
<code>-sl file...</code>	Every unopened <i>file</i> is compared with the depot, and listed with a status of <i>same</i> , <i>diff</i> , or <i>missing</i> . If you use the <code>-f</code> flag together with the <code>-sl</code> flag, files that are open for edit are also compared and their status is listed.
<code>-t</code>	Diff the revisions even if the files are not of type <code>text</code> .
<code>g-opts</code>	See the <i>Global Options</i> 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
<code>-dn</code>	RCS output format, showing additions and deletions made to the file and associated line ranges.
<code>-dc</code>	context output format, showing line number ranges and three lines of context around the changes.
<code>-ds</code>	summary output format, showing only the number of chunks and lines added, deleted, or changed.
<code>-du</code>	unified output format, showing added and deleted lines with sufficient context for compatibility with the <code>patch(1)</code> utility.
<code>-dl</code>	ignore line-ending (CR/LF) convention when finding diffs
<code>-db</code>	ignore changes made within whitespace; this flag implies <code>-dl</code> .
<code>-dw</code>	ignore whitespace altogether; this flag implies <code>-dl</code> .

- 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

<code>p4 diff file#5</code>	Compare the client workspace revision of file <code>file</code> to the fifth depot revision.
<code>p4 diff @1999/05/22</code>	Compare all open files in the client workspace to the revisions in the depot as of midnight on May 22, 1999.
<code>p4 diff -du file</code>	Run the comparison on file <code>file</code> , displaying output in a format suitable for the <code>patch(1)</code> utility.
<code>p4 diff -sr p4 -x - revert</code>	<p>Revert all open, unchanged files.</p> <p>This differs from <code>p4 revert -a</code> (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.</p> <p>The first command shows all open, unchanged files. The second command (running <code>p4 -x</code> and taking arguments, one per line, from standard input, abbreviated as “-”) reverts each file in that list.</p> <p>(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).</p> <p>For more information about the <code>-x</code> option to <code>p4</code>, see the <i>Global Options</i> section.</p>

Related Commands

To compare two depot revisions	<code>p4 diff2</code>
To view the entire contents of a file	<code>p4 print</code>

p4 diff2

Synopsis

Compare two depot file revisions.

Syntax

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

Description

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

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

You can specify file patterns as arguments in place of specific files, with or without revision specifiers; this causes Perforce to perform multiple diffs for each pair of files that match the given pattern. If you invoke `p4 diff2` with file patterns, escape the file patterns from the OS shell by using quotes or backslashes, and be sure that the wildcards in the two file patterns match.

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

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

The possible values and meanings of *summary* are:

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

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

Options

<code>-q</code>	Quiet diff. Display only the header, and don't even display that when the file revisions' contents and types are identical.
<code>-dflags</code>	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the <i>Usage Notes</i> below for a listing of these flags.
<code>-b branchname fromfile[rev] tofile[rev]</code>	Use a branch specification to diff files in two branched codelines. The files that are compared can be limited by file patterns in either <i>fromfile</i> or <i>tofile</i> .
<code>-t</code>	Diff the file revisions even if the file(s) are not of type text.
<code>-u</code>	Generate unified output format, showing added and deleted lines with sufficient context for compatibility with the <code>patch(1)</code> utility. Only those files that differ are included. File names and dates remain in Perforce syntax.
<code>g-opts</code>	See the <i>Global Options</i> 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
<code>-dn</code>	RCS output format, showing additions and deletions made to the file and associated line ranges.
<code>-dc</code>	context output format, showing line number ranges and three lines of context around the changes.
<code>-ds</code>	summary output format, showing only the number of chunks and lines added, deleted, or changed.
<code>-du</code>	unified output format, showing added and deleted lines with sufficient context for compatibility with the <code>patch(1)</code> utility.

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

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

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

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

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

is printed if they are not identical.
- The option `-b branch [[fromfile[rev]] tofile[rev]]` may seem incorrect at first. Since the branch 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	<code>p4 diff</code>
To view the entire contents of a file	<code>p4 print</code>

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

<code>-C</code>	Display only those directories that are mapped through the current client workspace view.
<code>-D</code>	Include subdirectories that contain only deleted files. By default, these directories are not displayed.
<code>-H</code>	Include only those directories that contain files on the current client workspace's <code>p4 have</code> list.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	<code>list</code>

- 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 “...” 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#] [-n] [-t type] file...
```

Description

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

Perforce controls the local OS file permissions; when `p4 edit` is run, the OS write permission is turned on for the specified files.

When a file that has been opened for edit with `p4 edit` is submitted to the depot, the file revision that exists in the depot is not replaced. Instead, the new file revision is assigned the next revision number in sequence, and previous revisions are still accessible. By default, the newest revision (the *head revision*) is used by all commands that refer to the file.

By default, the specified files are added to the default changelist. Use `-c` to specify a different changelist.

To move files already opened for edit from one changelist to another, use `p4 reopen`.

Options

<code>-c change#</code>	Opens the files for edit within the specified changelist. If this flag is not provided, the files are linked to the default changelist.
<code>-t type</code>	Stores the new file revision as the specified type, overriding the file type of the previous revision of the same file. See the <i>File Types</i> section for a list of file types.
<code>-n</code>	Preview which files would be opened for edit, without actually changing any files or metadata.
<code>g-opts</code>	See the <i>Global Options</i> 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 must 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

<code>p4 edit -t text+k doc/*.txt</code>	Opens all files ending in <code>.txt</code> within the current directory's <code>doc</code> subdirectory for edit. These files are linked to the default changelist; these files are stored as type <code>text</code> with keyword expansion.
<code>p4 edit -t +l //depotname/...</code>	Implements pessimistic locking (exclusive-open) for all files in a depot. After this changelist is submitted, only one user at a time will be able to edit files in the depot named <code>depotname</code> .

```
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 250.

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 -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, import, or integrate;

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

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

See p4 integrated for a full description of integration actions.

- *date* is the submission date (by default), or date and time (if the `-t` flag is used).
- *user* is the name of the user who submitted the revision;
- *client* is the name of the client workspace from which the revision was submitted;
- *type* is the *type* of the file at the given revision; and
- *description* is the first 30 characters of the corresponding changelist's description.

If the `-l` option is used, the *description* is the full changelist description as entered when the changelist was submitted. If the `-L` option is used, the description is the full changelist description, truncated to 250 characters.

Options

<code>-i</code>	Follow file history across branches. If a file was created by integration via <code>p4 integrate</code> , 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).
<code>-l</code>	List long output, with the full text of each changelist description.
<code>-L</code>	List long output, with the full text of each changelist description truncated at 250 characters.
<code>-t</code>	Display the time as well as the date.
<code>-m maxrev</code>	List only the first <i>maxrev</i> changes per file output.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- Because `p4 filelog`'s output can be quite large when called with highly non-restrictive file arguments (for example, `p4 filelog //depot/...` displays the revision history for every file in the depot), `p4 filelog` commands may be subject to a `maxresults` limitation as set in `p4 group`.
- If both the `-i` and the `-m maxrev` flags are used, and a branch is encountered within the most recent *maxrev* revisions of the file, the most recent *maxrev* revisions of the file prior to the branch point are also displayed. `p4 filelog -i` follows branches down to a depth of 50 levels, which should be more than sufficient for any site.
- Old revisions of temporary object files (file type modifier `+Sn`) are displayed with an action of `purge`.

Examples

<code>p4 filelog //depot/proj1/...</code>	Display the revision history for every file under the depot's <code>proj1</code> directory.
<code>p4 filelog file1.c file1.h</code>	Show the revision history for files <code>file1.c</code> and <code>file1.h</code> , which reside locally in the current working directory.

Related Commands

To read additional information about each file	<code>p4 files</code>
To display file information in a format suitable for scripts	<code>p4 fstat</code>
To view a list of open files	<code>p4 opened</code>
To view a list of files you've synced to your client workspace	<code>p4 have</code>

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

<code>-a</code>	For each file, list all revisions within a specified revision range, rather than only the highest revision in the range.
<code>g-opts</code>	See the <i>Global Options</i> 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

<code>p4 files //depot/...</code>	Provides information about all files in the depot.
<code>p4 files //clientname/...</code>	Provides information about all depot files visible through the client view.
<code>p4 files @2000/12/10</code>	Provides information about all depot file revisions that existed on December 10, 2000.
<code>p4 files @2001/03/31:08:00,@2001/03/31:17:00</code>	Lists all files and revisions changed during business hours on March 31, 2001.
<code>p4 files //depot/proj2/...@p2lab</code>	Lists files and revisions under the directory <code>//depot/proj2/...</code> that are included in label <code>p2lab</code> .
<code>p4 files //depot/file.c</code>	Show information on the head revision of <code>//depot/file.c</code> . (that is, the <i>highest</i> revision in the implied range of <code>#1, #head</code>)
<code>p4 files -a //depot/file.c</code>	Show information on every revision of <code>//depot/file.c</code> (that is, <i>all</i> revisions in the implied range of <code>#1, #head</code>)

Related Commands

To list the revision history of files	<code>p4 filelog</code>
To see a list of all currently opened files	<code>p4 opened</code>
To see a list of the file revisions you've synced to	<code>p4 have</code>
To view the contents of depot files	<code>p4 print</code>

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

<code>-d</code>	Delete the fix record for the specified job at the specified changelist. The job's status will not change.
<code>-s status</code>	<p>Upon submission of the changelist, change the job's status to <i>status</i>, rather than the default value <code>closed</code>.</p> <p>If the changelist to which you're linking the job been submitted, the status value is immediately reflected in the job's status.</p> <p>If the changelist is <code>pending</code>, the job status is changed on submission of the changelist, provided that the <code>-s</code> flag is also supplied to <code>p4 submit</code> and the desired status appears next to the job in the <code>p4 submit</code> form's <code>Jobs:</code> field.</p>
<code>g-opts</code>	See the <i>Global Options</i> 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 suspended job002433
```

Mark job002433 as suspended, rather than closed, when changelist 201 is submitted.

Requires use of the -s flag with p4 submit.

Related Commands

To add or delete a job from a pending changelist	p4 change
To add or delete a job from the default changelist	p4 submit
To view a list of connections between jobs and changelists	p4 fixes
To create or edit a job	p4 job
To list all jobs, or a subset of jobs	p4 jobs
To change the format of jobs at your site (<i>superuser only</i>)	p4 jobspec
To read information about the format of jobs at your site	p4 jobspec -o

p4 fixes

Synopsis

List jobs and the changelists that fix them.

Syntax

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

Description

After a job has been linked to a particular numbered changelist with `p4 fix`, `p4 change`, or `p4 submit`, the job is said to have been *fixed* by the changelist (even if the changelist is still pending). The `p4 fixes` command lists changelists and the jobs they fix.

If invoked without arguments, `p4 fixes` displays all fix records. Fix records are displayed in the following format:

```
jobname fixed by change changelist# on date by user
```

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

- Use the `-c changelist` option to list only the jobs fixed by that pending or submitted changelist.
- Use the `-j job` option to list only those pending or submitted changelists that fix that job.
- Provide one or more file pattern arguments. If you provide a file argument, only submitted changelists affecting files that match the file patterns are listed; pending changelists are not included. If a revision specifier or revision range is included, only submitted changelists that affected files at the given revisions are listed. You can use the `-i` flag with a file pattern argument to include fixes made by changelists that were integrated into the specified files.
- Use the `-m max` flag to limit the output to the first `max` fixes.

Options

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

Usage Notes

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

Examples

<code>p4 fixes //depot/proj1/...</code>	Display all fixes made by submitted changelists that included any files under //depot/proj1.
<code>p4 fixes -c 414</code>	Display all jobs fixed by pending or submitted changelist 414.

Related Commands

To create or edit an existing job	<code>p4 job</code>
To list all jobs known to the system	<code>p4 jobs</code>
To attach a job to a particular changelist; the job is fixed by that changelist	<code>p4 fix</code>
To change the format of jobs at your site (<i>superuser only</i>)	<code>p4 jobspec</code>
To read information about the format of jobs at your site	<code>p4 jobspec -o</code>

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 than 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

<code>-n</code>	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.
<code>g-opts</code>	See the <i>Global Options</i> 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 than 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

<code>p4 flush</code> is an alias for <code>p4 sync -k</code>	<code>p4 sync -k</code>
To copy files from the depot to the client workspace	<code>p4 sync</code>
To bring the client workspace in sync with the have list after a bad <code>p4 flush</code>	<code>p4 sync -f</code>

p4 fstat

Synopsis

Dump file info in format suitable for parsing by scripts.

Syntax

```
p4 [g-opts] fstat [-m max] [-c|-e changelist#] [-Oflags -Rflags] file[rev]...
```

Description

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

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

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 <code>-Op</code> 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... <i>n</i>
headRev	head revision number, if in depot	1, 2, 3... <i>n</i>
headTime	Head revision changelist time, if in depot. Time is measured in seconds since 00:00:00 UTC, January 1, 1970	919283152 is a date in early 1999

Field Name	Description	Example/Notes
headRevModTime	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 <i>File Types</i> .)
haveRev	revision last synced to workspace, if on workspace	1, 2, 3... <i>n</i>
desc	changelist description (if using <i>-e changelist</i> and if the file was part of <i>changelist</i>)	A Perforce changelist
digest	MD5 digest of a file (requires <i>-Ol</i> option)	A 32 hexadecimal digit string
fileSize	file length in bytes (requires <i>-Ol</i> option)	63488
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... <i>n</i>
resolved	the number, if any, of resolved integration records	1, 2, 3... <i>n</i>
unresolved	the number, if any, of unresolved integration records	1, 2, 3... <i>n</i>

Field Name	Description	Example/Notes
otherOpen	the number of other users who have the file open, blank if no other users have the file open	1, 2, 3... <i>n</i> , preceded by <i>n</i> records listing the users (0 through <i>n</i> -1) with otherOpen <i>n</i> , otherAction <i>n</i> , and otherLock <i>n</i> fields as applicable. For example: ... otherOpen 3 otherOpen0 user1@cws1 otherOpen1 user2@cws2 otherOpen2 user3@cws3
otherOpen <i>n</i>	for each user with the file open, the workspace and user with the open file	user123@workstation9
otherLock	present and set to null if another user has the file locked, otherwise not present	unset (... otherLock) or not present
otherLock <i>n</i>	for each user with the file locked, the workspace and user holding the lock	user123@workstation9 Because only one user at a time, may lock a file, if <i>n</i> is set, <i>n</i> is always 0.
otherAction <i>n</i>	for each user with the file open, the action taken	one of add, edit, delete, branch, or integrate
otherChang <i>n</i>	for every changelist with the file open, the changelist	A changelist number
ourLock	present and set to null if the current user has the file locked, otherwise not present	unset (... ourLock) or not present
resolveAction <i>n</i> resolveBaseFile <i>n</i> resolveFromFile <i>n</i> resolveStartFromRev <i>n</i> resolveEndFromRev <i>n</i>	Pending integration action, base file, base revision number, from file, starting, and ending revision, respectively.	For pending integration record information, use the -Or option.

Options

<code>-c changelist#</code>	Display only files affected after the given changelist number. This operation is much faster than using a revision range on the affected files.
<code>-e changelist#</code>	Display only files affected by the given changelist number. This option is much faster than using a revision range on the affected files.
<code>-m max</code>	Produce fstat output for only the first <i>max</i> files.
<code>-Of</code>	Output all revisions for the given files, suppressing the <code>other[...]</code> and <code>resolve[...]</code> fields.
<code>-Ol</code>	Output a <code>fileSize</code> field displaying the length of the file and a <code>digest</code> field for each revision. On servers older than release 2005.1, this field may be expensive to compute, particularly for text files with many revisions.
<code>-Op</code>	Display the <code>clientFile</code> in Perforce syntax, as opposed to local syntax.
<code>-Or</code>	Display pending integration record data for files open in the current workspace.
<code>-Os</code>	Shorten output by excluding client workspace data (for instance, the <code>clientFile</code> field).
<code>-Rc</code>	Limit output to files mapped into the current workspace.
<code>-Rh</code>	Limit output to files on your have list; that is, to files synced to the current workspace.
<code>-Rn</code>	Limit output to files opened at revisions not at the head revision.
<code>-Ro</code>	Limit output to open files in the current workspace.
<code>-Rr</code>	Limit output to open files that have been resolved.
<code>-Ru</code>	Limit output to open files that are unresolved.
<code>g-opts</code>	See the <i>Global Options</i> section. The <code>-s</code> global option (which prefixes each line of output with a tag describing the type of output as <code>error</code> , <code>warning</code> , <code>info</code> , <code>text</code> , or <code>exit</code>) can be particularly useful when used with <code>p4 fstat</code> .

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.
- The `size` and `digest` fields are based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.

Examples

<code>p4 fstat file.c</code>	Displays information on <code>file.c</code>
<code>p4 fstat -Rc 20 *.c</code>	Displays information on all <code>.c</code> files affected after the checking-in of files under changelist 20.
<code>p4 fstat -Os file.c</code>	No client workspace information lines (i.e. <code>clientFile</code>) are displayed
<code>p4 fstat -Osl file.c</code>	No client workspace information lines are displayed, but the <code>fileSize</code> and <code>digest</code> lines are displayed.
<code>p4 fstat -Os -Ol file.c</code>	Equivalent to <code>p4 fstat -Osl</code> .

Related Commands

To read additional information about each file	<code>p4 files</code>
To display file information including change descriptions	<code>p4 filelog</code>

p4 group

Synopsis

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

Syntax

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

Description

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

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

Form Fields

Field Name	Type	Description
Group:	Read-only	The name of the group, as entered on the command line.
MaxResults:	Writable	The maximum number of results that members of this group can access from the server from a single command. The default value is <i>unlimited</i> . See the <i>Usage Notes</i> 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 <i>unlimited</i> . See the <i>Usage Notes</i> below for more details.
MaxLockTime	Writable	The maximum length of time (in milliseconds) that any one operation can lock any database table when scanning data. The default value is <i>unlimited</i> . See the <i>Usage Notes</i> below for more details.

Field Name	Type	Description
Timeout :	Writable	The duration (in seconds) of the validity of a session ticket created by <code>p4 login</code> . The default value is 43200 seconds (12 hours). To create a ticket that does not expire, set the <code>Timeout :</code> field to 0.
Users :	Writable, multi-line	The Perforce usernames of the group members. Each user name must be typed on its own line, and should be indented.
Subgroups :	Writable, multi-line	<p>Names of other Perforce groups.</p> <p>To add all users in a previously defined group to the group you're presently working with, include the group name in the <code>Subgroups :</code> field of the <code>p4 group</code> form. Note that user and group names occupy separate namespaces, and thus, groups and users can have the same names.</p> <p>Every member of any previously defined group you list in the <code>Subgroups :</code> field will be a member of the group you're now defining.</p>
Owners :	Writable, multi-line	<p>Names of other Perforce users.</p> <p>Group owners without <code>super</code> access are permitted to administer this group, provided that they use the <code>-a</code> flag.</p>

Options

<code>-a</code>	Allow a (non-superuser) group owner to administer the group. The user must be listed in the <code>Owner :</code> field of the group.
<code>-d groupname</code>	Delete group <i>groupname</i> . The members of the group are affected only if their access level or <code>maxresults</code> value changes as a result of the group's deletion.
<code>-i</code>	Read the form from standard input without invoking the user's editor. The new group specification replaces the previous one.
<code>-o</code>	Write the form to standard output without invoking the user's editor.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- Ticket `timeout` values for users who belong to multiple groups are calculated the same way as `maxresults` values: the largest `timeout` value for all the groups of which the user is a member. Users in no groups have the default `timeout` value of 43200. To create a ticket that does not expire, set the `timeout` to 0.
- As the number of files in the depot grows, certain commands can significantly slow down the server if called with no parameters, or if called with non-restrictive arguments. For example, `p4 print //depot/...` will print the contents of every file in the depot on the user's screen, and `p4 filelog //depot/...` will attempt to retrieve data on every file in the depot at *every revision*.

The Perforce superuser can limit the amount of data that the server returns to the client by setting the `MaxResults` value for groups of users. The superuser can also limit the amount of data scanned by the server (whether returned to the client or not) by setting the `MaxScanRows` value, and the length of time any database table can be locked in by any single operation by setting the `MaxLockTime` value.

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

If a user belongs to multiple groups, the server computes her `MaxResults` value to be the maximum of the `MaxResults` for all the groups of which the user is a member (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` and `MaxLockTime` limits are computed in the same way.)

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

- To display a group's `maxresults`, `maxscanrows`, `maxlocktime`, and `timeout` limits, use `p4 groups -v groupname`.
- Use `p4 help maxresults` to obtain the list of commands that are affected by any of the three limiting values.

Related Commands

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

p4 groups

Synopsis

List groups of users.

Syntax

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

Description

Shows a list of all current groups of users as created by `p4 group`. Only the group names are displayed.

If the optional *user* argument is provided, only the groups containing that user are listed. If the optional *group* argument is provided, only groups containing the named group are listed.

Use the `-i` option to include groups to which the user (or group) belongs by means of being a member of a subgroup. If a group argument is given, only groups that contain the named group are displayed.

Use the `-v` option to display the `MaxResults`, `MaxScanRows`, `MaxLockTime`, and `Timeout` values for the named group, or, if no group is specified, for all groups.

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

Options

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

Usage Notes

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

- To see all the members of a particular group, use `p4 group -o groupname`. This variation of `p4 group` requires only `list` access.
- The `-v` and `-i` options are mutually exclusive.

Examples

<code>p4 groups bob</code>	Display the names of all groups of which user bob is a member.
----------------------------	--

Related Commands

To create or edit an existing group of users	<code>p4 group</code>
To view a list of all the members and specifications of a particular group	<code>p4 group -o groupname</code>
To set Perforce access levels for the members of a particular group	<code>p4 protect</code>

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

<i>g-opts</i>	See the <i>Global Options</i> 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
<code>p4 help simple</code>	Provides short descriptions of the eight most basic Perforce commands.	(none)
<code>p4 help commands</code>	Lists all the Perforce commands	<i>Table of Contents</i>
<code>p4 help charset</code>	Describes how to control Unicode translation	P4CHARSET description.
<code>p4 help environment</code>	Lists the Perforce environment variables and their meanings	<i>Environment and Registry Variables</i>
<code>p4 help filetypes</code>	Lists the Perforce filetypes and their meanings	<i>File Types</i>
<code>p4 help jobview</code>	Describes Perforce jobviews	<code>p4 jobs</code> description
<code>p4 help revisions</code>	Describes Perforce revision specifiers	<i>File Specification</i>
<code>p4 help usage</code>	Lists the six flags available with all Perforce commands	<i>Global Options</i>
<code>p4 help views</code>	Describes the meaning of Perforce views	<i>Views</i>

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	<code>p4 info</code>
--	----------------------

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

<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 help</code>
To view version information for your Perforce client program	<code>p4 -v</code>

p4 integrate

Synopsis

Open files for branching or merging.

Syntax

```
p4 [g-opts] integrate [options] fromFile[revRange] toFile
p4 [g-opts] integrate [options] -b branch [toFile[fromRevRange]...]
p4 [g-opts] integrate [options] -b branch -s fromFile[revRange] [toFile...]
options: -c changelist# -d -Dflag -f -h -i -I -o -n -r -t -v
```

Description

When you've made changes to a file that need to be propagated to another file, start the process with `p4 integrate`. The simplest form of this command is `p4 integrate fromFile toFile`; this lets the Perforce server know that changes in *fromFile* need to be propagated to *toFile*, and has the following effects:

- If *toFile* doesn't yet exist, *fromFile* is copied to *toFile*, then *toFile* is opened for branch in the client workspace.
- If *toFile* exists, and shares a common ancestor with *fromFile* as above, then *toFile* is opened for integrate. You can then use `p4 resolve` to propagate all of, portions of, or none of the changes in *fromFile* to *toFile*. The `p4 resolve` command uses *fromFile* as *theirs*, *toFile* as *yours*, and the common ancestor of *fromFile* as *base*.
- If both *toFile* and *fromFile* exist, but *toFile* shares no common ancestor with *fromFile*, the integration is rejected. Use the `-i` flag to force a baseless merge.
- If *fromFile* was deleted at its last revision (and all previous changes have already been integrated between *fromFile* and *toFile*), *toFile* is opened for delete in the client workspace.

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

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

To specify multiple files, use wildcards in *fromFile* and *toFile*. Any wildcards used in *fromFile* must match identical wildcards in *toFile*. Perforce compares the *fromFile* pattern to the *toFile* pattern, creates a list of *fromFile/toFile* pairs, and performs an integration on each pair.

The syntax `p4 integrate fromFiles toFiles` requires you to specify the mapping between *fromFiles* and *toFiles* each time changes need to be propagated from *fromFiles* to *toFiles*. Alternatively, use `p4 branch` to store the mappings between *fromFiles* and *toFiles* in a *branch view*, and then use `p4 integrate -b branchview` whenever you need to propagate changes between *fromFiles* and *toFiles*.

Options

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

Basic Integration Flags

<code>-b branchname [toFiles...]</code>	Integrate the files using the <i>sourceFile/targetFile</i> mappings included in the branch view of <i>branchname</i> . If the <i>toFiles</i> argument is included, include only those target files in the branch view that match the pattern specified by <i>toFiles</i> .
<code>-n</code>	Display the integrations this command would perform without actually performing them.
<code>-v</code>	Open files for branching without copying <i>toFiles</i> into the client workspace. Without this flag, <code>p4 integrate</code> copies newly-branched <i>toFiles</i> into the client workspace from <i>fromFiles</i> . When the <code>-v</code> (<i>virtual</i>) flag is used, Perforce won't copy <i>toFiles</i> to the client workspace. Instead, you can fetch them with <code>p4 sync</code> when you need them.
<code>-c changelist#</code>	Open the <i>toFiles</i> for branch, integrate, or delete in the specified pending changelist. If this option is not provided, the files are opened in the default changelist.
<code>g-opts</code>	See the <i>Global Options</i> 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`

The `-d` flag enables integrations around deleted revisions. Using `-d` implies `-Dt`, `-Ds`, and `-Di`.

If the target file has been deleted and the source file has changed, (or the source file has changed and subsequently been deleted), using `-d` re-branches the changes from the source file on top of the target file.

If the source file has been deleted and the target file has changed, using `-d` deletes the target file.

If the source file has been deleted and re-added, using `-d` integrates all outstanding revisions of the file, including those revisions prior to the file's deletion.

If you do not use the `-d` flag, outstanding edits may not be mixed with a deleted file.

-Dt	The -Dt flag allows integration around a deleted target file; the source file is branched onto the deleted target.
-Ds	The -Ds flag allows integration around a deleted source file; if the source file has been deleted, any modified target file is also deleted.
-Di	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 <i>fromFile</i> and <i>toFile</i> , 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 <i>toFile</i> and <i>fromFile</i> share no common ancestor, using the first revision as the <i>base</i> .
-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

<code>p4 integ //depot/dev/... //depot/rel2/...</code>	Branch or merge all files in <code>//depot/dev/...</code> to the corresponding files in <code>//depot/rel2/...</code> If there is no corresponding file in <code>//depot/rel2/...</code> , this creates it.
<code>p4 integ -b rel2br</code>	Branch or merge all <i>fromFiles</i> contained in the branch view <code>rel2br</code> into the corresponding <i>toFiles</i> as mapped through the branch view.
<code>p4 integ -b rel2br //depot/rel2/headers/...</code>	Branch or merge those <i>fromFiles</i> contained in the branch view <code>rel2br</code> that map to the <i>toFiles</i> <code>//depot/rel2/headers/...</code>
<code>p4 integ -b rel2br -r //depot/rel2/README</code>	Branch or merge <i>fromFile</i> <code>//depot/rel2/README</code> from its <i>toFile</i> as mapped through the branch view <code>rel2br</code> .

Related Commands

To create or edit a branch specification	<code>p4 branch</code>
To view a list of existing branch specifications	<code>p4 branches</code>
To view a list of integrations that have already been performed and submitted	<code>p4 integrated</code>

To propagate changes from one file to another after opening files with `p4 integrate`

`p4 resolve`

To view a history of all integrations performed on a particular file

`p4 filelog`

p4 integrated

Synopsis

Show integrations that have been submitted.

Syntax

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

Description

The `p4 integrated` command shows the integration history of the selected files, in the format:

```
file#revision-range - integrate-action partner-file#revision-range
```

where

- *file* is the file argument provided to `p4 integrated`;
- *partner-file* is the file it was integrated from or into; and
- *integrate-action* describes what the user did during the `p4 resolve` process, and is one of the following:

Integrate Action	What the User Did During the <code>p4 Resolve</code> Process
branch from	<i>file</i> did not previously exist; it was created as a copy of <i>partner-file</i> .
branch into	<i>partner-file</i> did not previously exist; it was created as a copy of <i>file</i> .
merge from	<i>file</i> was integrated from <i>partner-file</i> , accepting <i>merge</i> .
merge into	<i>file</i> was integrated into <i>partner-file</i> , accepting <i>merge</i> .
copy from	<i>file</i> was integrated from <i>partner-file</i> , accepting <i>theirs</i> .
copy into	<i>file</i> was integrated into <i>partner-file</i> , accepting <i>theirs</i> .
ignored	<i>file</i> was integrated from <i>partner-file</i> , accepting <i>yours</i> .
ignored by	<i>file</i> was integrated into <i>partner-file</i> , accepting <i>yours</i> .
delete from	<i>file</i> was integrated from <i>partner-file</i> , and <i>partner-file</i> had been previously deleted.
delete into	<i>file</i> was integrated into <i>partner-file</i> , and <i>file</i> had been previously deleted.

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

If a file *toFile* was ever integrated from a file *fromFile*, and both *toFile* and *fromFile* match the p4 integrated *filepattern* argument, each integrated action is listed twice in the p4 integrated output: once in its *from* form, and once in its *into* form, as described above.

If the optional *-b branch* flag is used, only files integrated from the source to target files in the branch view are shown.

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

Options

<i>g-opts</i>	See the <i>Global Options</i> 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] <i>file</i>

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 `jobNNNNNN` 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 <code>new</code> . When the form is closed, this is replaced with the name <code>jobNNNNNN</code> , where <code>NNNNNN</code> 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 <code>open</code> , <code>closed</code> , or <code>suspended</code> . When the job is linked to a changelist, the value of this field is set to <code>closed</code> 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

<code>-d jobname</code>	Delete job <i>jobname</i> , but only if it has no associated pending or submitted fixes.
<code>-f</code>	Force flag. Allows Perforce administrators to edit read-only fields.
<code>-i</code>	Read the job form from standard input without invoking an editor.
<code>-o</code>	Write the job form to standard output without invoking an editor.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- 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

<code>p4 job</code>	Create a new job; by default, its name is of the form <code>jobNNNNNNN</code> .
<code>p4 job job000135</code>	Edit job <code>job000135</code> .

Related Commands

To list all jobs, or a subset of jobs	<code>p4 jobs</code>
To attach a job to an existing changelist	<code>p4 fix</code>
To view a list of connections between jobs and changelists	<code>p4 fixes</code>
To add or delete a job from a pending changelist	<code>p4 change</code>
To change the format of jobs at your site (superuser only)	<code>p4 jobspec</code>
To read information about the format of jobs at your site	<code>p4 jobspec -o</code>

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

<code>-e jobview</code>	List only those jobs that match the criteria specified by <i>jobview</i> . Please see the <i>Usage Notes</i> below for a discussion of job views.
<code>-i files...</code>	Include jobs fixed by changelists that affect files integrated into the named files.
<code>-l</code>	Output the full description of each job.
<code>-m max</code>	Include only the first <i>max</i> jobs, sorted alphanumerically. If used with the <code>-r</code> flag, the last <i>max</i> jobs are included.
<code>-r</code>	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.
<i>g-opts</i>	See the <i>Global Options</i> 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.

Amperstands (`&`) 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.
text	<p>The relational operators perform comparisons in ASCII order.</p> <p>The equality operator (=) matches the job if the word given as the value is found anywhere in the specified field.</p> <p>The relational operators are of limited use here, since they match the job if <i>any</i> word in the specified field matches the provided value.</p> <p>For example, if a job has a text field <code>ShortDescription</code> that contains only the phrase <code>gui bug</code>, and the jobview is "<code>ShortDesc<filter</code>", the job matches the jobview, because <code>bug<filter</code>.</p>
line	As for field type <code>text</code> , above.
select	The equality operator (=) matches a job if the value of the named field is the specified word. The relational operators perform comparisons in ASCII order.
date	Dates are matched chronologically. If a specific time is not provided, the operators <code>=</code> , <code><=</code> , and <code>>=</code> 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

<pre>p4 jobs //depot/proj/file#1</pre>	List all jobs attached to changelists that include revisions of <code>//depot/proj/file</code> .
<pre>p4 jobs -i //depot/proj/file</pre>	List all jobs attached to changelists that include revisions of <code>//depot/proj/file</code> or revisions of files that were integrated into <code>//depot/proj/file</code>
<pre>p4 jobs -e gui</pre>	List all jobs that contain the word <code>gui</code> in any field.
<pre>p4 jobs -e "gui Submitted-By=joe"</pre>	List all jobs that contain the word <code>gui</code> in any field and the word <code>joe</code> in the <code>Submitted-By</code> field.

<code>p4 jobs -e "gui ^Submitted-By=joe"</code>	List all jobs that contain the word <code>gui</code> in any field and any value <i>other than</i> <code>joe</code> in the <code>Submitted-By:</code> field.
<code>p4 jobs -e "window*"</code>	List all jobs containing the word <code>"window"</code> , <code>"window.c"</code> , <code>"Windows"</code> , in any field. The quotation marks are used to prevent the local shell from expanding the <code>"*"</code> on the command line.
<code>p4 jobs -e window.c</code>	List all jobs referring to <code>window.c</code> in any field.
<code>p4 jobs -e "job=* ^unwanted"</code>	List all jobs not containing the word <code>unwanted</code> in any field.
<code>p4 jobs -e "(fast quick) &date>1998/03/14"</code>	List all jobs that contain the word <code>fast</code> or <code>quick</code> in any field, and have a <code>date:</code> field pointing to a date on or after 3/14/98.
<code>p4 jobs -e "fast quick" //depot/proj/...</code>	List all jobs that have the word <code>fast</code> or <code>quick</code> in any field, and that are linked to changelists that affected files under <code>//depot/proj</code> .

Related Commands

To create or edit an existing job	<code>p4 job</code>
To attach a job to a particular changelist, indicating that the job is fixed by that changelist	<code>p4 fix</code>
To list all jobs and changelists that have been linked together	<code>p4 fixes</code>
To view all the information about a particular changelist, including the jobs linked to the changelist	<code>p4 describe</code>
To change the format of the jobs used on your server (superuser only)	<code>p4 jobspec</code>
To read information about the format of jobs used on your site (any user)	<code>p4 jobspec -o</code>
To set a default jobview that includes jobs matching the jobview in all new changelists	<code>p4 user</code>

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:	<p>A list of field definitions for your site's jobs, one field per line. Each line has five parts, and is of the form <i>code name type length persistence</i>.</p> <ul style="list-style-type: none">• 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 <i>Usage Notes</i> 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:<ul style="list-style-type: none">• 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)	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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 <code>\$now</code> and <code>\$user</code> variables; it allows you to change the date a job was modified and the name of the modifying user.
Values:	<p>Contains a lists of fields and valid values for <code>select</code> fields.</p> <p>Enter one line for each field of datatype <code>select</code>. Each line must contain the fieldname, a space, and the list of acceptable values separated by slashes. For example:</p> <pre>JobType bug/request/problem.</pre>
Presets:	<p>Contains a list of fields and their default values for each field that has a persistence of <code>default</code>, <code>required</code>, <code>once</code>, or <code>always</code>.</p> <p>Each line must contain the field name and the default value, separated by a space. For example:</p> <pre>JobType bug</pre> <p>Any one-line string can be used, or one of three built-in variables:</p> <ul style="list-style-type: none"> • <code>\$user</code>: the user who created the job • <code>\$now</code>: the current date • <code>\$blank</code>: the phrase <code><enter description here></code> <p>When users enter jobs, any fields in your jobspec with a preset of <code>\$blank</code> must be filled in by the user before the job is added to the system.</p>

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 <i>Usage Notes</i> 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 <i>Global Options</i> 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.

Labels can be either automatic or static. Automatic labels refer to the revisions provided in the `View:` and `Revision:` fields. Static labels refer only to those specific revisions tagged by the label by means of either the `p4 labelsync` or `p4 tag` commands.

Only the `Owner:` of an unlocked label 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, optional	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 <code>p4 labelsync</code> on the label, or by otherwise referring to a file with the label revision specifier <code>@label</code> .
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 <code>p4 labelsync</code> .
Revision:	Writable	An optional revision specification for an automatic label. If you use the <code>#</code> character to specify a revision number, you must use quotes around it in order to ensure that the <code>#</code> is parsed as a revision specifier, and not as a comment field in the form.
View:	Writable	A list of depot files that can be tagged with this label. No files are actually tagged until <code>p4 labelsync</code> 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 <i>Views</i> chapter for more information.

Options

<code>-d [-f]</code>	Delete the named label if it's unlocked. The <code>-f</code> flag forces the deletion even if the label is locked. (Deleting a locked label requires admin or super access.)
<code>-i</code>	Read the label definition from standard input without invoking the editor.
<code>-o</code>	Write the label definition to standard output without invoking the editor.
<code>-f</code>	Allow the <code>Update:</code> field's date to be set. Can be used with either the <code>-i</code> flag or the <code>-t</code> flag for the same purpose.
<code>-t template</code>	Copy label <i>template</i> 's view and options into the <code>View:</code> and <code>Options:</code> fields of this label.
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 labelsync</code>
To list all labels known to the system	<code>p4 labels</code>
To create a label and tag files with the label	<code>p4 tag</code>

p4 labels

Synopsis

Display list of defined labels.

Syntax

```
p4 [g-opts] labels [ -u user ] [ -m max ] [ 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 static labels that tag specific files, specify a file pattern, with an optional revision range. (Because automatic labels refer to all files in the label view at a specified revision range, automatic labels are not shown when you use `p4 labels` with a file pattern.)

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

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

Options

<code>-m <i>max</i></code>	List only the first <i>max</i> labels.
<code>-u <i>user</i></code>	List only labels owned by <i>user</i> .
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- To see a list of files tagged by a particular label, use `p4 files @labelname`.

Examples

To list all labels in the system	<code>p4 labels</code>
To list all labels that contain any revision of <code>file.c</code>	<code>p4 labels file.c</code>
To list only labels containing revisions #3 through #5 of <code>file.c</code>	<code>p4 labels file.c#3,5</code>

Related Commands

To create a label and tag files with the label	<code>p4 tag</code>
To create or edit a label specification	<code>p4 label</code>
To add, delete, or change the files included in a label	<code>p4 labelsync</code>
To view a list of files included in a label	<code>p4 files @labelname</code>

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

<code>-a</code>	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.
<code>-d</code>	Delete the label tag from the named files.
<code>-l labelname</code>	Specify the label to be applied to file revisions
<code>-n</code>	Display what <code>p4 labelsync</code> would do without actually performing the operation.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- By default, `p4 labelsync` operates on the revisions of files last synced to your client workspace. To tag the head revisions of files (or the highest revision in a specified range), use `p4 tag`.

Related Commands

To create or edit a label	<code>p4 label</code>
To list all labels known to the system	<code>p4 labels</code>
To create a label and tag files with the label	<code>p4 tag</code>

p4 license

Synopsis

Update or display the license file.

Syntax

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

Description

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

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

Most new license files obtained from Perforce can be installed with `p4 license`, except for when the server IP address has changed. If the server IP address has changed, you must still stop the Perforce Server, manually copy the license file into place, and restart the Server.

Options

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

Usage Notes

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

Examples

<code>p4 license -o</code>	Display the current license file on the standard output.
<code>p4 license -i</code>	Read in a new license file from the standard input.

p4 lock

Synopsis

Lock an opened file against changelist submission.

Syntax

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

Description

Locking files prevents all other users from submitting changes to those files. If the files are already locked by another user, `p4 lock` fails. When the user who locked a particular file submits the file, the lock is released.

This command is normally called with a specific file argument; if no file argument is provided, all open files in the default changelist are locked. If the `-c changelist#` flag is used, all open files matching the given file pattern in changelist `changelist#` are locked.

Options

<code>-c changelist#</code>	Lock only files included in changelist <code>changelist#</code>
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 unlock</code>
To display all your open, locked files (UNIX)	<code>p4 opened grep "*locked*"</code>

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

<code>-c sequence#</code>	List all events happening after this sequence number.
<code>-t countername</code>	List all events after this counter number.
<code>-c sequence# -t countername</code>	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.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- The `p4 logger` command is not intended for use by end users. It exists to support propagation of information to an external defect tracking system.

Related Commands

To list users who have subscribed to review particular files	<code>p4 reviews</code>
To set or read the value of a Perforce counter	<code>p4 counter</code>
To see full information about a particular changelist	<code>p4 describe</code>
To see a list of all changelists, limited by particular criteria	<code>p4 changes</code>

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

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

Usage Notes

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

- The default timeout value of 43200 seconds (12 hours) is defined on a per-group basis in the `p4 group` form.
- To create tickets that do not expire, set the timeout value to 0 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.
- Tickets are stored in the file specified by the `P4TICKETS` environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.

Examples

<code>p4 login</code>	Prompt the user for a password; if the password is entered correctly, issue a ticket valid on the user's machine.
<code>p4 -u builder login -a</code>	Attempt to log in as user <code>builder</code> ; if the password is entered correctly, issue a ticket valid on all machines.

Related Commands

To end a login session	<code>p4 logout</code>
To display tickets	<code>p4 tickets</code>

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 <i>Global Options</i> section.

Usage Notes

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

- Tickets are stored in the file specified by the `P4TICKETS` environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.

Examples

p4 logout	Log out of Perforce by removing the local session ticket.
p4 logout -a	Log out of Perforce by 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. You can control server process monitoring by setting the `monitor` counter to 0 (disable monitoring), 1 (enable monitoring of active processes), or 2 (enable monitoring of both active and idle processes). You must stop and restart the Perforce server for any change in this counter to take effect.

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

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

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

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

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

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

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

Options

<code>g-opts</code>	See the <i>Global Options</i> section.
<code>-a</code>	Show all arguments associated with the process (for example, edit <code>file.c</code> , or <code>sync -f //depot/src/...</code>). Perforce user names are truncated to 10 characters, and each line is limited to a total of 80 characters of output.
<code>-e</code>	Show environment information including Perforce client application (if known), host IP address, and client workspace name.
<code>-l</code>	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, super

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

Examples

<code>p4 monitor show</code>	Show Perforce processes information (commands only). Requires <code>list</code> access only.
<code>p4 monitor show -l</code>	Show arguments and commands, without limits on line length. Requires <code>admin</code> access.
<code>p4 monitor show -a</code>	Show arguments and commands, limited to 80 characters per line of output. Requires <code>admin</code> access.

<code>p4 monitor terminate 123</code>	Instruct the Perforce server to mark process 123 for termination. Requires super access.
<code>p4 monitor clear all</code>	Clears the monitor table of all entries. Requires super access.

Related Commands

To turn on server monitoring (requires server restart)	<code>p4 counter -f monitor 1</code>
To turn off server monitoring (requires server restart)	<code>p4 counter -f monitor 0</code>

p4 obliterate

Synopsis

Removes files and their history from the depot.

Syntax

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

Options

<code>-y filespec</code>	Perform the obliterate operation. Without this flag, <code>p4 obliterate</code> merely reports what it would do.
<code>g-opts</code>	See the <i>Global Options</i> 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.

Examples

<code>p4 obliterate dir/...</code>	Do not obliterate any files; list the files that would be obliterated with the <code>-y</code> option. In this case, all files in directory <code>dir</code> and below would be subject to deletion with the <code>-y</code> option.
<code>p4 obliterate -y file</code>	Obliterate <code>file</code> from the depot. All history and metadata for every revision of <code>file</code> are erased.
<code>p4 obliterate -y file#3</code>	Obliterate only the third revision of <code>file</code> . If #3 <i>was</i> the head revision, the new head revision is now #2 and the next revision will be revision #3. If #3 <i>was not</i> the head revision, the head revision remains unchanged.
<code>p4 obliterate -y file#3,5</code>	Obliterate revisions 3, 4, and 5 of <code>file</code> . If #5 <i>was</i> the head revision, the new head revision is now #2, and the next revision will be #3. If #5 <i>was not</i> 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. `p4 delete`
This is the normal way of deleting files.

p4 opened

Synopsis

List files that are open in pending changelists.

Syntax

```
p4 [g-opts] opened [-a -c changelist# -C workspace -m max] [file ...]
```

Description

Use `p4 opened` to list files that are currently open via `p4 add`, `p4 edit`, `p4 delete`, or `p4 integrate`. By default, all open files in the current client workspace are listed. You can use command line arguments to list only those files in a particular pending changelist, or to show open files in all pending changelists, and to limit the number of files displayed.

If file specifications are provided as arguments to `p4 opened`, only those files that match the file specifications are included in the report.

The information displayed for each opened file includes the file's name, its location in the depot, the revision number that the file was last synced to, the number of the changelist under which the file was opened, the operation it is opened for (add, edit, delete, or integrate), and the type of the file. The output for each file looks like this:

```
depot-file#rev - action chnum change (type) [lock-status]
```

where:

- *depot-file* is the path in depot syntax;
- *rev* is the revision number;
- *action* is the operation the file was open for: add, edit, delete, branch, or integrate;
- *chnum* is the number of the submitting changelist; and
- *type* is the *type* of the file at the given revision.
- If the file is locked (see `p4 lock`), a warning that it is **locked** appears at the line's end.

Options

<code>-a</code>	List opened files in any client workspace.
<code>-c changelist#</code>	List the files in pending changelist <i>changelist#</i> . To list files in the default changelist, use <code>p4 opened -c default</code> .
<code>-C workspace</code>	List only files that are open in the specified client <i>workspace</i> .

<code>-m max</code>	List only the first <i>max</i> open files.
<code>g-opts</code>	See the <i>Global Options</i> 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

<code>p4 opened -c 35 //depot/main/...</code>	List all files in pending changelist 35 that lie under the depot's <code>main</code> subdirectory.
<code>p4 opened -a -c default</code>	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	<code>p4 add</code> <code>p4 edit</code> <code>p4 delete</code> <code>p4 integrate</code>
To move a file from one changelist to another	<code>p4 reopen</code>
To remove a file from all changelists, reverting it to its previous state	<code>p4 revert</code>
To create a new, numbered changelist	<code>p4 change</code>
To view a list of changelists that meet particular criteria	<code>p4 changes</code>

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 `P4PASSWORD` to the password value;
- Create a setting for `P4PASSWORD` 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

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

Usage Notes

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

- The `p4 passwd` command never sends plaintext passwords over the network; a challenge/response mechanism is used to send the encrypted password to the server.
- Passwords 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	<code>p4 user</code>
To change users' access levels	<code>p4 protect</code>
To log in using tickets instead of passwords	<code>p4 login</code>

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

<code>-a</code>	For each file, print all revisions within a specified revision range, rather than only the highest revision in the range.
<code>-q</code>	Suppress the one-line file header normally added by Perforce.
<code>-o outfile</code>	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.
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 diff2</code>
To compare the contents of an opened file in the client workspace to a depot file revision	<code>p4 diff</code>

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 <code>p4 files</code> , <code>p4 client</code> , <code>p4 job</code> , <code>p4 describe</code> , <code>p4 branch</code> , etc.
read	The user can do everything permitted with <code>list</code> access, and also run any command that involves reading file data, including <code>p4 print</code> , <code>p4 diff</code> , <code>p4 sync</code> , and so on.
open	This gives the user permission to do everything she can do with <code>read</code> access, and gives her permission to <code>p4 add</code> , <code>p4 edit</code> , and <code>p4 delete</code> 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 <code>p4 submit</code> and lock them with <code>p4 lock</code> .

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 <code>list</code> and <code>read</code> can do, and grants permission to run <code>p4 review</code> and <code>p4 counter</code> . It does not include <code>open</code> or <code>write</code> access.
admin	Includes all of the above, including administrative commands that override changes to metadata, but do not affect server operation. These include <code>p4 branch -f</code> , <code>p4 change -f</code> , <code>p4 client -f</code> , <code>p4 job -f</code> , <code>p4 jobspec</code> , <code>p4 label -f</code> , <code>p4 obliterate</code> , <code>p4 typemap</code> , <code>p4 unlock -f</code> , and <code>p4 verify</code> .
super	Includes all of the above, plus access to the superuser commands such as <code>p4 admin</code> , <code>p4 counter</code> , <code>p4 triggers</code> , <code>p4 protect</code> , 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 <code>list</code> , <code>read</code> , <code>open</code> , <code>write</code> , <code>review</code> , or <code>super</code> , as defined above.
User or Group	Does this protection apply to a user or a group? The value of this field must be <code>user</code> or <code>group</code> .
Group Name or User Name	The name of the user or the name of the group, as defined by <code>p4 group</code> . To grant this permission to all users, use the <code>*</code> wildcard.
Host	The IP address. CIDR notation is supported. The <code>*</code> wildcard can also be used to refer to all IP addresses, but only when you are not using CIDR notation.
Depot File Path	The depot file path this permission is granted on, in Perforce <i>depot syntax</i> . The file specification can contain Perforce <i>wildcards</i> . To exclude this mapping from the permission set, use a dash (<code>-</code>) as the first character of this value.

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

When exclusionary mappings are used, order is relevant: the exclusionary mapping overrides any matching protections listed above it in the table. No matter what access

level is being denied in the exclusionary protection, all the access levels for the matching users, files, and IP addresses are denied.

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

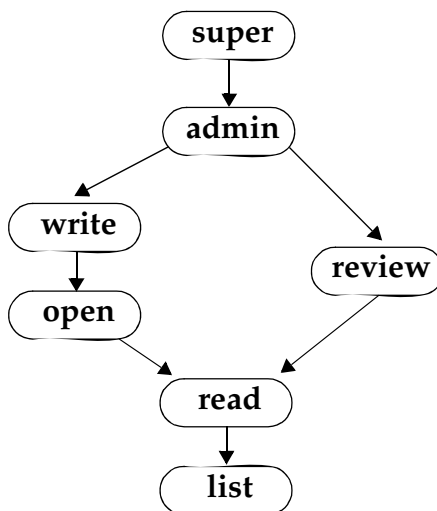
Options

<code>-i</code>	Read the form from standard input without invoking an editor.
<code>-o</code>	Write the form to standard output without invoking an editor.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	<code>super</code>

- 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	Notes
<code>add</code>	<code>open</code>	
<code>admin</code>	<code>super</code>	
<code>annotate</code>	<code>read</code>	
<code>branch</code>	<code>open</code>	The <code>-f</code> flag to override existing metadata or other users' data requires <code>admin</code> access.
<code>branches</code>	<code>list</code>	
<code>change</code>	<code>open</code>	The <code>-f</code> flag to override existing metadata or other users' data requires <code>admin</code> access.
<code>changes</code>	<code>list</code>	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
<code>client</code>	<code>list</code>	The <code>-f</code> flag to override existing metadata or other users' data requires <code>admin</code> access.
<code>clients</code>	<code>list</code>	
<code>counter</code>	<code>review</code>	<code>list</code> access is required to view an existing counter's value; <code>review</code> access is required to change a counter's value or create a new counter.
<code>counters</code>	<code>list</code>	
<code>delete</code>	<code>open</code>	
<code>depot</code>	<code>super</code>	The <code>-o</code> flag to this command, which allows the form to be read but not edited, requires only <code>list</code> access.
<code>depots</code>	<code>list</code>	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
<code>describe</code>	<code>read</code>	The <code>-s</code> flag to this command, which does not display file content, requires only <code>list</code> access.
<code>diff</code>	<code>read</code>	
<code>diff2</code>	<code>read</code>	
<code>dirs</code>	<code>list</code>	
<code>edit</code>	<code>open</code>	

Command	Access Level	Notes
filelog	list	
files	list	
fix	open	
fixes	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
fstat	list	
group	super	The -o flag to this command, which allows the form to be read but not edited, requires only <code>list</code> access. The -a flag to this command requires only <code>list</code> access, provided that the user is also listed as a group owner.
groups	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
have	list	
help	none	
info	none	
integrate	open	The user must have <code>open</code> access on the target files and <code>read</code> access on the source files.
integrated	list	
job	open	The -o flag to this command, which allows the form to be read but not edited, requires only <code>list</code> access. The -f flag to override existing metadata or other users' data requires <code>admin</code> access.
jobs	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
jobspec	admin	The -o flag to this command, which allows the form to be read but not edited, requires only <code>list</code> access.
label	open	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot. The -f flag to override existing metadata or other users' data requires <code>admin</code> access.

Command	Access Level	Notes
labels	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
labelsync	list	
license	super	
lock	write	
login	list	
logout	list	
monitor	list	super access is required to terminate or clear processes, admin access is required to view arguments.
obliterate	admin	
opened	list	
passwd	list	
print	read	
protect	super	
protects	list	super access is required to use the -a, -g, and -u flags.
reopen	open	
resolve	open	
resolved	open	
revert	open	
review	review	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
reviews	list	
set	list	
sizes	list	
submit	write	
sync	read	
tag	open	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
tickets	none	

Command	Access Level	Notes
triggers	super	
typemap	admin	The <code>-o</code> flag to this command, which allows the form to be read but not edited, requires only <code>list</code> access.
unlock	open	The <code>-f</code> flag to override existing metadata or other users' data requires <code>admin</code> access.
user	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
users	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
verify	admin	
where	none	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

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

- To limit or eliminate the use of the files on a particular server as a remote depot from 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, see the *Protections* chapter of the *System Administrator's Guide*.

Examples

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

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

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

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

Related Commands

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

p4 protects

Synopsis

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

Syntax

```
p4 [g-opts] protects [ -a | -u user | -g group | -h host ] [-m] [ file... ]
```

Description

Use the `p4 protects` command to display the lines from the protections table that apply to a user, group, or set of files.

With no options, `p4 protects` displays the lines in the protections table that apply to the current user. If a *file* argument is provided, only those lines in the protection table that apply to the named files are displayed.

Use the `-a` flag to display lines for all users, or `-u user`, `-g group`, or `-h host` flags to display lines for a specific user, group, or host IP address.

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

Options

<code>-a</code>	Displays protection lines for all users. This option requires <i>super</i> access.
<code>-g <i>group</i></code>	Displays protection lines that apply to the named group. This option requires <i>super</i> access.
<code>-h <i>host</i></code>	Displays protection lines that apply to the specified host IP address. This option requires <i>super</i> access.
<code>-u <i>user</i></code>	Displays protection lines that apply to the named user. This option requires <i>super</i> access.
<code>-m</code>	Display a one-word summary of the maximum applicable access level. (Note: this does not take into account exclusionary mappings.)
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	<i>list</i> , <i>super</i> for <code>-a</code> , <code>-h</code> , <code>-g</code> , <code>-u</code>

Related Commands

To edit the protections table

p4 protect

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
<i>fromFile</i> : Yes <i>toFile</i> : No	No	read access for <i>fromFile</i> write access for <i>toFile</i>

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	<code>p4 integrate</code>
To delete a file from the depot	<code>p4 delete</code>
To submit changes to the depot	<code>p4 submit</code>

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

<code>-c <i>changelist#</i> <i>file</i></code>	Move all open files matching file pattern <i>file</i> to pending changelist <i>changelist#</i> . To move a file to the default changelist, use <code>default</code> as the changelist number.
<code>-t <i>filetype</i> <i>file</i></code>	When submitted, store file as type <i>filetype</i> . All subsequent revisions will be of that file type until the type is changed again.
	See the <i>File Types</i> section for a list of file types.
<code><i>g-opts</i></code>	See the <i>Global Options</i> section.

Usage Notes

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

Examples

```
p4 reopen -t text+k //...
```

Reopen all open files as text files with keyword expansion.

```
p4 reopen -c 410  
//depot/proj1/... //.../README
```

Move all open files under directory //depot/proj1 or that are named README to pending changelist 410.

```
p4 reopen -c default -t binary+S //....exe
```

Move all open .exe files to the default changelist, overwriting older revisions of those files in the depot.

Related Commands

To submit a changelist to the depot

```
p4 submit
```

To create a new changelist

```
p4 change
```

To remove a file from all pending changelists

```
p4 revert
```

To list opened files

```
p4 opened
```

To list all the files included in a changelist

```
p4 opened -c changelist#
```

To list all pending changelists

```
p4 changes -p pending
```

To open a file for edit under a particular pending changelist and as a particular type

```
p4 edit -c changelist# -t type
```

To open a file for add under a particular pending changelist and as a particular type

```
p4 add -c changelist# -t type
```

To implement pessimistic locking (exclusive-open) for all files in a depot. After this changelist is submitted, only one user at a time will be able to edit files in the depot named *depotname*.

```
p4 edit -t +l //depotname/...
```

p4 resolve

Synopsis

Resolve conflicts between file revisions.

Syntax

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

Description

Use `p4 resolve` to combine the contents of two files or file revisions into a single file revision. 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
<i>yours</i>	The revision of the file in the client workspace	The file to which changes are being propagated (in integration terminology, this is the <i>target</i> file). Changes are made to the version of this file in the client workspace, and this file is later submitted to the depot.
<i>theirs</i>	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 <i>source</i> file). This file is not changed in the depot or the client workspace.
<i>base</i>	The file revision synced to the client workspace before it was opened for edit.	The previously-integrated revision of <i>theirs</i> . The latest common ancestor of both <i>yours</i> and <i>theirs</i> .
<i>merge</i>	A file version generated by Perforce from <i>yours</i> , <i>theirs</i> , and <i>base</i> . The user can edit this revision during the resolve process if the file is a text file.	Same as the meaning at left.

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

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
e	edit merged	Edit the preliminary merge file generated by Perforce.	no
ey	edit yours	Edit the revision of the file currently in the 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 <i>yours</i> and <i>base</i> .	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
dt	diff theirs	Show diffs between <i>theirs</i> and <i>base</i> .	no
dm	diff merge	Show diffs between <i>merge</i> and <i>base</i> .	no
d	diff	Show diffs between <i>merge</i> and <i>yours</i> .	yes
m	merge	Invoke the command: <code>P4MERGE base theirs yours merge</code> To use this option, you must set the environment variable <code>P4MERGE</code> 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 <code>P4MERGE</code> is not set.	no
?	help	Display help for <code>p4 resolve</code> .	yes
s	skip	Don't perform the resolve right now.	yes
ay	accept yours	Accept <i>yours</i> , ignoring changes that may have been made in <i>theirs</i> .	yes
at	accept theirs	Accept <i>theirs</i> into the client workspace as the resolved revision. The revision (<i>yours</i>) that was in the client workspace is overwritten. When resolving simple conflicts, this option is identical to performing <code>p4 revert</code> on the client workspace file. When resolving for integrate, this copies the source file to the target file.	yes
am	accept merge	Accept the <i>merged</i> file into the client workspace as the resolved revision without any modification. The revision (<i>yours</i>) originally in the client workspace is overwritten.	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
ae	accept edit	If you edited the file (i.e., by selecting “e” from the <code>p4 resolve</code> dialog), accept the edited version into the client workspace. The revision (<i>yours</i>) originally in the client workspace is overwritten.	no
a	accept	Keep Perforce’s recommended result: <ul style="list-style-type: none">• if <i>theirs</i> is identical to <i>base</i>, accept <i>yours</i>;• if <i>yours</i> is identical to <i>base</i>, accept <i>theirs</i>;• if <i>yours</i> and <i>theirs</i> are different from <i>base</i>, and there are no conflicts between <i>yours</i> and <i>theirs</i>; accept <i>merge</i>;• otherwise, there are conflicts between <i>yours</i> and <i>theirs</i>, so skip this file	no

Resolution of a file is completed when any of the accept dialog options are chosen. To resolve the file later or to revert the change, skip the file.

To help decide which option to choose, counts of four types of changes that have been made to the file revisions are displayed by `p4 resolve`:

Diff Chunks: 2 yours + 3 theirs + 5 both + 7 conflicting

The meanings of these values are:

Count	Meaning
<i>n</i> yours	<i>n</i> non-conflicting segments of <i>yours</i> are different than <i>base</i> .
<i>n</i> theirs	<i>n</i> non-conflicting segments of <i>theirs</i> are different than <i>base</i> .
<i>n</i> both	<i>n</i> non-conflicting segments appear identically in both <i>theirs</i> and <i>yours</i> , but are different from <i>base</i> .
<i>n</i> conflicting	<i>n</i> segments of <i>theirs</i> and <i>yours</i> are different from <i>base</i> 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	Skip the resolution dialog, and resolve the files automatically as follows:
-af	
-as	<ul style="list-style-type: none"> -am: Automatic Mode. Automatically accept the Perforce-recommended file revision: if <i>theirs</i> is identical to <i>base</i>, accept <i>yours</i>; if <i>yours</i> is identical to <i>base</i>, accept <i>theirs</i>; if <i>yours</i> and <i>theirs</i> are different from <i>base</i>, and there are no conflicts between <i>yours</i> and <i>theirs</i>; accept <i>merge</i>; otherwise, there are conflicts between <i>yours</i> and <i>theirs</i>, so skip this file.
-at	
-ay	<ul style="list-style-type: none"> -ay: Accept <i>Yours</i>, ignore <i>theirs</i>. -at: Accept <i>Theirs</i>. Use this flag with caution, as the file in the client workspace will be overwritten! -as: Safe Accept. If either <i>yours</i> or <i>theirs</i> is different from <i>base</i>, (and the changes are in common) accept that revision. If both are different from <i>base</i>, skip this file. -af: Force Accept. Accept the <i>merge</i> file no matter what. If the <i>merge</i> file has conflict markers, they will be left in, and you'll need to remove them by editing the file.
-dflag	When merging files, ignore specified differences in whitespace or line-ending convention. (If you use these flags, and the files differ by whitespace only, p4 resolve uses the text in the client file.)
	<ul style="list-style-type: none"> -db: Ignore whitespace-only changes (for instance, a tab replaced by eight spaces) -dw: Ignore whitespace altogether (for instance, deletion of tabs or other whitespace) -dl: Ignore differences in line-ending convention
-f	Allow already resolved, but not yet submitted, files to be resolved again.
-n	List the files that need resolving without actually performing the resolve.
-o	Output the base file name and revision to be used during the resolve.

<code>-t</code>	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 <code>P4MERGE</code> is set to a utility that can do such a thing.
<code>-v</code>	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.
<code>g-opts</code>	See the <i>Global Options</i> 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 than 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	<code>p4 resolved</code>
To schedule the propagation of changes between two separate files	<code>p4 integrate</code>
To submit a set of changed files to the depot	<code>p4 submit</code>
To copy a file to the client workspace, or schedule an open file for resolve	<code>p4 sync</code>

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

<code>-o</code>	Output the base file name and revision that was used during the resolve.
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 integrated</code>
To view a list of integrations that have not yet been resolved	<code>p4 resolve -n</code>
To schedule the propagation of changes from one file to another	<code>p4 integrate</code>
To resolve file conflicts, or to propagate changes as scheduled by <code>p4 integrate</code>	<code>p4 resolve</code>

p4 revert

Synopsis

Discard changes made to open files.

Syntax

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

Description

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

When you revert files you opened with `p4 delete`, the files are reinstated in the client workspace. When you revert files that have been opened by `p4 add`, Perforce leaves the client workspace files intact. When you revert files you've opened with `p4 integrate`, Perforce removes the files from the client workspace.

Options

<code>-a</code>	Revert only those files that haven't changed (in terms of content or filetype) since they were opened. The only files reverted are those whose client revisions are: <ul style="list-style-type: none"> • open for edit but have unchanged content and unchanged filetype; or • open for integrate via <code>p4 integrate</code> and have not yet been resolved with <code>p4 resolve</code>.
<code>-n</code>	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.
<code>-k</code>	Keep workspace files; the file(s) are removed from any changelists, and the server records the files as being no longer open, but the file(s) are unchanged in the client workspace.
<code>-c changelist#</code>	Reverts only those files in the specified changelist.
<code>g-opts</code>	See the <i>Global Options</i> 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

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

Related Commands

To open a file for add	<code>p4 add</code>
To open a file for deletion	<code>p4 delete</code>
To copy all open files to the depot	<code>p4 submit</code>
To read files from the depot into the client workspace	<code>p4 sync</code>
To list all opened files	<code>p4 opened</code>
To forcibly bring the client workspace in sync with the files that Perforce thinks you have, overwriting any unopened, writable files in the process.	<code>p4 sync -f</code>

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

<code>-c changelist#</code>	List all submitted changelists above and including <i>changelist#</i> .
<code>-t countername</code>	List all submitted changelists above the value of the Perforce counter <i>countername</i> .
<code>-c changelist# -t countername</code>	Set the value of counter <i>countername</i> to <i>changelist#</i> . This command has been replaced by <code>p4 counter</code> , but has been maintained for backwards compatibility.
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 reviews</code>
To set or read the value of a Perforce counter	<code>p4 counter</code>
To see full information about a particular changelist	<code>p4 describe</code>
To see a list of all changelists, limited by particular criteria	<code>p4 changes</code>

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

<code>-c changelist#</code>	List all users who have subscribed to reviews any files included in submitted changelist <code>changelist#</code> .
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 user</code>
List all submitted changelists above a provided changelist number	<code>p4 review</code>
To set or read the value of a Perforce counter	<code>p4 counter</code>
To read full information about a particular changelist	<code>p4 describe</code>

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

<code>-s</code>	Set the value of the registry variables for the local machine. Without this flag, <code>p4 set</code> sets the variables in the <code>HKEY_CURRENT_USER</code> hive; when you use the <code>-s</code> flag, the variables are set in the <code>HKEY_LOCAL_MACHINE</code> hive. These locations are reflected in the output of <code>p4 set</code> on Windows.
<code>-S svcname</code>	Set the value of the registry variables as used by service <i>svcname</i> . You must have administrator privileges to do this.
<code>g-opts</code>	See the <i>Global Options</i> 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 sizes

Synopsis

Display size information for files in the depot.

Syntax

```
p4 [g-opts] sizes [ -a -s -b blocksize ] file[revRange] ...
```

Description

The `p4 sizes` command displays the sizes of files stored in the depot. When called with no options, only the size of the head revision of the file or files is displayed. One line of output is provided per file.

Use the `-a` option to see how much space is occupied by each individual revision in the specified revision range, rather than just the highest revision in the specified range. One line of output is provided per file, per revision.

Use the `-s` option to obtain the sum of all files specified. Only one line of output is provided, showing the file specification, the number of files summarized, the total number of bytes required, and (if the `-b` option is provided) the total number of blocks required.

Options

<code>-a</code>	Include all revisions within the range, rather than just the highest revision in the range.
<code>-b <i>blocksize</i></code>	Display results in blocks of <i>blocksize</i> bytes. Each accumulated file size is rounded up to the nearest <i>blocksize</i> bytes.
<code>-s</code>	Calculate the sum of the file sizes for the specified file argument.
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

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

- The `p4 sizes` command is functionally similar to the UNIX `du` command.
- If no revision range is specified, the implicit revision range of `#1` through `#head` is assumed.

- File sizes are based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.

Examples

<code>p4 sizes file.c</code>	Show the size of the head revision of <code>file.c</code> in the depot.
<code>p4 sizes -a file.c</code>	Show the sizes of each revision of <code>file.c</code> stored in the depot.
<code>p4 sizes -s -a file.c</code>	Show the total size of all revisions of <code>file.c</code> stored in the depot.
<code>p4 sizes -s -a -b 512 //depot/...</code>	Show the number of files and the total diskspace (in bytes and 512-byte blocks) currently used by a Perforce Server hosting <code>//depot/...</code>
<code>p4 sizes -s //workspace/...</code>	Show the number of files and the total local diskspace (in bytes) required to sync the head revisions of files mapped to the client workspace named <code>workspace</code> .

p4 submit

Synopsis

Send changes made to open files to the depot.

Syntax

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

Description

When a file has been opened by `p4 add`, `p4 edit`, `p4 delete`, or `p4 integrate`, the file is listed in a *changelist*. The user's changes to the file are made only within in the client workspace copy until the changelist is sent to the depot with `p4 submit`.

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

By default, all files in the changelist are submitted to the depot, and files open for `edit`, `add`, and `branch` are closed when submitted, whether there are any changes to the files or not. To change this default behavior, set the `SubmitOptions:` field in the `p4 client` form for your workspace. To override your workspace's `SubmitOptions:` setting from the command line, use `p4 submit -f submitoption`.

When used with the default changelist, `p4 submit` brings up a form for editing in the editor defined by the `EDITOR` (or `P4EDITOR`) environment or registry variable. Files can be deleted from the changelist by deleting them from the form, but these files will remain open in the next default changelist. To close a file and remove it from all changelists, use `p4 revert`.

All changelists have a `Status:` field; the value of this field is `pending` or `submitted`. Submitted changelists have been successfully submitted with `p4 submit`; pending changelists have been created by the user but not yet been submitted successfully.

`p4 submit` works atomically: either all the files listed in the changelist are saved in the depot, or none of them are. `p4 submit` fails if it is interrupted, or if any of the files in the changelist are not found in the current client workspace, are locked in another client workspace, or require resolution and remain unresolved.

If `p4 submit` fails while processing the default changelist, the changelist is assigned the next number in the changelist sequence, and the default changelist is emptied. The

changelist that failed submission must be resubmitted by number after the problems are fixed.

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

Form Fields

Field Name	Type	Description
Change:	Read-only	The change number, or <i>new</i> 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 <i>pending</i> , <i>submitted</i> , or <i>new</i> . Not editable by the user. The status is <i>new</i> when the changelist is created; <i>pending</i> when it has been created but has not yet been submitted to the depot with <code>p4 submit</code> , and <i>submitted</i> when its contents have been stored in the depot with <code>p4 submit .</code>
Description:	Writable	Textual description of changelist. This value <i>must</i> 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 <code>p4 user</code> 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

<code>-c changelist#</code>	<p>Submit changelist number <i>changelist#</i>.</p> <p>Changelists are assigned numbers either manually by the user with <code>p4 change</code>, or automatically by Perforce when submission of the default changelist fails.</p>
<code>-d description</code>	<p>Immediately submit the default changelist with the <i>description</i> supplied on the command line, and bypass the interactive form. This option is useful when scripting, but does not allow for jobs to be added, nor for the default changelist to be modified.</p>
<code>-f submitoption</code>	<p>Override the <code>SubmitOptions:</code> setting in the <code>p4 client</code> form. Valid <i>submitoption</i> values are:</p> <ul style="list-style-type: none">• <code>submitunchanged</code> All open files (with or without changes) are submitted to the depot. This is the default behavior of Perforce.• <code>submitunchanged+reopen</code> All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist.• <code>revertunchanged</code> Only those files with content or type changes are submitted to the depot. Unchanged files are reverted.• <code>revertunchanged+reopen</code> Only those files with content or type changes are submitted to the depot and reopened in the default changelist. Unchanged files are reverted and <i>not</i> reopened in the default changelist.• <code>leaveunchanged</code> Only those files with content or type changes are submitted to the depot. Any unchanged files are moved to the default changelist.• <code>leaveunchanged+reopen</code> Only those files with content or type changes are submitted to the depot. Unchanged files are moved to the default changelist, and changed files are reopened in the default changelist. This option is similar to <code>submitunchanged+reopen</code>, except that no unchanged files are submitted to the depot.
<code>-i</code>	<p>Read a changelist specification from standard input. Input must be in the same format as that used by the <code>p4 submit</code> form.</p>

<code>-r</code>	Reopen files for <code>edit</code> in the default changelist after submission. Files opened for <code>add</code> or <code>edit</code> in will remain open after the submit has completed.
<code>-s</code>	<p>Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of <code>closed</code>.</p> <p>On new changelists, the fix status is displayed as the special status <code>ignore</code>. (If the status is left unchanged, the job is not fixed by the submission of the changelist.)</p> <p>This option works in conjunction with the <code>-s</code> option to <code>p4 fix</code>, and is intended for use by Perforce Defect Tracking Integration (P4DTI).</p>
<code>g-opts</code>	See the <i>Global Options</i> section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	<code>write</code>

- 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

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

Related Commands

To create a new, numbered changelist	<code>p4 change</code>
To open a file in a client workspace and list it in a changelist	<code>p4 add</code> <code>p4 edit</code> <code>p4 delete</code> <code>p4 integrate</code>
To move a file from one changelist to another	<code>p4 reopen</code>
To remove a file from all changelists, reverting it to its previous state	<code>p4 revert</code>
To view a list of changelists that meet particular criteria	<code>p4 changes</code>
To read a full description of a particular changelist	<code>p4 describe</code>
To read files from the depot into the client workspace	<code>p4 sync</code>
To edit the mappings between files in the client workspace and files in the depot	<code>p4 client</code>

p4 sync

Synopsis

Copy files from the depot into the workspace.

Syntax

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

Description

`p4 sync` brings the client workspace into sync with the depot by copying files matching its file pattern arguments from the depot to the client workspace. When no file patterns are specified on the command line, `p4 sync` copies a particular depot file only if it meets all of the following criteria:

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

In new, empty, workspaces, all depot files meet the last two criteria, so all the files visible through the workspace view are copied into the user's workspace.

If file patterns are specified on the command line, only those files that match the file patterns and that meet the above criteria are copied.

If the file pattern contains a revision specifier, the specified revision is copied into the client workspace.

If the file argument includes a revision range, only files selected by the revision range are updated, and the highest revision in the range is used. Files that are no longer in the workspace view are not affected if the file argument includes a revision range.

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

Options

-f	Force the sync. Perforce performs the sync even if the client workspace already has the file at the specified revision. If the file is writable, it is overwritten. This flag does not affect open files, but it <i>does</i> override the <code>noclobber</code> client option.
-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.
-k	Keep existing workspace files; update the have list without updating the client workspace. Use <code>p4 sync -k</code> only when you need to update the have list to match the actual state of the client workspace.
-p	Populate a client workspace, but do not update the have list. Any file that is already synced or opened is bypassed with a warning message. This option is typically used for workspaces used in processes (such as certain build or publication environments) where there is no need to track the state of the workspace after it has first been synced.
<i>g-opts</i>	See the <i>Global Options</i> section.

Usage Notes

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

- If the client view has changed since the last sync, the next sync removes from the client workspace those files that are no longer visible through the client view (unless a revision range is used), and copies into the client workspace those depot files that were not previously visible.

By default, any empty directories in the client view are cleared of files, but the directories themselves are not deleted. To remove empty directories upon syncing, turn on the `rmdir` option in the `p4 client` form.

- If a user has made certain files writable by using OS commands outside of Perforce's control, `p4 sync` will not normally overwrite those files. If the `clobber` option in the `p4 client` form has been turned on, however, these files will be overwritten.
- `p4 flush` is an alias for `p4 sync -k`. All of the warnings that apply to `p4 flush` also apply to `p4 sync -k`.

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%40junelst.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@junelst.txt`.

For details, see “Limitations on characters in filenames and entities” on page 250.

```
p4 sync file.c#none
```

Sync to the nonexistent revision of `file.c`; the file is deleted from the workspace.

```
p4 sync ...#none
```

Sync to the nonexistent revision of all files; all files in the workspace (that are under Perforce control) are removed.

Related Commands

To open a file in a client workspace and list it in a changelist

```
p4 add  
p4 edit  
p4 delete  
p4 integrate
```

To copy changes to files in the client workspace to the depot

```
p4 submit
```

To view a list of files and revisions that have been synced to the client workspace

```
p4 have
```

p4 tag

Synopsis

Tag files with a label.

Syntax

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

Description

Use `p4 tag` to tag specified file revisions with a label. A *labelname* is required. If a label named *labelname* does not exist, it is created automatically. If the label already exists, you must be the `Owner`: of the label and the label must be `unlocked` in order for you to tag or untag files with the label. (Use `p4 label` to change label ownership or lock status.)

If the *file* argument does not include a revision specification, the head revision is tagged with the label. If the file argument includes a revision range specification, only files with revisions in that range are tagged. (If more than one revision of the file exists in the specified range, the highest revision in the specified range is tagged.)

Options

<code>-d</code>	Delete the label tag from the named files.
<code>-n</code>	Display what <code>p4 tag</code> would do without actually performing the operation.
<code>-l <i>labelname</i></code>	Specify the label to be applied to file revisions
<code>g-opts</code>	See the <i>Global Options</i> 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

<code>g-opts</code>	See the <i>Global Options</i> section.
---------------------	--

Usage Notes

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

- Tickets are stored in the file specified by the `P4TICKETS` environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.

Examples

<code>p4 tickets</code>	Display all tickets stored in a user's local ticket file.
-------------------------	---

Related Commands

To start a login session (to obtain a ticket)	<code>p4 login</code>
To end a login session (to delete a ticket)	<code>p4 logout</code>

p4 triggers

Synopsis

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

Syntax

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

Description

Perforce *triggers* are user-written scripts that are called by a Perforce server whenever certain operations (such as changelist submission or changes to forms) are performed. If the script returns a value of 0, the operation continues; if the script returns any other value, the operation fails. Upon failure, the script's standard output (not error output) is used as the text of the failed operation's error message.

Perforce supports twelve trigger types, divided into four categories. *Changelist submission triggers* (change-submit, change-content, and change-commit) are fired when users submit changelists. *Fix triggers* (fix-add and fix-delete) are fired when fixes are added or deleted from changelists by means of the p4 fix, p4 submit, or p4 change commands. *Form triggers* (form-save, form-out, form-in, form-commit, and form-delete) are fired when users generate or modify form specifications. *Authentication triggers* (auth-check and auth-set) are fired when administrators wish to integrate Perforce with external authentication mechanisms such as LDAP or Active Directory.

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

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

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

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

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

To configure Perforce to run trigger scripts when users edit specification forms, use *form triggers*: these are triggers of type `form-save`, `form-in`, `form-out`, `form-commit`, and `form-delete`. Use form triggers to generate customized 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.

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

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

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

Form Fields

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

Field	Meaning
<i>name</i>	The user-defined name of the trigger. A run of the same trigger name on contiguous lines is treated as a single trigger, so that multiple <i>paths</i> may be specified. In this case, only the <i>command</i> of the first such trigger line is used.

Field	Meaning
<i>type</i>	<p>There are twelve trigger types, divided into four subtypes: changelist submission triggers, fix triggers, form triggers, and authentication triggers. Changelist submission triggers:</p> <ul style="list-style-type: none">• <code>change-submit</code>: Execute a changelist trigger after changelist creation, but before file transfer. Trigger may not access file contents.• <code>change-content</code>: Execute a changelist trigger after changelist creation and file transfer, but before file commit. <p>To obtain file contents, use commands such as <code>p4 diff2</code>, <code>p4 files</code>, <code>p4 fstat</code>, and <code>p4 print</code> with the revision specifier <code>@=change</code>, where <i>change</i> is the changelist number of the pending changelist as passed to the script in the <code>%changelist%</code> variable.</p> <ul style="list-style-type: none">• <code>change-commit</code>: Execute a changelist trigger after changelist creation, file transfer, and changelist commit. <p>Fix triggers:</p> <p>The special variable <code>%jobs%</code> is available for expansion; it expands to one argument for every job listed on the <code>p4 fix</code> command line (or in the <code>Jobs:</code> field of a <code>p4 change</code> or <code>p4 submit</code> form), and must therefore be the last argument supplied to the trigger script.</p> <ul style="list-style-type: none">• <code>fix-add</code>: Execute fix trigger prior to adding a fix.• <code>fix-delete</code>: Execute fix trigger prior to deleting a fix.

Field	Meaning
	<p>Form triggers:</p> <ul style="list-style-type: none"> • <code>form-save</code>: Execute a form trigger after its contents are parsed, but before its contents are stored in the Perforce database. The trigger cannot modify the form specified in <code>%formfile%</code> variable. • <code>form-out</code>: Execute form trigger upon generation of form to end user. The trigger can modify the form. • <code>form-in</code>: Execute form trigger on edited form before contents are parsed and validated by the Perforce server. The trigger can modify the form. • <code>form-delete</code>: Execute form trigger after its contents are parsed, but before the specification is deleted from the Perforce database. Trigger may not modify form. • <code>form-commit</code>: Execute form trigger after it has been committed, enabling access to automatically-generated fields such as <code>jobname</code>, <code>dates</code>, etc. For job forms, this trigger is run by <code>p4 job</code> as well as <code>p4 fix</code> (after the status is updated). The <code>form-commit</code> trigger has access to the new job name created by <code>p4 job</code>; any <code>form-in</code> and <code>form-save</code> triggers are run before the job name is created. For job forms, this trigger is also run by <code>p4 change</code> (if a job is added or deleted by editing the <code>Jobs:</code> field of the changelist), and <code>p4 submit</code> (for any jobs present in the <code>Jobs:</code> field of the changelist). In these cases, the special variable <code>%action%</code> is available for expansion on the job <code>form-commit</code> trigger command line. The trigger cannot modify the form. <p>Authentication triggers:</p> <ul style="list-style-type: none"> • <code>auth-check</code>: Execute an authentication check trigger to verify a user's password against an external password manager during login, or when setting a new password. If an <code>auth-check</code> trigger is present, the Perforce security counter (and any associated password strength requirement) is ignored, as authentication is now controlled by the trigger script. • <code>auth-set</code>: Execute an authentication set trigger to send a new password to an external password manager. <p>You must restart the Perforce server after adding an <code>auth-check</code> trigger.</p>

Field	Meaning
<i>path</i>	<p>For changelist submission triggers (change-submit, change-content, or change-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.</p> <p>For fix triggers (fix-add or fix-delete), use fix as the path value.</p> <p>For form triggers (form-save, form-out, form-in, form-commit, or form-delete), the name of the type of form, (one of branch, change, client, depot, group, job, label, protect, spec, triggers, typemap, or user). Triggers that fire on the <code>p4 triggers</code> command are ignored.</p> <p>For authentication triggers (auth-check or auth-set), use auth as the path value.</p>
<i>command</i>	<p>The command for the Perforce server to run when a matching <i>path</i> 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.</p> <p>For change-submit and change-content triggers, changelist submission continues if the trigger script exits with 0, or fails if the script exits with a nonzero value. For change-commit triggers, changelist submission succeeds regardless of the trigger script's exit code, but subsequent change-commit triggers do not fire if the script exits with a nonzero value.</p> <p>For form-in, form-out, form-save, and form-delete triggers, the data in the specification becomes part of the Perforce database if the script exits with 0. Otherwise, the database is not updated.</p> <p>The form-commit trigger type never rejects a change; it exists primarily so that scripts can access a job number (from the <code>%formname%</code> variable) during the process of job creation.</p> <p>For fix-add and fix-delete triggers, fix addition or deletion continues if the trigger script exits with 0, or fails if the script exits with a nonzero value.</p> <p>For auth-check triggers (fired by <code>p4 login</code>), the user's typed password is supplied to the trigger command as standard input. If the trigger executes successfully, the Perforce ticket is issued. The user name is available as <code>%user%</code> to be passed on the command line.</p> <p>For auth-set triggers, (fired by <code>p4 passwd</code>, but only after also passing an auth-check trigger check) the user's old password and new password are passed to the trigger as standard input. The user name is available as <code>%user%</code> to be passed on the command line.</p>

Options

<code>-i</code>	Read the trigger table from standard input without invoking the editor.
<code>-o</code>	Write the trigger table to standard output without invoking the editor.
<code>g-opts</code>	See the <i>Global Options</i> 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
<code>%action%</code>	Either null or a string reflecting an action taken to a changelist or job. For example, “pending change 123 added” or “submitted change 124 deleted” are possible <code>%action%</code> values on change forms, and “job000123 created” or “job000123 edited” are possible <code>%action%</code> values for job forms.	<code>form-commit</code>
<code>%changelist%</code> <code>%change%</code>	The number of the changelist being submitted. (The abbreviated form <code>%change%</code> is equivalent to <code>%changelist%</code> .)	<code>change-submit</code> , <code>change-content</code> , <code>change-commit</code> , <code>fix-add</code> , <code>fix-delete</code> , <code>form-commit</code>
<code>%client%</code>	Triggering user’s client workspace name.	<code>all</code>
<code>%clienthost%</code>	Hostname of the client.	<code>all</code>
<code>%clientip%</code>	The IP address of the client.	<code>all</code>

Argument	Description	Available for type
%jobs%	A string of job numbers, expanded to one argument for each job number specified on a <code>p4 fix</code> command or for each job number added to (or removed from) the <code>Jobs:</code> field in a <code>p4 submit</code> , or <code>p4 change</code> form.	<code>fix-add</code> <code>fix-delete</code>
%serverhost%	Hostname of the Perforce server.	<code>all</code>
%serverip%	The IP address of the server.	<code>all</code>
%serverport%	The IP address and port of the Perforce server, in the format <code>ip_address:port</code> .	<code>all</code>
%serverroot%	The <code>P4ROOT</code> directory of the Perforce server.	<code>all</code>
%user%	Perforce username of the triggering user.	<code>all</code>
%formfile%	Path to temporary form specification file. To modify the form from an in or out trigger, overwrite this file. The file is read-only for triggers of type <code>save</code> and <code>delete</code> .	<code>form-commit</code> , <code>form-save</code> , <code>form-out</code> , <code>form-in</code> , <code>form-delete</code>
%formname%	Name of form (for instance, a branch name or a changelist number).	<code>form-commit</code> , <code>form-save</code> , <code>form-out</code> , <code>form-delete</code>
%formtype%	Type of form (for instance, <code>branch</code> , <code>change</code> , and so on).	<code>form-commit</code> , <code>form-save</code> , <code>form-out</code> , <code>form-in</code> , <code>form-delete</code>

- If your trigger script needs to know what files were (or are about to be) submitted in the changelist, use the command `p4 opened -ac changelist`.
- Pre-submit trigger scripts cannot access submitted file contents from the server, because at the time a pre-submit trigger runs, file contents have not yet been transferred to the server.
- Perforce commands in trigger scripts are always run by a specific Perforce user. If no user is specified, an extra Perforce license for a user named `SYSTEM` (or on UNIX, the user that owns the `p4d` process) is assumed. To prevent this from happening:
 - Pass a `%user%` argument to the script that calls each Perforce command to ensure that each command is called by. For example, if Joe submits a changelist that activates

trigger script `trigger.pl`, and `trigger.pl` calls the `p4 changes` command, the script can run the command as `p4 -u %user% changes`.

- Set `P4USER` for the account that runs the trigger script to the name of an existing user. (If your Perforce server is installed as a service under Windows, note that Windows services cannot have a `P4USER` value; on Windows, you must therefore pass a user value to each command as described above.)
- For the four form trigger types (`form-in`, `form-out`, `form-save`, and `form-delete`), the `%formname%` variable is unset on job creation. This limitation is due to the fact that a job's name is unknown to the server until after job creation.

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

After job creation, subsequent user changes to a job correctly set `%formname%` for use by form trigger scripts.

- Trigger types were renamed in Release 2005.2. The following old trigger type names will continue to work but are deprecated:

Old trigger type	New trigger type (as of 2005.2)
<code>submit</code>	<code>change-submit</code>
<code>content</code>	<code>change-content</code>
<code>commit</code>	<code>change-commit</code>
<code>out</code>	<code>form-out</code>
<code>in</code>	<code>form-in</code>
<code>save</code>	<code>form-save</code>
<code>delete</code>	<code>form-delete</code>

- As of Release 2007.3, trigger standard output is passed to the client program on both success and failure of the trigger script. Prior to this, standard output was only passed to the client program upon failure of a trigger.

Examples

Suppose that the trigger table consists of the following entries:

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

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

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

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

Related Commands

To obtain information about the changelist being submitted	p4 describe p4 opened
To aid daemon creation	p4 review p4 reviews p4 counter p4 counters p4 user

p4 typemap

Synopsis

Modify the file name-to-type mapping table.

Syntax

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

Description

The `p4 typemap` command allows Perforce administrators to set up a table linking Perforce file types to file name specifications. If a filename matches an entry in the typemap table, it overrides the file type that would otherwise have been assigned by the Perforce client.

By default, Perforce automatically determines if a file is of type `text` or `binary` based on an analysis of the first 8192 bytes of a file. If the high bit is clear in each of the first 8192 bytes, Perforce assumes it to be `text`; otherwise, it's `binary`.

Although this default behavior can be overridden by the use of the `-t filetype` flag, it's easy to overlook this, particularly in cases where files' types were usually (but not always) detected correctly. The most common examples of this are associated with PDF files (which sometimes begin with over 8192 bytes of ASCII comments) and RTF files, which usually contain embedded formatting codes.

The `p4 typemap` command provides a more complete solution, allowing administrators to bypass the default type detection mechanism, ensuring that certain files (for example, those ending in `.pdf` or `.rtf`) will always be assigned the desired Perforce filetype upon addition to the depot.

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

Form Fields

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

Column	Description
<i>filetype</i>	Any valid Perforce file type. For a list of valid file types, see the <i>File Types</i> section.
<i>pattern</i>	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

<code>-i</code>	Reads the typemap table from standard input without invoking the user's editor.
<code>-o</code>	Writes the typemap table to standard output without invoking the user's editor.
<code>g-opts</code>	See the <i>Global Options</i> 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 <code>list</code> to use the <code>-o</code> flag

- To specify all files with a given extension at or below a desired subdirectory, use four periods after the directory name, followed by the extension. (for instance, `//path/...ext`) The first three periods specify “all files below this level”. The fourth period and accompanying file extension are parsed as “ending in these characters”.
- File type modifiers may be used in the typemap table. Useful applications include forcing keyword expansion on or off across directory trees, enforcing the preservation of original file modification times (the `+m` file type modifier) in directories of third-party DLLs, or implementing pessimistic locking policies.
- If you use the `-t` flag and file type modifiers to specify a file type on the command line, and the file to which you are referring falls under a `p4 typemap` mapping, the file type specified on the command line overrides the file type specified by the typemap table.

Examples

To tell the Perforce server to regard all PDF and RTF files as binary, use `p4 typemap` to modify the typemap table as follows:

```
Typemap:
    binary //...pdf
    binary //...rtf
```

The first three periods (“...”) in the specification are a Perforce wildcard specifying that all files beneath the root directory are included as part of the mapping. The fourth period and the file extension specify that the specification applies to files ending in “.pdf” (or “.rtf”)

A more complicated situation might arise in a site where users in one area of the depot use the extension .doc for plain ASCII text files containing documentation, and users working in another area use .doc to refer to files in a binary file format used by a popular word processor. A useful typemap table in this situation might be:

```
Typemap:
    text //depot/dev_projects/...doc
    binary //depot/corporate/annual_reports/...doc
```

To enable keyword expansion for all .c and .h files, but disable it for your .txt files, do the following:

```
Typemap:
    text+k //depot/dev_projects/main/src/...c
    text+k //depot/dev_projects/main/src/...h
    text //depot/dev_projects/main/src/...txt
```

To ensure that files in a specific directory have their original file modification times preserved (regardless of submission date), use the following:

```
Typemap:
    binary //depot/dev_projects/main/bin/...
    binary+m //depot/dev_projects/main/bin/thirdpartydll/...
```

All files at or below the `bin` directory are assigned type `binary`. Because later mappings override earlier mappings, files in the `bin/thirdpartydll` subdirectory are assigned type `binary+m` instead. For more information about the `+m` (modtime) file type modifier, see the *File Types* section.

By default, Perforce supports concurrent development, but environments in which only one person is expected to have a file for edit at a time can implement pessimistic locking by using the +1 (exclusive open) modifier as a partial filetype. If you use the following typemap, the +1 modifier is automatically applied to all newly-added files in the depot:

```
Typemap:
    +1 //depot/...
```

Related Commands

To add a new file with a specific type, overriding the typemap table	<code>p4 add -t <i>type file</i></code>
To change the filetype of an opened file, overriding any settings in the typemap table	<code>p4 reopen -t <i>type file</i></code>

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

<code>-c changelist#</code>	Unlock files in pending changelist <i>changelist#</i>
<code>-f</code>	Superuser force flag; allows unlocking of files opened by other users.
<code>g-opts</code>	See the <i>Global Options</i> 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	<code>p4 lock</code>
To display all your open, locked files (UNIX)	<code>p4 opened grep "*locked*"</code>

p4 user

Synopsis

Create or edit Perforce user specifications and preferences.

Syntax

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

Description

By default, any system user becomes a valid Perforce user the first time he uses any Perforce command that can update the depot or its metadata. Perforce automatically creates a user spec with default settings for the invoking user. Use the `p4 user` command to edit these settings or to create new user records. (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 <code>p4 user</code> was invoked. By default, this is the user's system username.
Email:	Writable	The user's email address. By default, this is <code>user@client</code> .
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 <i>Usage Notes</i> below).
Password:	Writable	The user's password (described in the <i>Usage Notes</i> below).
Reviews:	Writable List	A list of files the user would like to review (see the <i>Usage Notes</i> below).

Options

<code>-d username</code>	Deletes the specified user. Only user <i>username</i> , or the Perforce superuser, can run this command.
<code>-f</code>	Superuser force flag; allows the superuser to modify or delete the specified user, or to change the last modified date.
<code>-i</code>	Read the user specification from standard input. The input must conform to the <code>p4 user</code> form's format.
<code>-o</code>	Write the user specification to standard output.
<code>g-opts</code>	See the <i>Global Options</i> 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

<code>p4 user joe</code>	View the user specification of Perforce user <code>joe</code> .
<code>p4 user</code>	Edit the user specification for the current Perforce user.
<code>p4 user -d sammy</code>	Delete the user specification for the Perforce user <code>sammy</code> .
<code>p4 -u joe -P hey submit</code>	Run <code>p4 submit</code> as user <code>joe</code> , whose password is <code>hey</code> . This command does not work at higher security levels.
<code>p4 user -f joe2</code>	Create a new Perforce user named <code>joe2</code> if the caller is a Perforce superuser, and <code>joe2</code> doesn't already exist as a Perforce user. If user <code>joe2</code> already exists, allow a Perforce superuser to modify the user's settings.

Related Commands

To view a list of all Perforce users	<code>p4 users</code>
To change a user's password	<code>p4 passwd</code>
To view a list of users who have subscribed to review particular files	<code>p4 reviews</code>

p4 users

Synopsis

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

Syntax

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

Description

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

If a *user* argument is provided, only information pertaining to that user is displayed. The *user* argument may contain the * wildcard; in this case, all users matching the given pattern are reported on. (If you use a wildcard, be sure to quote the user argument, because the OS will likely attempt to expand the wildcard to match file names in the current directory).

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

Options

<i>-m max</i>	List only the first <i>max</i> users.
<i>g-opts</i>	See the <i>Global Options</i> section.

Usage Notes

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

Related Commands

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

p4 verify

Synopsis

Verify that the server archives are intact.

Syntax

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

Description

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

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

Options

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

Usage Notes

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

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

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

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

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

p4 where

Synopsis

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

Syntax

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

Description

`p4 where` uses the client view and client root, as set in `p4 client`, to print files' locations relative to the top of the depot, relative to the top of the client workspace, and relative to the top of the local OS directory tree. The command does not check to see if the file exists; it merely reports where the file *would be* located if it *did* exist.

For each file provided as a parameter, a set of mappings is output. Each set of mappings is composed of lines consisting of three parts: the first part is the filename expressed in depot syntax, the second part is the filename expressed in client syntax, and the third is the local OS path of the file.

Options

<code>g-opts</code>	See the <i>Global Options</i> 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/b/file.txt /home/user/root/b/file.txt
```

This can be interpreted as saying that the first line of the client view would have caused the file to appear in `/home/user/root/a/b/file.txt`, except that it was overridden by the second mapping in the view. An exclusionary mapping was applied to perform the override, and the second mapping applies, sending the file to `/home/user/root/b/file.txt`.

- The simplest case (one line of output per file, showing each filename in depot, client, and local syntax) is by far the most common.

Examples

```
p4 where file.c
```

Show depot, client workspace, and local filesystem locations of `file.c` (or where `file.c` would appear if it existed in the depot.)

```
p4 where 100%25.txt
```

Use ASCII expansion of “%” character to locations for file `100%.txt`.

ASCII expansion is supported for the following four special characters: `@ (%40)`, `# (%23)`, `* (%2A)`, and `% (%25)`.

Related Commands

To list the revisions of files as synced from the depot

```
p4 have
```


p4 workspace

Synopsis

Create or edit a client workspace specification and its view.

Syntax

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

Description

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

p4 workspaces

Synopsis

List all client workspaces currently known to the system.

Syntax

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

Description

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

Environment and Registry Variables

Each operating system and shell has its own syntax for setting environment variables. The following table shows how to set the `P4CLIENT` environment variable in each OS and shell:

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

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
<code>P4CLIENT</code>	<code>P4CONFIG</code>	<code>P4PAGER</code>	<code>P4AUDIT</code>
<code>P4PORT</code>	<code>P4DIFF</code>	<code>PWD</code>	<code>P4JOURNAL</code>
<code>P4PASSWD</code>	<code>P4EDITOR</code>	<code>TMP, TEMP</code>	<code>P4LOG</code>
<code>P4USER</code>	<code>P4MERGE</code>	<code>P4LANGUAGE</code>	<code>P4PORT</code>

Crucial Variables	Useful Variables	Esoteric Variables	Server Variables
	P4CHARSET	P4TICKETS	P4ROOT
		P4COMMANDCHARSET	P4DEBUG
		P4DIFFUNICODE	
		P4MERGEUNICODE	

P4AUDIT

Description

Location of the server audit log file.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	<code>p4d -A auditlog</code>	N/A

Value if not Explicitly Set

Operating System	Value
All	None. If no log file is specified, auditing is disabled.

Notes

P4AUDIT specifies the location of the audit log file.

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

Lines in the audit log appear in the form:

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

For example:

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

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

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

P4CHARSET

Description

Character set used for translation of unicode files.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	<code>p4 -C charset cmd</code>	Yes

Value if not Explicitly Set

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

Notes

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

For servers operating in the default (non-Unicode mode), P4CHARSET must be left unset on client machines. If P4CHARSET is set, but the server is not operating in internationalized mode, the server returns the following error message:

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

For servers operating in Unicode mode, P4CHARSET must be set on client machines. If P4CHARSET is unset, but the server is operating in Unicode mode, client programs return the following error message:

```
Unicode server permits only unicode enabled clients.
```

For more about Unicode mode, including settings of P4CHARSET for various UTF-8, UTF-16, and UTF-32 character sets, with and without byte-order marks, see the *Internationalization Notes*:

```
http://www.perforce.com/perforce/doc.073/user/i18nnotes.txt
```

P4COMMANDCHARSET

Description

Used to support UTF-16 and UTF-32 character sets from the Command-line Client.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	<code>p4 -Q commandcharset cmd</code>	Yes

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

If you have set P4CHARSET to a UTF-16 or UTF-32 value, you must set P4COMMANDCHARSET to a non-UTF-16 or -32 value in order to use the p4 Command-line Client. For details, see the *Internationalization Notes*:

<http://www.perforce.com/perforce/doc.073/user/i18nnotes.txt>

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	<code>p4 -c <i>clientname</i> <i>cmd</i></code>	Yes

Value if not Explicitly Set

Operating System	Value
Windows	Value of <code>COMPUTERNAME</code> 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
- P4TICKETS
- P4USER

P4DEBUG

Description

Set Perforce server or proxy trace flags.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	None	No

Value if not Explicitly Set

Operating System	Value
All	If not set, this variable is not used.

Examples

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

Notes

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

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

Some server debug levels require specific server release levels.

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

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

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 <code>DIFF</code> has been set, then the value of <code>DIFF</code> ; otherwise, if the environment variable <code>SHELL</code> has been set to <i>any</i> value, then the program <code>diff</code> is used; otherwise, <code>p4diff.exe</code> .
All Others	If the environment variable <code>DIFF</code> has been set, then the value of <code>DIFF</code> ; otherwise, Perforce’s internal <code>diff</code> routine is used.

Examples

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

Notes

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

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

P4DIFFUNICODE

Description

Used to support UTF-16 and UTF-32 character sets from the Command-line Client.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

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

<http://www.perforce.com/perforce/doc.073/user/relnotes.txt>

P4EDITOR

Description

The editor invoked by those Perforce commands that use forms.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
UNIX	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	<code>p4 -H <i>hostname command</i></code>	Yes

Value if not Explicitly Set

Operating System	Value
All	The value of the client hostname as returned by <code>p4 info</code> .

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	<code>p4d -J file</code>	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	<code>p4 -L language cmd</code>	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	<code>p4d -L file</code> <code>p4p -L file</code>	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 <code>MERGE</code> environment variable (or registry variable on Windows, as set by <code>p4 set</code>) 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
```

P4MERGEUNICODE

Description

Used to support UTF-16 and UTF-32 character sets from the Command-line Client.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	None	Yes

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

This environment variable is used in place of P4MERGE if the file being resolved is of type `unicode` or `utf16`, and the character set is passed as the first argument to the command. For details, see the *Release Notes*:

<http://www.perforce.com/perforce/doc.073/user/relnotes.txt>

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 <code>PAGER</code> is set, then the value of <code>PAGER</code> ; 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	<code>p4 -P passwd command</code>	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	<code>p4p -r <i>directory</i></code>	N/A

Value if not Explicitly Set

Operating System	Value
All	<p>p4p's directory.</p> <p>Windows administrators running the Perforce Proxy process as a service should use <code>p4 set -S svcname P4PCACHE=<i>directory</i></code> to set the value of P4PCACHE for the named service.</p>

Notes

Create this directory before starting the Perforce Proxy (p4p).

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

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

P4PFSIZE

Description

For the Perforce Proxy, the size (in bytes) of the smallest file to be cached. All files larger than P4PFSIZE bytes in length are cached.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	p4p -e size	N/A

Value if not Explicitly Set

Operating System	Value
All	0; that is, cache all files

Notes

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

P4OPTIONS

Description

Set Perforce Proxy options for a Windows service.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	p4p %P4OPTIONS%	N/A

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

For example, if you normally run the Proxy with the command

```
p4p -p 1999 -t mainserver:1666
```

you can set the P4OPTIONS variable for the Windows proxysvc to run with

```
p4 set -S "Perforce Proxy" P4OPTIONS="-p 1999 -t mainserver:1666"
```

When you run P4P under the "Perforce Proxy" service, the Proxy will listen to port 1999 and communicate with the Perforce Server at mainserver:1666.

Most installations do not need to use P4OPTIONS, because there are already environment variables associated with most p4p flags; in the example shown above, you can use P4PORT and P4TARGET. Use P4OPTIONS when you need to call p4p with flags for which there are no corresponding environment variables, and when you are doing so within the context of a Windows service.

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

P4PORT

Description

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

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

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	<code>p4 -p <i>host:port cmd</i></code>	Yes

Value if not Explicitly Set

Program	Value
Perforce server	1666
Perforce proxy	1666
Perforce client	<code>perforce:1666</code>

Examples

Perforce client examples	Perforce server examples
1818	1818
<code>squid:1234</code>	1234
<code>perforce.squid.com:1234</code>	1234
<code>192.168.0.123:1818</code>	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	<code>p4d -r <i>directory</i></code>	N/A

Value if not Explicitly Set

Operating System	Value
All	<p>p4d's directory.</p> <p>Windows administrators running the Perforce back-end process as a service should use <code>p4 set -S <i>svcname</i> P4ROOT=<i>directory</i></code> to set the value of P4ROOT for the named service.</p>

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	<code>p4p -t <i>host:port</i></code>	N/A

Value if not Explicitly Set

Program	Value
Perforce Proxy	<code>perforce:1666</code>

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*.

P4TICKETS

Description

The location of the ticket file used by `p4 login`.

Usage Notes

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

Value if not Explicitly Set

Program	Value
Windows	%USERPROFILE%\p4tickets.txt
All others	\$HOME/.p4tickets

Examples

```
/staff/username/p4tickets.txt
```

Notes

The `P4TICKETS` environment variable must point to the actual ticket file, not merely a directory in which `p4tickets.txt` or `.p4tickets` is expected to exist. If you set `P4TICKETS` to point to a directory, you will not be able to log in.

P4USER

Description

Current Perforce username.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	<code>p4 -u username command</code>	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	<code>p4 -d <i>directory command</i></code>	No

Value if not Explicitly Set

Operating System	Value
UNIX	The value of <code>PWD</code> as set by the shell; if not set by the shell, <code>getcwd()</code> 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 `csch`, `PWD` will be inherited from your `csch` environment but not updated properly, causing possible confusion in subsequent Perforce commands.

If you encounter such difficulties, check to be sure you've unset `PWD` in your `.profile` or `.kshrc` file. (If you're running `sh` or `ksh` as your login shell, `PWD` will be managed properly by the shell regardless of any unsettings you've placed in your startup files; the confusion only occurs when variables are exported to subshells.)

TMP, TEMP

Description

The directory to which Perforce clients and servers write temporary files.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	None	No

Value if not Explicitly Set

Operating System	Value
UNIX	/tmp
All Others	On Perforce clients: the current working directory. On Perforce servers: P4ROOT

Notes

If TEMP is set, TEMP is used. Otherwise, if TMP is set, this is used. If neither TEMP nor TMP are set, temporary files will be written in the directories described in the table above.

Additional Information

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

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

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

Global Options

Synopsis

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

Syntax

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

Options

<code>-c client</code>	Overrides any <code>P4CLIENT</code> setting with the specified client name.
<code>-d dir</code>	Overrides any <code>PWD</code> setting (i.e. current working directory) and replaces it with the specified directory.
<code>-G</code>	Causes all output (and batch input for form commands with <code>-i</code>) to be formatted as marshalled Python dictionary objects. This is most often used when scripting.
<code>-H host</code>	Overrides any <code>P4HOST</code> setting and replaces it with the specified hostname.
<code>-p port</code>	Overrides any <code>P4PORT</code> setting with the specified port number.
<code>-P pass</code>	Overrides any <code>P4PASSWD</code> setting with the specified password.
<code>-s</code>	Prepends a descriptive field (for example, <code>text:</code> , <code>info:</code> , <code>error:</code> , <code>exit:</code>) to each line of output produced by a Perforce command. This is most often used when scripting.
<code>-u user</code>	Overrides any <code>P4USER</code> , <code>USER</code> , or <code>USERNAME</code> setting with the specified user name.
<code>-x file</code>	Instructs Perforce to read arguments, one per line, from the specified file. If file is a single hyphen (<code>-</code>), then standard input is read.
<code>-C charset</code>	Overrides any <code>P4CHARSET</code> setting with the specified character set.
<code>-Q charset</code>	Overrides any <code>P4COMMANDCHARSET</code> setting with the specified character set.
<code>-L language</code>	This feature is reserved for system integrators.
<code>-V</code>	Displays the version of the <code>p4</code> client program and exits.
<code>-h</code>	Displays basic usage information and exits.

Usage Notes

- Be aware that the global options must be specified on the command line before the Perforce command. Options specified after the Perforce command will not be interpreted as global options, but as options for the command being invoked. It is therefore possible to have the same command line option appearing twice in the same command, being interpreted differently each time.

For example, the command `p4 -c anotherclient edit -c 140 file.c` will open file `file.c` for edit in pending changelist 140 under client workspace `anotherclient`.

- The `-x` option is useful for automating tedious tasks; a user adding several files at once could create a text file with the names of these files and invoke `p4 -x textfile add` to add them all at once.

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

In this example, the `grep` command lists occurrences of `file.h` in the `*.c` files, the `-l` option tells `grep` to list each file only once, and the `cut` command splits off the filename from `grep`'s output before passing it to the `p4 -x` command.

- The `-s` option can be useful in automated scripts.

For example, a script could be written as part of an in-house build process which executes `p4 -s` commands, discards any output lines beginning with "info:", and alerts the user if any output lines begin with "error:".

- Python developers will find the `-G` option extremely useful for scripting. For instance, to get a dictionary of all fields of a job whose ID is known, use the following:

```
job_dict = marshal.load(os.popen('p4 -G job -o ' + job_id, 'rb'))
```

In some cases, it may not be intuitively obvious what keys 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, 'rb')
        for key in dict.keys(): print "%s: %s" % (key,dict[key])
except EOFError: pass
```

Python developers on Windows should be aware of potential CR/LF translation issues; in the example, it is necessary to call `marshal.load()` to read the data in binary (“rb”) mode.

- Some uses of the global options are absurd.

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

Examples

<code>p4 -p new_server:1234 sync</code>	Performs a sync using server <code>new_server</code> and port 1234, regardless of the settings of the <code>P4PORT</code> environment variable or registry setting.
<code>p4 -c new_client submit -c 100</code>	The first <code>-c</code> is the global option to specify the client name. The second <code>-c</code> specifies a changelist number.
<code>p4 -s -x filelist.txt edit</code>	<p>If <code>filelist.txt</code> contains a list of files, this command opens each file on the list for editing, and produces output suitable for parsing by scripts.</p> <p>Any errors as a result of the automated <code>p4 edit</code> commands (for example, a file in <code>filelist.txt</code> not being found) can then be easily detected by examining the command’s output for lines beginning with “error:”</p>

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, when used in views.

For example:

Expression	Matches
<code>J*</code>	Files in the current directory starting with <code>J</code>
<code>*/help</code>	All files called <code>help</code> in current subdirectories
<code>./...</code>	All files under the current directory and its subdirectories
<code>./....c</code>	All files under the current directory and its subdirectories, that end in <code>.c</code>
<code>/usr/bruno/...</code>	All files under <code>/usr/bruno</code>
<code>//bruno_ws/...</code>	All files in the workspace or depot that is named <code>bruno_ws</code>
<code>//depot/...</code>	All files in the depot
<code>//...</code>	All files in all depots

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
<code>file#n</code>	Revision specifier: The <i>n</i> th revision of <i>file</i> .
<code>file#none</code> <code>file#0</code>	The nonexistent revision: If a revision of <i>file</i> 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 <code>p4 sync file#none</code> . The filespec <code>#0</code> may be used as a synonym for <code>#none</code> - the nonexistent revision can be thought of as the one that “existed” before the first revision was submitted to the depot.
<code>file#head</code>	The head revision (latest version) of <i>file</i> . Except where explicitly noted, this is equivalent to referring to the file without a revision specifier.
<code>file#have</code>	The revision on the current client: the revision of file last <code>p4 sync</code> d into the client workspace
<code>file@n</code>	Change number: The revision of <i>file</i> immediately after changelist <i>n</i> was submitted.
<code>file@labelname</code>	Label name: The revision of <i>file</i> in the label <i>labelname</i> .

Modifier	Meaning
<i>file@clientname</i>	Client name: The revision of <i>file</i> last taken into client workspace <i>clientname</i> .
<i>file@datespec</i>	<p>Date and time: The revision of <i>file</i> at the date and time specified.</p> <p>If no time is specified, the head revision at 00:00:00 on the morning of the date specified is returned.</p> <p>Dates are specified <i>yyyy/mm/dd:hh:mm:ss</i> or <i>yyyy/mm/dd hh:mm:ss</i> (with either a space or a colon between the date and the time).</p> <p>The datespec <i>@now</i> may be used as a synonym for the current date and time.</p>

Revision specifiers can be used to operate on many files at once: `p4 sync //myclient/...#4` copies the fourth revision of all non-open files into the client workspace.

If specifying files by date and time (i.e., using specifiers of the form *file@datespec*), the date specification should be parsed by your local shell as a single token. You may need to use quotation marks around the date specification if you use it to specify a time as well as a date.

Some Perforce file specification characters may be intercepted and interpreted by the local shell, and need to be escaped before use. For instance, `#` is used as the comment character in most UNIX shells, and `/` may be interpreted by (non-Perforce) DOS commands as an option specifier. File names with spaces in them may have to be quoted on the command line.

For information on these and other platform-specific issues, see the release notes for your platform.

Using revision ranges

A few Perforce commands can use revision ranges to modify file arguments. Revision ranges are two separate revision specifications, separated by a comma. For example, `p4 changes file#3,5` lists the changelists that submitted file *file* at its third, fourth, and fifth revisions.

Revision ranges have two separate meanings, depending on which command you're using. The two meanings are:

- Run the command on all revisions in the specified range. For example, `p4 jobs //...#20,52` lists all jobs fixed by any changelist that submitted any file at its 20th through 52nd revision.

This interpretation of revision ranges applies to `p4 changes`, `p4 fixes`, `p4 integrate`, `p4 jobs`, and `p4 verify`.

- Run the command on only the highest revision in the specified range. For example, the command `p4 print file@30,50` prints the highest revision of file `file` submitted between changelists 30 and 50. This is different than `p4 print file@50`: if revision 1 of file `file` was submitted in changelist 20, and revision 2 of file `file` was submitted in changelist 60, then `p4 print file@30,50` prints nothing, while `p4 print file@50` prints revision 1 of `file`.

The commands `p4 files`, `p4 print`, and `p4 sync` all use revision ranges in this fashion.

Revision ranges can be very powerful. For example, the command `p4 changes file#3,@labelname` lists all changelists that submitted file `file` between its third revision and the revision stored in label `labelname`.

Limitations on characters in filenames and entities

To support internationalization, Perforce permits the use of “unprintable” (non-ASCII) characters in filenames, label names, client workspace names, and other identifiers.

The pathname component separator (`/`) is not permitted in filenames, depot names, or client workspace names, but may appear in label names, job names, or user names. The recursive subdirectory wildcard (`...`) is not permitted in file names, label names, or other identifiers.

Character	Reason
<code>...</code>	Perforce wildcard: matches anything, works at the current directory level and includes files in all directory levels below the current level.
<code>/</code>	Perforce separator for pathname components.

To refer to files containing the Perforce revision specifier wildcards (@ and #), file matching wildcard (*), or positional substitution wildcard (%) in either the file name or any directory component, use the ASCII expression of the character's hexadecimal value. ASCII expansion applies only to the following four characters:

Character	ASCII expansion
@	%40
#	%23
*	%2A
%	%25

To add a file such as `status@june.txt`, force a literal interpretation of special characters by using:

```
p4 add -f //depot/path/status@june.txt
```

When you submit the changelist, the characters are automatically expanded and appear in the change submission form as follows:

```
//depot/path/status%40june.txt
```

After submitting the changelist with the file's addition, you must use the ASCII expansion in order to sync it to your workspace or edit it within your workspace:

```
p4 sync //depot/path/status%40june.txt
p4 edit //depot/path/status%40june.txt
```

Most special characters tend to be difficult to use in filenames in cross-platform environments: UNIX separates path components with /, while many DOS commands interpret / as a command line switch. Most UNIX shells interpret # as the beginning of a comment. Both DOS and UNIX shells automatically expand * to match multiple files, and the DOS command line uses % to refer to variables.

Similarly, although non-ASCII characters are allowed in filenames and Perforce identifiers, entering these characters from the command line may require platform-specific solutions. Users of GUI-based file managers can manipulate such files with drag-and-drop operations.

Views

Synopsis

There are three types of views: *client views*, *branch views*, and *label views*.

- Client views map files in the depot to files in the client workspace
- Branch views map files in the depot to other parts of the depot
- Label views associate groups of files in the depot with a single label.

Each type of view consists of lines which map files from the depot into the appropriate namespace. For client and branch views, the mappings consist of two file specifications. The left side of the mapping always refers to the depot namespace, and the right side of the mapping refers to the client workspace or depot namespace. For label views, only the left side (the depot namespace) of the mapping need be provided - the files are automatically associated with the desired label.

All views construct a one-to-one mapping between files in the depot and the files in the client workspace, branch, or label. If more than one mapping line refers to the same file(s), the earlier mappings are overridden. Mappings beginning with a hyphen (-) specifically exclude any files that match that mapping. In client views, mappings beginning with a plus sign (+) overlay previous mappings. (Overlay mappings do not apply to branch or label views.)

File specifications within mappings are provided in the usual Perforce syntax, beginning with //, followed by the depot name or workspace name, and followed by the actual file name(s) within the depot or workspace. (You cannot use revision specifiers in views.)

Usage Notes

Views are set up through the `p4 client`, `p4 branch`, or `p4 label` commands as part of the process of creating a client workspace, label view, or branch view respectively.

The order of mappings in a client or branch view is important. For instance, in the view defined by the following two mappings:

```
//depot/... //cws/...
//depot/dir1/... //cws/dir2/...
```

the entire depot is mapped to the client workspace, but the file `//depot/dir1/file.c` is mapped to `//cws/dir2/file.c`. If the order of the lines in the view is reversed, however:

```
//depot/dir1/... //cws/dir2/...
//depot/... //cws/...
```

then the file `//depot/dir1/file.c` is mapped to `//cws/dir1/file.c`, as the first mapping (mapping the file into `//cws/dir2`) is overridden by the second mapping

(which maps the entire depot onto the client workspace). A later mapping in a view always overrides an earlier mapping.

Spaces in path and file names

If a path or file name in a workspace view, branch view, or label view contains spaces, make sure to quote the path:

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

Special characters in path and file names

To map file and directory names that contain the characters @, #, *, or %, (that is, to interpret such characters as components of path and filenames, and *not* as Perforce wildcards), expand the characters to their ASCII equivalents as follows:

Character	ASCII expansion
@	%40
#	%23
*	%2a
%	%25

Client Views

Client views are used to map files in the depot to files in client workspaces, and vice versa. A client workspace is an area in which users perform their work; files are 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/...

Client View	Sample Mapping
Some files in the depot are excluded from the client workspace	//depot/dir1/... //cws/... -//depot/dir1/exclude/... //cws/dir1/...
Files in the client workspace are mapped to different names than their depot names.	//depot/dir1/old.* //cws/renamed/new.*
Portions of filenames in the depot are rearranged in the client workspace	//depot/dir1/%%1.%%2 //cws/dir1/%%2.%%1
The files do not map the same way in each direction. The second line takes precedence, and the first line is ignored.	//depot/dir1/... //cws/build/... //depot/dir2/... //cws/build/...
An overlay mapping is used to map files from more than one depot directory into the same place in the workspace.	//depot/dir1/... //cws/build/... +//depot/dir2/... //cws/build/...

To create a client view, use `p4 client` to bring up a screen where you can specify how files in the depot are mapped to the files in your client workspace.

Branch Views

Branching of the source tree allows multiple sets of files to evolve along different paths. The creation of a branch view allows Perforce to automatically manage the file copying and edit propagation tasks associated with branching.

Branch views map existing areas of the depot (the source files) onto new areas of the depot (the target files). They are defined in a manner similar to that used for defining client views, but rather than mapping files directly into a client workspace, they merely set up mappings within the depot. Because integration can take place in either direction, every line in a branch view must be unambiguous in both directions; overlay mappings are therefore not permitted in branch views.

Branch View	Sample Mapping
New code branching off from the main codeline	//depot/main/... //depot/1.1dev/...
Rearranging directories in the new release	//depot/main/... //depot/1.1dev/... //depot/main/*.c //depot/1.1dev/src/*.c //depot/main/*.txt //depot/1.1dev/doc/*.txt

To create a branch view, use `p4 branch newbranch`. This will bring up a screen (similar to the one associated with `p4 client`) and allow you to map the donor files from the main source tree onto the target files of the new branch.

No files are copied when a branch view is first created. To copy the files, you must ensure that the newly-created files are included in any client workspace view intending to use those files. You can do this by adding the newly-mapped branch of the depot to your current client workspace view and performing a `p4 sync` command.

Label Views

Label views assign a label to a set of files in the depot. Unlike client views and branch views, a label view doesn't copy any files; label views are used to limit the set of files that are taggable by a label. .

Label View	Sample Mapping
A new release	//depot/1.1final/...
The source code for the new release	//depot/1.1final/src/...
A distribution suitable for clients	//depot/1.1final/bin/...
	//depot/1.1final/doc/...
	//depot/1.1final/readme.txt

To create a label, use `p4 label labelname`, and enter the depot side of the view. Because a label is merely a list of files and revision levels, only the depot side (the left side) of the view needs to be specified, and overlay mappings are not permitted.

File Types

Synopsis

Perforce supports six base file types:

- text files,
- compressed binary files,
- native apple files on the Macintosh,
- Mac resource forks,
- symbolic links (`symlinks`), and
- unicode and `utf16` files.

File type modifiers are then applied to the base types allowing for support of RCS keyword expansion, file compression on the server, and more.

When 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 8192 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`. (Files in unicode environments are detected differently; for details, see the *Internationalization Notes*.)

Perforce administrators can use the type mapping feature (`p4 typemap`) to override Perforce's default file type detection mechanism. This feature is useful for `binary` file formats (such as Adobe PDF, or Rich Text Format) where files can start with 8192 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
<code>text</code>	Text file	Treated as text on the client. Line-ending translations are performed automatically on Windows and Macintosh clients.	deltas in RCS format
<code>binary</code>	Non-text file	Accessed as binary files on the client. Stored compressed within the depot.	full file, compressed
<code>symlink</code>	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. For full details, please see the Mac client release notes.	full file, compressed, AppleSingle format.
resource	Macintosh resource fork	The only file type for Mac resource forks in Perforce 99.1 and before. Still supported, but the apple file type is preferred. For full details, please see the Mac client release notes.	full file, compressed
unicode	Unicode file	Perforce servers operating in unicode mode support the unicode file type. These files are translated into the local character set specified by P4CHARSET. Perforce servers not in unicode mode do not support the unicode file type. For details, see the <i>Internationalization Notes</i> .	UTF-8
utf16	Unicode file	If the Perforce server is operating in unicode mode, files are translated into the local character set as specified by P4CHARSET. If the Perforce server is operating in non-unicode mode, files are transferred as UTF-8, and translated to UTF-16 (with byte order mark, in the byte order appropriate for the client machine) in the client workspace. For details, see the <i>Internationalization Notes</i> .	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.
+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: <ul style="list-style-type: none"> • \$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.

Modifier	Description	Comments
+D	Server stores deltas in RCS format	Default server storage mechanism for text files.
+F	Server stores full file per revision, uncompressed	Useful for large binaries, or for long ASCII files that aren't read by users as text, such as PostScript files.
+S	Only the head revision is stored on the server	Older revisions are purged from the depot upon submission of new revisions. Useful for executable or .obj files.
+Sn	Only the most recent <i>n</i> revisions are stored on the server, where <i>n</i> is a number from 1 to 10, or 16, 32, 64, 128, 256, or 512.	Older revisions are purged from the depot upon submission of more than <i>n</i> new revisions, or if you change an existing +Sn file's <i>n</i> to a number less than its current value.
+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`.

Partial filetypes are also acceptable. For example, to change an existing `text` file to `text+x`, use `p4 reopen -t +x script.sh`. Most partial filetype modifiers are added to the filetype, but the storage modifiers (`+C`, `+D`, and `+F`) replace the file's storage method. To remove a modifier, you must specify the full filetype.

Perforce file types for common file extensions

The following table lists recommended Perforce file types and modifiers for common file extensions.

File Type	Perforce file type	Description
.asp	text	Active server page file
.avi	binary+F	Video for Windows file
.bmp	binary	Windows bitmap file
.btr	binary	Btrieve database file
.cnf	text	Conference link file
.css	text	Cascading style sheet file
.doc	binary	Microsoft Word document
.dot	binary	Microsoft Word template
.exp	binary+w	Export file (Microsoft Visual C++)
.gif	binary+F	GIF graphic file
.htm	text	HTML file
.html	text	HTML file
.ico	binary	Icon file
.inc	text	Active Server include file
.ini	text+w	Initial application settings file
.jpg	binary	JPEG graphic file
.js	text	JavaScript language source code file
.lib	binary+w	Library file (several programming languages)
.log	text+w	Log file
.mpg	binary+F	MPEG video file
.pdf	binary	Adobe PDF file
.pdm	text+w	Sybase Power Designer file
.ppt	binary	Microsoft Powerpoint file
.xls	binary	Microsoft Excel file
.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 <i>YYYY/MM/DD</i>	\$Date: 2000/08/18 \$
\$DateTime\$	Date and time of last submission in format <i>YYYY/MM/DD hh:mm:ss</i> Date and time are as of the local time on the Perforce server at time of submission.	\$DateTime: 2000/08/18 23:17:02 \$
\$Change\$	Perforce changelist number under which file was submitted	\$Change: 439 \$
\$File\$	File name only, in depot syntax (without revision number)	\$File: //depot/path/file.txt \$
\$Revision\$	Perforce revision number	\$Revision: #3 \$
\$Author\$	Perforce user submitting the file	\$Author: edk \$

Usage Notes

- The type of an existing file can be determined with `p4 opened` or `p4 files`.
- *Delta storage* (the default mode with `text` files) is a method whereby only the differences (or *deltas*) between revisions of files are stored. *Full file storage* (the default mode with `binary` files) involves the storage of the entire file. The file's type determines whether full file or delta storage is used. Perforce uses RCS format for delta storage.
- Some of the file types are compressed to `gzip` format for storage in the depot. The compression occurs during the submission process, and decompression happens while syncing. The process is transparent to the user; the client workspace always contains the file as it was submitted.

- Symbolic links in non-UNIX client workspaces appear as small text files containing a relative path to the linked file. Editing these files on a non-UNIX client should be done with caution, as submitting them to the depot may result in a symbolic link pointing to a nonexistent file on the UNIX workspace.
- Changing a file's type does not affect earlier revisions stored in the depot.

For instance, changing a file's type by adding the `+Sn` (temporary object) modifier tells Perforce to store only the most recent *n* revisions of the file in the depot. If you change an existing file into a temporary object, subsequent revisions (after the *n*th) will purge the revisions stored after the old head revision, but revisions to the file stored in the depot *before* the `+Sn` modifier was used will remain unaffected. (Syncing to a non-head revision submitted *after* the `+Sn` modifier was used will delete the file from your workspace. Such revisions are displayed as `purge` operations in the output of `p4 filelog`.)

- The `modtime (+m)` modifier is a special case: It is intended for use by developers who need to preserve a file's original timestamp. (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
cxttext	Compressed executable text file	text	+Cx
symlink	Symbolic link	symlink	none
resource	Macintosh resource fork	resource	none
uresource	Uncompressed Macintosh resource fork	resource	+F
ltext	Long text file	text	+F
xltext	Executable long text file	text	+Fx
ubinary	Uncompressed binary file	binary	+F
uxbinary	Uncompressed executable binary file	binary	+Fx
tempobj	Temporary object	binary	+FSw
ctempobj	Temporary object (compressed)	binary	+Sw
xtempobj	Temporary executable object	binary	+FSwx
xunicode	Executable unicode	unicode	+x

Index

Symbols

- #
 - as comment character 112
 - as revision specifier 248
 - in filenames 9, 82, 90, 208, 251
 - in views 254
 - not allowed in passwords 202
- %
 - in filenames 9, 82, 90, 208, 251
 - in views 254
- %%n
 - as wildcard 247
- &
 - as boolean AND 106
- *
 - as wildcard 247
 - as wildcard in job searches 106
 - as wildcard, in p4 users 204
 - as wildcard, in protections table 142
 - in filenames 9, 82, 90, 208, 251
 - in views 254
 - masks out password in p4 user form 202
- +m
 - modification time preservation 196
- ...
 - as wildcard 247
 - wildcard, required with p4 depot 47
 - wildcard, restrictions with p4 add 10
- /
 - as path component separator 247
 - as values separator in job templates 111
- /tmp
 - and TEMP 240
- =, >, , >=
 - as comparison operators 106
- @
 - as revision specifier 248
 - in filenames 9, 82, 90, 208, 251
 - in views 254
- ^
 - as boolean NOT 107
- |
 - as boolean OR 106
- A**
 - access
 - admin
 - 142
 - levels 141
 - limiting by IP address 141
 - superuser 142
 - access level
 - and commands, listing of 144
 - access levels
 - and p4 group 83
 - adding files
 - specifying default file types 9, 195, 257
 - admin access level 142
 - administering Perforce 12
 - administration
 - resetting passwords 137
 - allwrite 33
 - API
 - Perforce and p4 fstat 78
 - .asp files 261
 - atomic changes 174
 - audit trail 213
 - authentication 186
 - .avi files 261
- B**
 - base file types 257
 - batch file
 - and P4MERGE 227
 - BeOS
 - and symbolic links 257
 - binary files 257
 - comparing 58
 - .bmp files 261
 - boolean operators

- and jobviews 106
- branch specifications
 - creating and editing 17
 - listing 20
- branch view 255
 - and `p4 branch` 17
 - and `p4 diff2` 19
 - and `p4 integrate` 95
 - and `p4 sync` 256
 - codeline example 19
 - defined 253
- branches
 - comparing files across 57
- branching 17
- .btr files 261
- C**
- carriage return 34
- change review daemon 142, 147, 165, 167, 202
- changelist numbers
 - highest possible 41
 - pending vs. submitted changelists 41
- changelist submission triggers 188
- changelists
 - and jobs 22, 70
 - creating or editing 21, 26
 - default, and `p4 submit` 174
 - defined 21
 - deleting 22
 - details, describing 51
 - full descriptions, displaying 27
 - jobviews and users 107
 - listing 25, 27
 - listing associated files with `p4 opened` 23
 - listing associated jobs with `p4 fixes` 23
 - listing jobs linked to 73
 - listing with `p4 review` 165
 - meaning of 23
 - moving files between 153
 - moving files between with `p4 reopen` 23
 - numbered 174
 - numbered, changing description of 177
 - numbering of 21
 - pending vs. submitted 174

- pending, listing files in 134
- purpose of 177
- removing files from with `p4 revert` 23
- specifying when adding files 9
- specifying when deleting files 44
- specifying when editing files 62
- specifying when resubmitting 175
- submitting 174
- changes
 - atomic 174
 - conflicting, resolving 155
- changing file type
 - with `-t` 260
- characters
 - allowable in file names 250
- checkpoint 12
- client syntax 247
 - and `p4 files` 68
 - translating 207
- client view 254
 - and `p4 client` 30
 - and `p4 print` 139
 - and `p4 sync` 179
 - defined 253
- client workspace
 - alternate roots 31
 - automatically changing settings for 217
 - comparing files with depot 53
 - creating and editing 30, 209
 - defined 30
 - deleting 33
 - files in, vs. `p4 have` 89
 - listing all 38, 210
 - name of 216
 - options 33
 - populating with depot files 179
 - root 31
 - synchronizing labels with 119
 - using file types to set permissions of files in 259
- client workspace templates 33
- clients
 - and labels 119

- and temporary files 240
- clobber 33, 180
- closing jobs
 - with `p4 submit` 175
- .cnf files 261
- codelines
 - and branch views 19
 - comparing files across 57
- command-line options
 - globally-available 243
- commands
 - controlling access to 141
 - help on 91
 - listed by access level 144
- comments
 - in job templates, and P4Win 112
- comparing
 - binary files 58
 - files 53, 56
- comparison operators
 - and jobviews 106
- compress 33
- compression
 - of files, automatic 262
- COMPUTERNAME
 - default client workspace on Windows 216
- counter
 - limits 41
- counters
 - and `p4 review` 165
 - and review access 147
 - listing 43
 - setting 40
- CR/LF translation 34
 - and `LineEnd` setting 35
- creating
 - branch views 17
 - depot specifications 46
- creating users 200
- `crlf` 34
- cross-platform development
 - line endings 35
- .css files 261
- current directory 239
 - and temporary files on non-UNIX clients 240
- D**
- d flag
 - deleting changelists with 22
- daemons
 - and review access 147
 - change review 142, 147, 165, 167, 202
 - changelist numbers 41
 - tips for creating 192
- default changelist
 - listing open files in 134
- default changelists
 - and `p4 submit` 174
- deleting files 44
- deleting passwords 137
- deleting users 201
- delta storage
 - defined 262
- depot
 - and server root 48
 - comparing files with client workspace 53
 - comparing two revisions of files in 56
 - files, getting from 179
 - how files are stored in 262
 - listing files in 68
 - submitting changes to 174
 - verifying integrity of 205
- depot syntax 247
 - and have list 89
 - and `p4 branch` 17
 - and `p4 print` 139
 - and protections table 142
 - translating 207
- depots
 - creating or editing 46
 - deleting 48
 - empty 10
 - listing 50
 - populating 10
 - remote 46, 48

- remote, and protections 148
- diff chunks
 - and file conflicts 158
- diff program
 - and p4 describe 51
 - and p4 diff 53
 - and p4 diff2 56
 - Perforce internal routine 220
 - third-party, specifying 220
- diffing files 53, 56
- directories
 - and spaces 33
- directories, empty
 - removing on sync 35
- directory
 - current 239
- discarding changes 163
- disk space 172
 - reclaiming 132
- DNS
 - and P4PORT 234
- .doc files 261
- .dot files 261
- E**
- editing
 - branch views 17
 - depot specifications 46
 - files 62
 - user specifications 200
- editor
 - form, commands which use 222
 - form, specifying with P4EDITOR 222
- EDITOR_SIGNATURE
 - and P4EDITOR on Macintosh 222
- empty depots
 - populating 10
- environment variables
 - and Windows registry 169
 - how to set 211
 - overriding with global options 243
 - P4AUDIT 213
 - P4CHARSET 214
 - P4CLIENT 216
 - P4COMMANDCHARSET 215
 - P4CONFIG 217
 - P4DEBUG 219
 - P4DIFF 220
 - P4DIFFUNICODE 221
 - P4EDITOR 222
 - P4HOST 223
 - P4JOURNAL 224
 - P4LANGUAGE 225
 - P4LOG 226
 - P4MERGE 227
 - P4MERGEUNICODE 228
 - P4PAGER 229
 - P4PASSWD 230
 - P4PCACHE 231
 - P4PFSIZE 232
 - P4POPTIONS 233
 - P4PORT 234
 - P4ROOT 235
 - P4TARGET 236
 - P4TICKETS 237
 - P4USER 238
 - PWD 239
 - setting for a Windows service 211
 - setting with P4CONFIG 217
 - TMP, TEMP 240
- example
 - branching and codelines 19
 - changing file types 154
 - comparing files across a branch 58
 - creating a job 104
 - deleting a user 203
 - editing a job 104
 - editing user information 203
 - effects of protections 148
 - generating output for scripts 82
 - getting files from depot 181
 - integrating files 98
 - listing jobs by various criteria 108
 - listing opened files 135
 - moving files between changelists 154
 - p4 typemap 197
 - pending changelist, listing files in 135

- pipes and -x 55
 - pre-submit triggers, use of 193
 - propagating changes 98
 - protections table 148
 - RCS keyword expansion 262
 - renaming files 152
 - reverting files to pre-opened states 164
 - scheduling a resolve 98
 - submitting files in changelists 178
 - syncing a client workspace 181
 - viewing user information 203
 - working as another user 203
- exclusionary mappings 253
 - and p4 protect 142
 - and triggers 187
- .exp files 261
- external authentication 186
- F**
- f flag
 - editing previously-submitted changelists 22
 - editing read-only job fields with 103
 - forcing label deletion with 115
 - overriding client workspace settings 33
- fields
 - null, in jobs 108
- file names
 - valid characters for 250
 - with spaces, in views 254
 - with spaces, on command line 249
- file size 172
- file specifications
 - and p4 revert 164
 - and p4 submit 177
 - help on 91
 - interpreted by local shell 249
- file types 257, 261
 - and p4 add 10
 - and p4 edit 62
 - and permissions in client workspace 259
 - and storage in depot 262
 - apple 258
 - base 257
 - binary 257
 - changing 153
 - determined by Perforce 257
 - help on 91
 - keywords 264
 - listed 264
 - mapping to filenames 195
 - modifiers 259
 - partial 260
 - resource 258
 - showing 262
 - specifying 259
 - specifying with -t 260
 - symlink 257
 - text 257
- filenames
 - and spaces 33
 - mapping to file types 195
 - special characters 9, 82, 90, 208, 251, 254
- files
 - adding to depot 9
 - adding to label 119
 - adding, specifying default type 9, 195, 257
 - .asp 261
 - .avi 261
 - binary, comparing 58
 - .bmp 261
 - .btr 261
 - changing type 153
 - changing type with -t 260
 - checkpoints and journals 12
 - .cnf 261
 - comparing 53, 56
 - comparing between codelines 57
 - conflicts between, resolving 155
 - controlling access 141
 - copying from depot 179
 - .css 261
 - deleting from depot 44
 - deleting from label 119, 183
 - deleting permanently 131
 - delta and full-file storage 262
 - displaying info for scripts 78

- displaying revision histories 65
- .doc 261
- .dot 261
- editing 62
- editing older revisions 63
- .exp 261
- getting from depot 179
- getting latest revision 248
- .gif 261
- .htm 261
- .html 261
- .ico 261
- in a label, listing 117
- in changelists, detailed information 51
- .inc 261
- including in labels 114
- .ini 261
- integrated, listing 100
- integrating changes between 155
- .jpg 261
- .js 261
- .lib 261
- linked to changelist, listing 23
- listing 68
- listing contents of, by revision 139
- listing open files 134
- locating 207
- locked 135
- locking 122
- .log 261
- mapping Perforce file types to filenames 195
- modification time, preserving 196
- moving between changelists 23, 153
- .mpg 261
- multi-forked 258
- obliterating 131
- on other depots, accessing 46
- open, discarding changes 163
- open, listing 134
- open, submitting 174
- opening 32, 174, 176
- opening for add 9

- opening for branch with `p4 integrate` 94
- opening for delete 44
- opening for delete with `p4 integrate` 94
- opening for edit 62
- opening for integrate 94
- .pdf 261
- .pdm 261
- permanent removal of 131
- .ppt 261
- preventing other users from editing 122
- removing from changelists 23, 163
- removing with `#none` 248
- renaming 152
- reopening 23
- resolving conflicts between 155
- reverting 23, 32, 174, 176
- reverting to pre-edit state 163
- saving changes to depot 174
- scheduled for resolve, listing 161
- scheduling for resolve 160
- specifying 247
- specifying by change number 248
- specifying by date and time 249
- specifying by revision 248
- specifying type of 259
- stored compressed 262
- submitting 174
- syncing 179
- tagging 183
- types of 257
- unlocking 199
- unresolved, listing 161
- verifying integrity of 205
- .xls 261
- yours, theirs, base, merge*, meaning when resolving 156
- .zip 261

fixes

- deleting fix records with `p4 fix -d` 70
- listing 73
- to jobs over multiple changelists 70

forms
 commands which use 222
 specifying editor with `P4EDITOR` 222

full file storage
 defined 262

G

-G option 243

`getcwd()`

 in lieu of `PWD` 239

getting files from depot 179

.gif files 261

global options 243

 help on 91

groups

 and subgroups 84

 controlling access 141

 creating 83

 deleting 83

 listing users in 87

gzip 262

H

have list

 and `p4 delete` 44

 defined 89

 listing with `p4 have` 89

 vs. files in workspace 89

have revision 89, 248

head revision

 and `p4 delete` 44

 and `p4 edit` 62

 specifying 248

help

 use `p4 help` 91

history of changes to forms 46

hosts file

 and `P4PORT` 234

hosts, impersonating

 impersonating hosts 223

.htm files 261

.html files 261

I

-i flag

 changelists and integrated files 28

.ico files 261

.inc files 261

.ini files 261

integrate

 files, opening for 94

integration

 listing 100

 scheduling 155

IP addresses

 controlling access by 141

J

-J option

 and `p4d` 224

job specification

 displaying 107

job table

 reindexing 106

job templates

 comments in, and `P4Win` 112

job views

 help on 91

jobs

 * wildcard 106

 and changelists 22

 changing status of 71

 closing with `p4 submit` 175

 creating and editing 102

 defined 102

 excluding from query 108

 fixing over multiple changelists 70

 linked to changelist, showing 23

 linked to changelists, listing 73

 linking to changelists with `p4 fix` 70

 listing 105

 null fields 108

 wildcards 108

jobs template

 modifying 110

JobView field

 and `p4 user form` 107

 use of 107

Jobview field

 and changelists 22

- and p4 user 202
- jobviews
 - and comparison operators 107
 - and field types 107
 - limitations 108
 - searching jobs 105
- journal 12
- journal file
 - specifying with P4JOURNAL 224
- .jpg files 261
- .js files 261
- K**
- keywords
 - RCS, examples 262
 - RCS, expanding 259
 - specifying Perforce file types 264
- L**
- L flag
 - and long change descriptions 28, 66
- l flag
 - and long change descriptions 27, 28, 66
 - and long job descriptions 105
- L option
 - and p4d 226
- label 183
 - adding files to 119
 - deleting files from 119, 183
 - listing files in 117
 - unlocking 115
- label view 256
 - defined 253
- labels
 - and clients 119
 - listing 117
 - owner of, changing 114, 119
 - synchronizing with clients 119
- labelsync
 - ownership required 114, 119
- latest revision
 - specifying 248
- LDAP 186
- .lib files 261
- licence
 - and pre-submit triggers 192
- license
 - and remote virtual user 48
- limitations
 - and jobviews 108
- line endings 35
- LineEnd 35
 - CR/LF 32
- linefeed convention 34
- list access level 141
- listing
 - branches 20
 - changelists 25, 27
 - client workspaces 38, 210
 - counters 43
 - depots 50
 - file contents by revision 139
 - file integrations 100
 - files in a label 117
 - files in depot 68
 - files scheduled for resolve 161
 - fixes 73
 - groups 87
 - jobs 105
 - jobs linked to changelists 73
 - labels 117
 - open files 134
- listing subdirectories 60
- listing users 204
- local syntax 247
 - and have list 89
 - translating 207
- locked 34
- locked files
 - finding 135
- locking files 122
- .log files 261
- logging 213
- M**
- Macintosh
 - and file types 258
 - changing default form editor 222
 - linefeed convention 34

- resource fork file type 258
- mappings
 - and p4 client 30
 - and protections table 142
 - directories with spaces 33
 - exclusionary 253
 - exclusionary, and protections table 142
 - exclusionary, and triggers 187
 - in branch views 17, 255
 - in client views 254
 - in label views 115, 256
 - integration, and p4 branch 95
 - local and remote depots 47
 - overlay 253
- mappings, order of
 - and triggers 187
 - in protections 142
 - in views 253
- maxlocktime
 - commands affected by 85
- maxresults
 - and p4 filelog 66
 - and p4 files 69
 - and p4 print 140
 - commands affected by 85
 - setting with p4 group 83
- maxscanrows
 - commands affected by 85
 - setting with p4 group 83
- MD5
 - and p4 verify 205
 - and passwords 136, 230
- MERGE environment variable
 - and P4MERGE 227
- merge programs
 - third-party, specifying 227
- modifier
 - file type, +m 196
- modtime 34
 - changes as of 2000.1 34
- .mpg files 261
- multi-forked file 258

N

- network
 - data compression 33
- noallwrite 33
- noclobber 33, 180
- nocompress 33
- nocrlf 34
- nomodtime 34
 - changes as of 2000.1 34
- nonexistent revision
 - specifying 248
- normdir 35
- numbered changelists 174

O

- obliterating files 131
- online help
 - use p4 help 91
- open access level 141
- open files
 - changing type with p4 reopen 153
- opening files
 - for add 9
 - for delete 44
 - for edit 62
- openlink fix.fm
 - firstpage 186
- operators
 - boolean, and jobviews 106
 - comparison, and jobviews 106
- options
 - for client workspaces 33
 - global 243
- output
 - formatting for scripts with -s 243
- overlay mappings 253
- overriding
 - registry variable settings 170
- owner
 - of label, changing 114, 119

P

- p4
 - version of 243
- p4 add 9

- p4 admin 12
- p4 branch 17
 - and p4 integrate 95
- p4 branches 20
- p4 change 21
- p4 changelist 26
- p4 changelists 25
- p4 changes 27
- p4 client 30
 - options, and p4 sync 180
- p4 clients 38
- p4 counter 40
- p4 counters 43
- p4 delete 44
 - vs. p4 obliterate 131
- p4 depot 46
- p4 depots 50
- p4 describe 51
- p4 diff 53
 - and P4DIFF 220
- p4 diff2 56
 - and branch views 19
- p4 dirs 60
- p4 edit 62
- p4 executable
 - version of 93
- p4 filelog 65
- p4 files 68
- p4 fix 70
- p4 fixes
 - and changelists 23
- p4 flush 75
- p4 fstat 78
- p4 group 83
- p4 groups 87
- p4 have 89
 - vs. files in workspace 89
- p4 help 91
- p4 info 93
- p4 integ
 - abbreviation for p4 integrate 97
- p4 integrate 94
- p4 integrated 100
- p4 job 102
- p4 jobs 105
- p4 jobspec 110
 - and P4Win 112
- p4 labels 117
- p4 labelsync 119
 - and p4 label 114
- p4 license 121
- p4 lock 122
- p4 logger 123
- p4 login 124
- p4 logout 126
- p4 monitor 128
- p4 obliterate 131
 - and deleting depots 48
- p4 open 63
- p4 opened 134
 - and changelists 23
- p4 passwd 136
 - and P4PASSWD 230
 - setting passwords with 230
- p4 print 139
- p4 protect 141
 - and Protections field 142
 - required after server installation 147
 - required when creating new depots 48
- p4 protects 150
- p4 rename 152
- p4 reopen 153
 - and changelists 23
- p4 resolve 155
 - and P4DIFF 220
 - and P4MERGE 227
 - and P4PAGER 229
- p4 resolved 161
- p4 revert 163
 - and changelists 23
 - and p4 resolve -at 157
- p4 review 165
- p4 reviews 167
- p4 set 169
- p4 sizes 172
- p4 submit 174

- p4 sync 179
 - and branch view 256
- p4 tag 183
- p4 tickets 185
- p4 triggers 186
- p4 typemap 195, 257
 - and p4 add 9
- p4 unlock 199
- p4 user 200
 - and JobView field 107
 - and Reviews field 167
 - jobviews, and p4 submit 175
 - setting passwords with 230
 - specifying username with 238
- p4 users 204
- p4 verify 205
- p4 where 207
- p4 workspace 209
- p4 workspaces 210
- P4CHARSET 214
- P4CLIENT 216
- P4COMMANDCHARSET 215
- P4CONFIG 217
- p4d
 - logging errors to a file 226
 - specifying journal file 224
- P4DEBUG 219
- P4DIFF 220
 - and p4 diff 53
 - not used in p4 describe 51
 - not used in p4 diff2 56
- P4DIFFUNICODE 221
- P4EDITOR 222
 - commands affected by 222
- P4HOST 223
- P4JOURNAL 224
- P4LANGUAGE 225
- P4LOG 226
- P4MERGE 157, 227
 - batch file required on Windows 227
- P4MERGEUNICODE 228
- P4PAGER 229
- P4PASSWD 230
 - and p4 passwd 230
- P4PCACHE 231
- P4PFSIZE 232
- P4POPTIONS 233
- P4PORT 234
- P4ROOT 235
 - and depot files 48
 - and temporary files on Windows servers 240
- P4TARGET 236
- P4TICKETS 237
- P4USER 238
 - and pre-submit triggers on Windows 193
- P4Win
 - and comments in job templates 112
 - tooltips and jobspecs 112
- PAGER environment variable
 - and P4PAGER 229
- password
 - maximum length of 137
- passwords
 - and P4PASSWD 230
 - and users 202, 238
 - deleting 137
 - resetting 137
 - setting 136
 - special characters in 202
 - specifying on command line 136, 238
- .pdf files 261
- .pdm files 261
- pending changelists 174
 - editing description of 21
 - listing 25, 27
 - listing files in 134
- Perforce API
 - and p4 fstat 78
- Perforce client
 - and P4PORT 234
 - and temporary files 240
- Perforce client and server
 - obtaining version of 93
- Perforce file types 261
- Perforce Proxy

- and P4PCACHE 231
- and P4PFSIZE 232
- and P4OPTIONS 233
- and P4PORT 234
- and P4TARGET 236
- Perforce server
 - administering 12
 - and P4PORT 234
 - and P4ROOT 235
 - and temporary files 240
 - and triggers 190
 - checkpoints and journals 12
 - installing securely 147
 - stopping 12
 - verifying integrity of 205
- Perforce syntax 247
- Perforce usernames
 - and passwords 238
- permissions
 - files, and `p4 edit` 62
 - granting and denying 141
 - required before accessing new depot 48
 - setting in client workspace via file type 259
- populating depots 10
- port number
 - setting, on clients and servers 234
- positional specifiers 247
- POSIX\$SHELL
 - and P4EDITOR on VMS 222
- .ppt files 261
- preserving modification times 196
- pre-submit triggers 186
 - tips for creating scripts 192
- protections
 - and IP addresses 141
 - granting and denying 141
 - viewing 150
- Protections field 142
- protections table 141
 - example 148
- proxy
 - and P4PCACHE 231
 - and P4PFSIZE 232
 - and P4OPTIONS 233
 - and P4PORT 234
 - and P4TARGET 236
- PWD 239
- Python 243
- R**
- RCS file format 262
- RCS keyword expansion 259
 - examples 262
- read access level 141
- registry
 - never stores plaintext passwords 136, 230
 - setting variables in 169
- registry variables
 - overriding settings of 170
- remote depots 46, 48
 - and protections 148
- removing files
 - permanently 131
- renaming files 152
- resetting passwords 137
- resolve
 - scheduling files for 160
- resolving files 155
- resource fork 258
- reverting changes 23, 163
- review access level 142
- Reviews field
 - and `p4 user` 167
 - use of 202
- revision
 - latest, specifying 248
 - of file on current client 248
 - of file, displaying 139
 - specifying 248
- revision history
 - displaying 65
 - obliterating 131
- revision ranges
 - and `p4 changes` 27
 - and `p4 files` 69
 - and `p4 fixes` 73

- and p4 integrate 94
 - and p4 print 139, 140
 - and p4 resolved 161
 - and p4 sync 179
 - specifying 249
- revision specifiers 248
 - and labels 119
 - and p4 changes 27
 - and p4 sync 179
 - help on 91
- rmdir 35
- S**
- s option
 - and p4 fstat 81
 - formatting output for scripting 243
- scripting
 - and p4 dirs 60
 - and p4 fstat 78
 - and -s option 243
 - and triggers 186
 - and -x option 243
 - s and p4 fstat 81
 - triggers, tips for creating scripts 192
 - with Python 243
 - x option, example 55
- searching
 - for null job fields 108
 - jobs, with jobviews 105
- security
 - and p4 protect 147
- security level 202
- server
 - administering 12
 - and P4PORT 234
 - and temporary files 240
 - and triggers 190
 - changing IP address 121
 - checkpoints and journals 12
 - installation, and p4 protect 147
 - licensing 121
 - reclaiming disk space 132
 - specifying error log file 226
 - specifying journal file 224
 - stopping 12
 - upgrading 106
 - verifying integrity of 205
- server root 235
 - and depots 48
 - and temporary files on Windows servers 240
- server variables
 - listing 43
 - setting 40
- setting environment variables 211
 - for Windows services 169
 - on Windows services 211
- shell
 - interpreting file specifications 204, 249
- SHELL environment variable
 - and P4DIFF on Windows 220
 - and P4EDITOR on Windows 222
- spaces
 - within filenames 33
- spaces and client workspaces
 - translated to underscores 33
- spaces in file names
 - quotes around 249
- spaces in filenames
 - quotes around, in views 254
- spaces in passwords
 - quotes around 137
- spec depot
 - populating 12
- spec depot 46
- specification
 - job, displaying 107
- specification triggers 188
- specifiers
 - positional 247
 - revision 248
- specifying
 - default editor with P4EDITOR 222
 - file types 259
 - files for integration 94
 - files, by change number 248
 - files, by date and time 249

- files, by revision 248
- files, for integration 94
- files, latest version of 248
- program to display `p4 resolve` output 229
- revision ranges 249
- third-party diff programs 220
- third-party merge programs 227
- username with `-u` and `P4USER` 238
- standard input
 - reading from 243
- standard output
 - and `p4 print` 139
- status
 - of jobs, changing 71
- Status field
 - and `p4 submit` 174
- storage
 - of files in depot 262
- subdirectories
 - listing 60
- subgroups
 - and groups 84
- submit
 - reverting files 32, 174, 176
- submitted changelists 174
 - listing 25, 27
 - viewing 21
- submitting changelists 174
- submitting files 174
- super access level 142
- superuser 142
 - and creating users 200
 - and new server 147
- symbolic links 257
 - on non-UNIX systems 257, 263
- sync 179
- syntax forms
 - local, client, depot 247
 - translating between with `p4 where` 207
- T**
 - `-t` flag
 - and client workspace templates 33
 - and file type 260
- tag 183
- target server
 - and Perforce Proxy 236
- template
 - jobs, modifying 110
- templates
 - client workspace 33
- temporary files
 - where stored 240
- text files 257
- ticket file
 - location 237
- timestamps
 - on DLLs, preserving 36, 263
- TMP, TEMP 240
- tooltips 112
- translation
 - CR/LF 34
- triggers 186
 - and Windows services 193
 - naming 187
 - passing arguments to 191
 - script, specifying arguments to 190
 - types of 188
- troubleshooting
 - local shell and file specifications 249
- type mapping 195
- typemap 9
- types
 - of files, changing 153
- U**
 - `-u` flag
 - impersonating users with 238
 - unchanged files
 - reverting 32, 174, 176
 - undoing file edits 163
 - unicode 214, 215, 221, 228, 258
 - UNIX
 - line endings on mounted drives 35
 - UNIX
 - linefeed convention 34, 35
 - unlocked 34

- unlocking files 199
- unresolved files
 - listing 161
- upgrading
 - from 98.2 or earlier 106
- USER
 - and P4USER 238
- user preferences
 - setting 200
- USERNAME
 - and P4USER on Windows 238
- users
 - and files, unlocking 199
 - and forgotten passwords 137
 - and groups 83
 - and P4PASSWD 230
 - and passwords 136, 202, 238
 - changing with P4CONFIG and P4USER 200
 - controlling access 141
 - creating and editing 200
 - deleting 201
 - groups of, listing 87
 - groups, granting access to 141
 - listing 204
 - listing with p4 reviews 167
 - preventing others from editing files 122
 - running commands as 202, 238
 - virtual, remote 48, 148
- UTF-16 214, 215
- UTF-32 214, 215
- UTF-8 214
- V**
- variables
 - environment, how to set 211
 - overriding with global options 243
 - registry 169
 - server, listing 43
 - server, setting 40
- verifying file integrity 205
- version
 - of p4 243
 - of Perforce client and server programs 93
- versioned specifications 46
- view
 - branch 255
 - branch, and p4 diff2 19
 - branch, and p4 integrate 95
 - branch, and p4 sync 256
 - branch, creating or editing 17
 - client 254
 - client, and p4 sync 179
 - help on 91
 - introduced 253
 - label 256
- VMS
 - changing default form editor 222
- W**
- warnings
 - about counters and p4 review 166
 - about p4 counters 40
 - about p4 flush 75
 - about p4 jobspec 103
 - about p4 obliterate 131
 - about p4 revert 164
 - about pre-submit triggers 186
 - superuser access and p4 protect 147
- wildcards
 - and p4 add 10
 - and p4 integrate 94
 - in jobviews 106
 - listing users with 204
 - specifying files with 247
- Windows
 - batch file required for P4MERGE 227
 - COMPUTERNAME as default client work-space 216
 - default client workspace name 216
 - default forms editor 222
 - line endings 35
 - linefeed convention 34, 35
 - overriding registry variables 170
 - registry variables 169
 - services, and triggers 193
 - setting passwords on 230
 - setting variables for Windows services 211

- third-party DLLs 36, 263
- workspace
 - client, alternate roots 31
 - client, creating and editing 30, 209
 - client, listing 38, 210
 - files in, vs. have list 89
- write access level 141
- X**
 - x option
 - example with `p4 diff` 55
 - reading from standard input 243
- .xls files 261
- Z**
 - .zip files 261