PERFORCE

P4 Command Reference

2016.2

October 2016

P4 Command Reference
2016.2

October 2016

Copyright © 1999-2016 Perforce Software.

All rights reserved.

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

This product is subject to U.S. export control laws and regulations including, but not limited to, the U.S. Export Administration Regulations, the International Traffic in Arms Regulation requirements, and all applicable end-use, end-user and destination restrictions. Licensee shall not permit, directly or indirectly, use of any Perforce technology in or by any U.S. embargoed country or otherwise in violation of any U.S. export control laws and regulations.

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 might 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.

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

Any additional software included within Perforce software is listed in License Statements on page 621.

Table of Contents

About This Manual xxi

What's new in this guide for the 2016.2 release xxii
New and changed configurables and environment variables xxii
New specification fields xxii
New commands xxii
New command options and other command changes xxii
Helix documentation xxiii
Syntax conventions xxiii
Please give us feedback xxiv

Introduction 1

Getting help 1
Command aliases 2
Defining aliases 2
Command alias syntax 3
Basic examples 5
Complex examples 6
Putting it all together 8
Previewing alias substitutions 8
Advanced topics 9
Editing specifications 9
Creating alias files for each workspace 9
Limitations 9
Naming conventions 10
Creating scripts 11

p4 add 13

p4 admin 17

p4 aliases 21

p4 annotate 23

p4 archive 27

p4 attribute 29

p4 branch 31

p4 branches 35

p4 cachepurge 37

p4 change 39

p4 changelist 45

p4 changelists 47

p4 changes 49

p4 clean 53

p4 client 55

p4 clients 67

p4 clone 69

p4 configure 71

p4 copy 75

p4 counter 79

p4 counters 83

p4 cstat 85

p4 dbschema 87

p4 dbstat 89

p4 dbverify 91

p4 delete 93

p4 depot 97

p4 depots 105

p4 describe 107

p4 diff 109

p4 diff2 113

p4 dirs 117

p4 diskspace 119

p4 edit 121

p4 export 125

p4 fetch 129

p4 filelog 133

p4 files 137

p4 fix 141

p4 fixes 145

p4 flush 147

p4 fstat 151

p4 grep 161

p4 group 163

p4 groups 169

p4 have 171

p4 help 173

p4 ignores 175

p4 info 179

p4 init 181

p4 integrate 183

p4 integrated 189

p4 interchanges 193

p4 istat 195

p4 job 197

p4 jobs 201

p4 jobspec 207

p4 journalcopy 211

p4 journaldbchecksums 215

p4 journals 219

p4 key 223

p4 keys 225

p4 label 227

p4 labels 231

p4 labelsync 233

p4 ldap 235

p4 ldaps 241

p4 ldapsync 245

p4 license 247

p4 list 249

p4 lock 251

p4 lockstat 253

p4 logappend 255

p4 logger 257

p4 login 259

p4 logout 261

p4 logparse 263

p4 logrotate 265

p4 logschema 267

p4 logstat 269

p4 logtail 271

p4 merge 273

p4 monitor 277

p4 move 283

p4 obliterate 285

p4 opened 289

p4 passwd 293

p4 ping 297

p4 populate 299

p4 print 301

p4 property 303

p4 protect 305

p4 protects 319

p4 proxy 321

p4 prune 323

p4 pull 325

p4 push 331

p4 reconcile 335

p4 reload 339

p4 remote 341

p4 remotes 345

p4 rename 347

p4 renameuser 349

p4 reopen 353

p4 replicate 355

p4 reshelve 357

p4 resolve 359

p4 resolved 367

p4 restore 369

p4 resubmit 371

p4 revert 373

p4 review 377

p4 reviews 379

p4 server 381

p4 serverid 387

p4 servers 389

p4 set 395

p4 shelve 399

p4 sizes 403

p4 status 405

p4 stream 407

p4 streams 415

p4 submit 417

p4 switch 425

p4 sync 427

p4 tag 435

p4 tickets 437

p4 triggers 439

p4 trust 441

p4 typemap 443

p4 undo 447

p4 unload 449

p4 unlock 453

p4 unshelve 455

p4 unsubmit 457

p4 unzip 459

p4 update 463

p4 user 465

p4 users 471

p4 verify 473

p4 where 477

p4 workspace 479

p4 workspaces 481

p4 zip 483

Environment and Registry Variables 485

P4ALIASES 487

P4AUDIT 489

P4AUTH 491

P4BROKEROPTIONS 493

P4CHANGE 495

P4CHARSET 497

P4_port_CHARSET 499

P4CLIENT 501

P4CLIENTPATH 503

P4COMMANDCHARSET 505

P4CONFIG 507

P4DEBUG 509

P4DESCRIPTION 511

P4DIFF 513

P4DIFFUNICODE 515

P4EDITOR 517

P4ENVIRO 519

P4HOST 521

P4IGNORE 523

P4JOURNAL 525

P4LANGUAGE 527

P4LOG 529

P4LOGINSSO 531

P4MERGE 533

P4MERGEUNICODE 535

P4NAME 537

P4PAGER 539

P4PASSWD 541

P4PCACHE 543

P4PFSIZE 545

P4POPTIONS 547

P4PORT 549

P4ROOT 553

P4SSLDIR 555

P4TARGET 557

P4TICKETS 559

P4TRUST 561

P4USER 563

PWD 565

TMP, TEMP 567

Additional Information 569

Global Options 571

File Specifications 575

Synopsis 575
Syntax forms 575
Wildcards 575
Using revision specifiers 576
Using revision ranges 577
Limitations on characters in filenames and entities 578

Views 581

Synopsis 581
Usage Notes 581
Spaces in path and file names 582
Special characters in path and file names 582
Client Views 582
Branch Views 583
Label Views 584

File Types 585

Synopsis 585
Base filetypes 585
File type modifiers 587
Perforce file types for common file extensions 589
Keyword Expansion 590
Usage Notes 591

Configurables 595

Configurables that affect the server 595
Configurables that affect the client 595
Configurables that affect the proxy 596
Configurables 596

License Statements 621

About This Manual

This manual documents every Perforce command, environment variable, and configurable. 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. This manual is divided into the following parts:

- The first part lists commands in alphabetical order. The table below provides a function-based grouping of this same set of commands.
- The second part lists environment and registry variables in alphabetical order.
- The third part contains miscellaneous reference information about global options, file specifications, views, file types, and configurables.

The following table provides an index to the commands documented in this manual by functional area:

Function	Where to look
Help	**p4 help**, **p4 info**, "File Specifications" on page 575, "Views" on page 581, "Global Options" on page 571, "File Types" on page 585
Client workspace	**p4 clean**, **p4 client**, **p4 clients**, **p4 flush**, **p4 have**, **p4 ignores**, **p4 sync**, **p4 update**, **p4 where**, **p4 workspace**, **p4 workspaces**
Files	**p4 add**, **p4 attribute**, **p4 copy**, **p4 delete**, **p4 diff**, **p4 diff2**, **p4 dirs**, **p4 edit**, **p4 files**, **p4 fstat**, **p4 grep**, **p4 move**, **p4 lock**, **p4 print**, **p4 reconcile**, **p4 rename**, **p4 revert**, **p4 status**, **p4 sizes**, **p4 unlock**
Changelists	**p4 change**, **p4 changelist**, **p4 changes**, **p4 changelists**, **p4 describe**, **p4 filelog**, **p4 opened**, **p4 reopen**, **p4 review**, **p4 shelve**, **p4 submit**, **p4 undo**, **p4 unshelve**
Jobs	**p4 fix**, **p4 fixes**, **p4 job**, **p4 jobs**, **p4 jobspec**
Branching and Merging	**p4 branch**, **p4 branches**, **p4 copy**, **p4 cstat**, **p4 integrate**, **p4 integrated**, **p4 interchanges**, **p4 istat**, **p4 label**, **p4 labels**, **p4 labelsync**, **p4 list**, **p4 merge**, **p4 populate**, **p4 tag**, **p4 resolve**, **p4 resolved**, **p4 stream**, **p4 streams**
Administration	**p4 admin**, **p4 archive**, **p4 cachepurge**, **p4 configure**, **p4 counter**, **p4 counters**, **p4 dbschema**, **p4 dbstat**, **p4 depot**, **p4 depots**, **p4 diskspace**, **p4 journals**, **p4 key**, **p4 keys**, **p4 license**, **p4 lockstat**, **p4 logappend**, **p4 logger**, **p4 logparse**, **p4 logrotate**, **p4 logschema**, **p4 logstat**, **p4 logtail**, **p4 monitor**, **p4 obliterate**, **p4 ping**, **p4 property**, **p4 proxy**, **p4 pull**, **p4 reload**, **p4 renameuser**, **p4 replicate**, **p4 restore**, **p4 reviews**, **p4 server**, **p4 serverid**, **p4 servers**, **p4 triggers**, **p4 typemap**, **p4 unload**, **p4 verify**

Function	Where to look
Security	**p4 group**, **p4 groups**, **p4 login**, **p4 logout**, **p4 passwd**, **p4 protect**, **p4 protects**, **p4 tickets**, **p4 trust**, **p4 user**, **p4 users**, P4CLIENTPATH, P4SSLDIR, P4TRUST
Environment	**p4 set**, "Environment and Registry Variables" on page 485, P4AUDIT, P4AUTH, P4BROKEROPTIONS, P4CHANGE, P4CHARSET, P4COMMANDCHARSET, P4CLIENT, P4CONFIG, P4DEBUG, P4DIFF, P4DIFFUNICODE, P4EDITOR, P4HOST, P4IGNORE, P4JOURNAL, P4LANGUAGE, P4LOG, P4MERGE, P4MERGEUNICODE, P4NAME, P4PAGER, P4PASSWD, P4PCACHE, P4PFSIZE, P4POPTIONS, P4PORT, P4ROOT, P4TARGET, P4TICKETS, P4USER, PWD, TMP, TEMP

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

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.

What's new in this guide for the 2016.2 release

This section provides a summary of the notable changes in this guide for the 2016.2 release. For a list of all new functionality and major bug fixes in Helix Versioning Engine 2016.2, see the *Release Notes*.

New and changed configurables and environment variables

New configurables are described in "Configurables" on page 595.

server.locks.global Setting the new **server.locks.global** configurable to **1** makes running **p4 lock** from an edge server take global locks on the commit server by default.

New specification fields

p4 client The **Type** field in the **p4 client** command's fields can now take a value of **partitioned**, which has the same effect as the existing **readonly** value except that you can edit and submit using this client.

New commands

p4 undo Undo a range of revisions.

New command options and other command changes

p4 client New **-T** option allows you to specify the type of client to create.

p4 protect You can now grant access to the **p4 protect** command for a particular path, to a user or group.

Helix documentation

The following table lists and describes key documents for Helix users, developers, and administrators. For complete information see the following:

http://www.perforce.com/documentation

For specific information about…	See this documentation…
Introduction to version control concepts and workflows; Helix architecture, and related products.	*Introducing Helix*
Using the command-line interface to perform software version management and codeline management; working with Helix streams; jobs, reporting, scripting, and more.	*Helix Versioning Engine User Guide*
Basic workflows using P4V, the cross-platform Helix desktop client.	*P4V User Guide*
Working with personal and shared servers and understanding the distributed versioning features of the Helix Versioning engine.	*Using Helix for Distributed Versioning*
p4 command line (reference).	*P4 Command Reference*, **p4 help**
Installing and administering the Helix versioning engine, including user management, security settings.	*Helix Versioning Engine Administrator Guide: Fundamentals*
Installing and configuring Helix servers (proxies, replicas, and edge servers) in a distributed environment.	*Helix Versioning Engine Administrator Guide: Multi-site Deployment*
Helix plug-ins and integrations.	IDEs: *Using IDE Plug-ins* Defect trackers: *Defect Tracking Gateway Guide* Others: online help from the Helix menu or web site
Developing custom Helix applications using the Helix C/C++ API.	*C/C++ API User Guide*
Working with Helix in Ruby, Perl, Python, and PHP.	*APIs for Scripting*

Syntax conventions

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

Notation	Meaning
`literal`	Monospace font indicates a word or other notation that must be used in the command exactly as shown.
italics	Italics indicate a parameter for which you must supply specific information. For example, for a *serverid* parameter, you must supply the id of the server.
[`-f`]	Square brackets indicate that the enclosed elements are optional. Omit the brackets when you compose the command. Elements that are not bracketed are required.
…	Ellipses (…) indicate that the preceding element can be repeated as often as needed.
element1 | *element2*	A vertical bar (|) indicates that either *element1* or *element2* is required.

Please give us feedback

We are interested in receiving opinions on this manual from our users. In particular, we'd like to hear from users who have never used Perforce before. Does this guide teach the topic well? Please let us know what you think; we can be reached at manual@perforce.com.

If you need assistance, or wish to provide feedback about any of our products, contact support@perforce.com.

Introduction

This book contains reference material for Perforce users and administrators. This section provides information about topics that might be useful to you no matter which command or variable you're interested in. It contains the following subsections:

- Getting help for commands and environment variables
- Defining command aliases
- Naming conventions
- Creating scripts

Getting help

In addition to the material provided in this manual, you can get help for Perforce commands by using the **p4 help** command, which provides help for individual commands or for areas of interest like jobs, revisions, or file types.

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

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

You can now use this format:

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

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

The effect of the command is the same whether you use short options names or long option names. Options that are rarely used have only a short form.

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

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

This will generate output like the following:

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

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

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

Command aliases

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

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

This section covers the following topics:

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

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

Defining aliases

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

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

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

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

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

   Refer to the following sections for information about the syntax of alias definitions and for examples of alias definitions.

   - The file may contain blank lines; they will be ignored.

- The file may also contain comments: the first non-blank character in a comment line must be the **#** character.

  You can edit this file as often as you like to add, modify, or delete definitions.

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

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

   The output to the alias command will show you the command or commands that would be run without actually running the command. For more information, see "Previewing alias substitutions" on page 8.

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

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

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

Command alias syntax

The definition of a command alias can be quite complex. This section describes the basic syntax for defining a command alias and introduces the elements that you can use in a definition; the sections that follow provide a host of examples to illustrate how you can define increasingly complex aliases.

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

```
alias = transformation
```

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

```
fiches = files
```

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

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

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

Syntax for the transformation can vary widely; here's one possibility:

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

For example:

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

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

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

And the command would show the last five submitted changelists.

Alias definitions may contain the following elements:

- command arguments
- environment variables

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

The following table describes the special operators that you may use in command alias definitions:

Operator	Meaning
`$(arg)`	Specifies an alias argument in the *alias* and is matched with arguments in the *transformation*.
`#`	As the first non-blank character of a line, indicates a comment.
`&&`	Chain commands. See "Limitations" on page 9 for information about chaining commands.
`\`	Continue line (use to break up long lines when there are no new commands).
`<`	Take input from.
`>`	Send output to.
`$(EQ)`	Equal to.
`$(LT)`	Less than.

Operator	Meaning
`$(GT)`	Greater than.
`p4subst`	String substitution, for example in editing specs; for more information, see "Editing specifications" on page 9

Because it is hard to include these elements in a syntax definition, their use is illustrated in the next sections, which provide simple and complex sample alias definitions. Reading these sections will help you learn how to define command aliases and how to use these operators.

Basic examples

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

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

```
me = set P4USER
```

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

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

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

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

- **Set different defaults.**

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

- **Simplify system administration.**

  Shutting down the server, displaying active users:

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

- **Remembering to clean up empty changelists**

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

- **Change the order of arguments.**

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

Now the following command does what you want:

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

Complex examples

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

- **Cherry picking.**

This alias definition:

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

Turns this command:

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

Into this:

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

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

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

You could then execute a command like the following:

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

This would run the following commands:

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

- **Simple pipelining.**

  Starting with a simple example:

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

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

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

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

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

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

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

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

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

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

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

Putting it all together

Combining the various elements shown so far allows you to build interesting aliases.

- **Make a new task stream.**

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

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

  Then switching to a new stream becomes very simple; for example:

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

- **Delete a stream.**

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

- **Making an alias of an alias,**

```
checkout = sync
commit = submit

co  = checkout
cmt = commit
```

Previewing alias substitutions

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

For example, if your alias file contains the following:

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

And you execute the following command:

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

The command would return:

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

Advanced topics

This section describes more advanced uses of command aliases.

Editing specifications

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

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

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

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

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

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

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

Creating alias files for each workspace

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

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

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

Limitations

Aliases can be very powerful. Be mindful of consequences:

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

Naming conventions

Perforce enforces only a few rules related to what you name Perforce objects:

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

Otherwise, naming conventions are limited to best practices; the guiding principles being to use names that help you guess what the content is, to use consistent naming schemes, and to choose names that are likely to group like objects in alphabetical listings.

The following table provides some suggestions that are specific to Perforce objects:

Object	Naming convention
branches	Best to name them.
clients	The following scheme is commonly used, but not enforced in any way. Use it if it suits your purpose. *user.machineTag.product* *user.machineTag.product.branch* *user* is the OS user; *machineTag* is the host name or something that describes the host; for example `Win7VM` or `P4MBPro` (for Perforce MacBook Pro). Whether you use *product* or *product.branch* depends on whether your workspace gets re-purposed from stream to stream (in which case just use *product*), or you have multiple workspaces, one for each branch (in which case use *product.branch*, effectively tying the workspace name to the branch).
depots	Best to keep the names short. Depot names are part of an organization hierarchy for all your digital assets, so naming them and planning directory structure is especially important.
jobs	Name jobs to match your external defect tracker issues: for example, `PRJ-1234` for JIRA issues.
labels	Label names are site-dependent and might vary with code management schemes and versioning needs. For example `R-3.2.0` might refer to release 3.2.0.

Creating scripts

In addition to chaining commands in an alias to create a light script, you can combine the commands described in this manual in scripts. Perforce works with *Triggers*, user-written scripts called by a Perforce server whenever certain operations occur. Such operations include changelist submissions changes to forms, login attempts, and so on.

For more information about writing scripts, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

p4 add

Synopsis

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

Syntax

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

Description

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

The commands **p4 add *** or **p4 add ...** are synonymous to **p4 reconcile -a *** — that is, all files in the workspace that do not exist in the depot will be opened for add. Using the **-a** option does not affect the behavior of **p4 add -d**.

To open a file with **p4 add**, the file must exist in your client view, but does not need to exist in your workspace at the time of **p4 add**. The file must, however, exist in your workspace when you run **p4 submit**, or the submission will fail. **p4 add** does not create or overwrite files in your workspace; if a file does not exist, you must create it yourself.

By default, the specified files are opened in the default changelist. To open the files in a specified changelist, use the **-c** option. (To move files from the default changelist to a numbered changelist, use the **p4 change** command.)

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

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

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

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

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

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

Options

-c *changelist*	Opens the files for **add** within the specified *changelist*. If this option is not used, the files are linked to the default changelist.
-d	Downgrade file open status to simple add.
-f	Use the **-f** option to force inclusion of wildcards in filenames. See "File Specifications" on page 575 for details.
-I	Do not perform any ignore checking; ignore any settings specified by **P4IGNORE**.
-n	Preview which files would be opened for add, without actually changing any files or metadata.
-t *filetype*	Adds the file as the specified *filetype*, overriding any settings in the typemap table. See "File Types" on page 585 for a list of Perforce file types.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- "Wildcards" on page 575 in file specifications provided to **p4 add** are expanded by the local operating system, not by the Perforce service. For instance, the **...** wildcard cannot be used with **p4 add**.
- In Perforce, there is no difference between adding files to an empty depot and adding files to a depot that already contains other files. You can populate new, empty depots by adding files from a client workspace with **p4 add**.

- Do not use ASCII expansions of special characters with **p4 add -f**. To add the file **status@june.txt**, use:

  p4 add -f status@june.txt

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

Examples

p4 add -t binary file.pdf	Assigns a specific file type to a new file, overriding any settings in the typemap table.
p4 add -c 13 *	Opens all the files within the user's current directory for **add**, and links these files to changelist **13**.
p4 add README ~/src/*.c	Opens all ***.c** files in the user's **~/src** directory for **add**; also opens the **README** file in the user's current working directory for **add**. These files are linked to the default changelist.
p4 add -f *.c	Opens a file named ***.c** for **add**. To refer to this file in views, or with other Perforce commands, you must subsequently use the hex expansion **%2A** in place of the asterisk. For more information, see "Limitations on characters in filenames and entities" on page 578.

Related Commands

To open a file for edit	**p4 edit**
To open a file for deletion	**p4 delete**
To move (rename) a file	**p4 move**
To copy all open files to the depot	**p4 submit**
To read files from the depot into the client workspace	**p4 sync**
To create or edit a new changelist	**p4 change**
To change default behavior of text and binary file detection	**p4 configure**
To list all opened files	**p4 opened**
To revert a file to its unopened state	**p4 revert**

To move an open file to a different pending changelist	**p4 reopen**
To change an open file's file type	**p4 reopen -t filetype**

p4 admin

Synopsis

Perform administrative operations on the server.

Syntax

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

Description

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

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

To restart the service, use **p4 admin restart**. The database is locked, the service restarts, and any **p4 configure** settings that require a restart are then applied.

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

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

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

The **p4 admin journal** command is equivalent to **p4d -jj**. For details, see the *Helix Versioning Engine Administrator Guide: Fundamentals*. The files are created in the server root specified when the Perforce service was started.

The **p4 admin updatespecdepot** command causes the Perforce service to archive stored forms (specifically, **client**, **depot**, **branch**, **label**, **typemap**, **group**, **user**, and **job** forms) into the spec depot. If the **-a** option is used, all of the form specification types are archived. If the **-s** option option is used,

then only those of the specified *type* are archived. Only those forms that have not yet been archived are created.

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

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

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

Options

`-a`	For **p4 admin updatespecdepot**, update the spec depot with all current forms.
`-s` *type*	For **p4 admin updatespecdepot**, update the spec depot with forms of the specified type, where type is one of `client`, `depot`, `branch`, `label`, `typemap`, `group`, `user`, `job`, `stream`, `triggers`, `protect`, `server`, `license`, `jobspec`.
`-z`	For **p4 admin checkpoint** and **p4 admin journal**, save the checkpoint and saved journal file in compressed (gzip) format, appending the `.gz` suffix to the files.
`-Z`	For **p4 admin checkpoint** and **p4 admin journal**, save the checkpoint in compressed (gzip) format, appending the `.gz` suffix to the file, but leave the journal uncompressed for use by replica servers.
g-opts	See "Global Options" on page 571.

Usage Notes

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

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

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

Running **p4 admin resetpassword -a** resets only the passwords of users who presently exist (and who have passwords).

- Because **p4 admin stop** shuts down the Perforce service, you may see an error message indicating that the connection was closed unexpectedly. You can ignore this message.
- The spec depot must exist before running **p4 admin updatespecdepot**.
- **p4 dbstat**, **p4 lockstat**, and **p4 logstat** are standalone commands; the old **p4 admin** syntax remains as an alias for backward compatibility.
- For more about administering Perforce, see the *Helix Versioning Engine Administrator Guide: Fundamentals* and *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

Examples

p4 admin stop	Stop the shared Perforce service
p4 admin checkpoint	Create a checkpoint named **checkpoint.*n***, and start a new journal named **journal**, copying the old journal file to **journal.*n*-1**, where ***n*** is a sequence number.
p4 admin checkpoint name	Create a checkpoint named **name.ckp.*n***, and start a new journal named **journal**, copying the old journal file to **name.jnl.*n*-1**, where ***n*** is a sequence number.

Related Commands

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

p4 aliases

Synopsis

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

For complete information, see "Command aliases" on page 2.

Syntax

p4 [*g-opts*] aliases

Description

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

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

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

Options

g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 annotate

Synopsis

Print file lines along with their revisions.

By default, **p4 annotate** ignores changes to text files over 10 MB in length. Perforce superusers can override this limit by setting the **dm.annotate.maxsize** configurable.

Syntax

```
p4 [g-opts] annotate [-a -c -i -I -q -t -u -T] [-doptions] 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). Using the **-u** option displays the name of the user who modified the change and the date when the modification occurred. If you also want to know why the modification happened, you can use the **p4 filelog** command on the indicated revision(s).

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

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

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

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

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

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

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

```
320: samwise 2013/05/06            sr->w.digest.Clear();
172: markm 2004/10/27              sr->w.size.Unknown();
169: seiwald 2004/04/21            sr->w.traitLot.Clear();
196: anton 2006/06/12              sr->w.tampered.Clear();
```

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

```
... upper 962279
... lower 961206
... user samwise
... time 2011/03/09 11:57:14
... client samwise-silver
... data    else
```

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

Options

`-a`	All lines, including deleted lines and lines no longer present at the head revision, are included. Each line includes a starting and ending revision.
`-c`	Display the changelist number, rather than the revision number, associated with each line. If you use the `-a` option and the `-c` option together, each line includes a starting and ending changelist number.
`-d`*`options`*	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" on page 25 for a listing of these options.
`-i`	Follow file history across branches. If a file was created by branching, Perforce includes revisions up to the branch point. The use of the `-i` option implies the `-c` option. The `-i` option cannot be combined with `-I`.
`-I`	Follow integrations into the file. If a line was introduced into the file by a merge, the source of the merge is indicated as the changelist that introduced the line. If that source was itself the result of an integration, that source will be used instead, and so on. The use of the `-I` option implies the `-c` option. The `-I` option cannot be combined with `-i`.
`-q`	Quiet mode; suppress the one-line header for each file.
`-t`	Force **`p4 annotate`** to display non-text (binary) files.
`-u`	Display the name of the user who modified the change and the date when the modification occurred.

`-T | --tab=N`	Align output to a tab stop of 8. You can specify a different tab value using the `--tab` option and specifying the desired value for *N*.
g-opts	See "Global Options" on page 571.

Usage Notes

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

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

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

Examples

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

p4 archive

Synopsis

Archive obsolete revisions to an archive depot.

Syntax

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

Description

Moves the specified revisions into a *depot* of type `archive`.

When files are moved into an archive depot, their last action is changed to `archive`. Commands that access file content (for example, `p4 sync`, `p4 diff`, and so on) skip `archive` revisions, but commands that do not require access to file content (such as `p4 filelog`, for example) continue to report to metadata concerning the archived revisions.

Set the `server.locks.archive` configurable to disable server locks when running the `p4 archive` command.

Warning	**Use `p4 archive -p` with caution.** This is one of only two commands in Perforce that actually removes file data. (The other command that removes file data is `p4 obliterate`.)

Options

`-D depot`	Specify an archive depot to which files are to be archived.
`-h`	Do not archive head revisions.
`-n`	Do not archive revisions; report on which revisions would have been archived.
`-p`	Purge any archives of the specified files named in the archive depot. (The action for affected revisions is set to `purge` on completion. File contents are no longer accessible from `p4 restore`.)
`-q`	Quiet mode; suppress messages about skipped revisions.
`-t`	Archive text files (or other revisions stored in delta format, such as files of type `binary +D`)
g-opts	See "Global Options" on page 571.

Usage Notes

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

- By default, only files stored in full (`+F`) or compressed (`+C`) are archived. The files must be in a `local` depot (not a `remote` or another `archive` depot), and must neither be copied nor branched to (or from) another revision.
- To archive files stored in delta format, use the `-t` option. Be aware that there may be a computational cost associated with the manipulation of large numbers of RCS deltas.
- You can use **`p4 archive -n`** for testing purposes before mounting the filesystem associated with the archive depot. Storage for the archive depot must be mounted before running this command without the `-n` option.
- If a single revision is specified as a file argument, **`p4 archive`** implicitly targets revisions #1 through the specified revision for archiving. To archive only a single revision ***`rev`***, use the form **`p4 archive file#`*`rev`*`,`*`rev`***.
- If a revision is stored in an archive depot, and the stored revision is accessible to the versioning service, end users can use **`p4 print -A -o `*`filename myFile`*** to a file in order to determine which archived revision(s) are desired before (optionally) requesting that a Perforce Administrator use **`p4 restore`** to restore the file.

Examples

`p4 archive `*`file`*`#3`	Archive revisions 1, 2, and 3 of ***`file`***.
`p4 archive `*`file`*`#3,3`	Archive revision 3 of ***`file`***.
`p4 print -A -o `*`myOutputFile myArchiveFile`*	Redirect the contents of ***`myArchiveFile`*** to *myOutputFile* without restoring the archived file. Administrative privileges are not required. *myArchiveFile* might be something like `//archive/depot/myFile`

Related Commands

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

p4 attribute

Synopsis

Set per-revision attributes on revisions

Syntax

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

Description

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

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

Options

`-e`	Indicates that the value is specified in hex.
`-f`	Set the attribute on submitted files. If a propagating trait is set on a submitted file, a revision specifier cannot be used, and the file must not be currently open in any workspace.
`-i`	Read an attribute value from the standard input. Only one file argument is allowed when using this option.
`-n name`	The name of the attribute to set.
`-p`	Create a propagating attribute: an attribute whose value is propagated to subsequent revisions whenever the file is opened with **p4 add**, **p4 edit**, or **p4 delete**.
`-v value`	The value of the attribute to set. To clear an attribute, omit the `-v` option.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- Multiple attributes can be set or cleared by specifying multiple `-n name` options and an equal number of corresponding `-v value` options (to set) or no `-v` options (to clear).

- In distributed environments, the following commands are not supported for files with propagating attributes: **p4 copy**, **p4 delete**, **p4 edit**, **p4 integrate**, **p4 reconcile**, **p4 resolve**, **p4 shelve**, **p4 submit**, and **p4 unshelve**. Integration of files with propagating attributes from an edge server is not supported; depending on the integration action, target, and source, either the **p4 integrate** or the **p4 resolve** command will fail.

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

p4 branch

Synopsis

Create or edit a branch mapping and its view.

Syntax

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

Description

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

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

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

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

Form Fields

Field Name	Type	Description
`Branch:`	read-only	The branch name, as provided on the command line.
`Owner:`	mandatory	The owner of the branch mapping. By default, this will be set to the user who created the branch. This field is unimportant unless the `Option:` field value is `locked`. The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Access:`	read-only	The date the branch mapping was last accessed.
`Update:`	read-only	The date the branch mapping was last changed.
`Options:`	mandatory	Either `unlocked` (the default) or `locked`.

Field Name	Type	Description
		If `locked`, only the `Owner:` can modify the branch mapping, and the mapping can't be deleted until it is `unlocked`.
`Description:`	optional	A short description of the branch's purpose.
`View:`	mandatory	A set of mappings from one set of files in the depot (the ***`source files`***) to another set of files in the depot (the ***`target files`***). The view maps from one location in the depot to another; it can't refer to a client workspace. For example, the branch view `//depot/main/... //depot/r2.1/...` maps all the files under `//depot/main` to `//depot/r2.1`.

Options

`-d`	Delete the named branch mapping. Files are not affected by this operation; only the stored mapping from one codeline to another is deleted. Normally, only the user who created the branch can use this option.
`-f`	Force option. Combined with `-d`, permits Perforce administrators to delete branches they don't own. Also permits administrators to change the modification date of the branch mapping (the `Update:` field is writable when using the `-f` option).
`-i`	Read the branch mapping from standard input without invoking an editor.
`-o`	Write the branch mapping to standard output without invoking an editor.
`-P` ***`parent`***	For a specified stream, display the mapping that is generated by treating the stream as a child of the specified parent. Requires `-S`.
`-S` ***`stream`***	Display the mapping generated for the specified stream. This option enables you to see how change is propagated between the stream and its parent.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- A branch view defines the relationship between two related codelines. For example, if the development files for a project are stored under `//depot/project/dev/...`, and you want to create

a related codeline for the 2.0 release of the project under `//depot/project/r2.0/...`, specify the branch view as:

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

Branch views can contain multiple mappings. See "Views" on page 581 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/..."
  ```

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

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

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

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

  See "Views" on page 581 for more information on specifying views.

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

Related Commands

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

p4 branches

Synopsis

List existing branch mappings.

Syntax

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

Description

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

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

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

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

Options

`-e filter`	List only branches matching *filter* (case-sensitive).
`-E filter`	List only branches matching *filter* (case-insensitive).
`-m max`	List only the first *max* branch mappings.
`-t`	Display the time as well as the date of the last update to the branch.
`-u user`	List only branches owned by *user*.
`--me`	Equivalent to `-u $P4USER`.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To create or edit a branch mapping **p4 branch**

p4 cachepurge

Synopsis

Reclaim disk space on a replicated server.

Syntax

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

Description

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

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

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

Options

`-a`	Delete all file content. This option reclaims the maximum amount of disk space, but any file content must be retrieved from the master. If the `-O` option is not specified, file names are used to determine order of deletion.
`-D file ...`	Limit action of command to the specified set of files.
`-f n`	Delete sufficient file content to leave *n* number of bytes of free space for the file system.
`-i n`	Repeat the command every *n* seconds. If you omit this option, the command runs only once.
`-m n`	Delete *n* file revisions. The amount of space this frees up depends on the size of the files.
`-n`	Display a preview of the **cachepurge** operation without deleting any files.

`-O`	Delete the files from the oldest to the newest; that is, delete older files before deleting newer files.
`-R`	Delete files in the order specified by the `-O` option. If the `-O` option is not specified, file names are used to determine order of deletion.
`-s` *n*	Delete *n* bytes of file data. This can be helpful in those cases when you can predict the growth rate of file system resources.
`-S` *n*	Do not delete the *n* most recent revisions of each file. For example, specifying `-S 1`, means that the head revision of each file is retained in the replica's cache if it is already there.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 change

Synopsis

Create or edit a changelist specification.

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

Syntax

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

Description

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

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

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

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.

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.

Field Name	Type	Description
`Client:`	Read-only	Name of current client workspace.
`Date:`	Read-only	Date the changelist was last modified.
`User:`	Read-only	Name of the change owner. The owner of an empty pending changelist (that is, a pending changelist without any files in it) can transfer ownership of the changelist to another existing user either by editing this field, or by using the `-U` *`user`* option. The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Status:`	Read-only	`pending`, `shelved`, `submitted`, or `new`. Not editable by the user. The status is `new` when the changelist is created, `pending` when it has been created but has not yet been submitted to the depot with **`p4 submit`**, `shelved` when its contents are shelved with **`p4 shelve`**, and `submitted` when its contents have been stored in the depot with **`p4 submit`**.
`Description:`	Writable, mandatory	Textual description of changelist. This value *must* be changed before submission. If you do not have access to a restricted changelist, the description is replaced with a "no permission" message.
`Jobs:`	List	A list of jobs that are fixed by this changelist. The list of jobs that appears when the form is first displayed is controlled by the **`p4 user`** form's `JobView:` setting. Jobs can be deleted from or added to this list.
`Type:`	Writable, optional	Type of change: `restricted` or `public`. The `Type:` field can be used to hide the change or its description from users. A `shelved` or `committed` change (as denoted in the `Status:` field) that is `restricted` is accessible only to users who own the change or have `list` permission to at least one file in the change. Public changes are displayed without restrictions. By default, changelists are `public`. A Perforce superuser can set the default changelist type (for changelists created after the configurable is set) by setting the `defaultChangeType` configurable.

Field Name	Type	Description
`Files:`	List	The list of files being submitted in this changelist. Files can be deleted from this list, and files that are found in the default changelist can be added.
`ImportedBy:`	Read-only	Displays the name of the user who ran the **p4 fetch**, **p4 push**, or **p4 unzip** command that imported this change into the server. This field is primarily useful for distributed versioning (DVCS) scenarios, in which changelists are copied from one server to another, and help you correlate the changelist's basic identity as it is copied. In such configurations, Perforce recommends using the **submit.identity** configurable to enable automatic generation of changelist identities by the **p4 submit**.
`Identity:`	Writable, mandatory	Contains a label which uniquely identifies this changelist across all servers where it has been fetched, pushed, or unzipped. This field is primarily useful for distributed versioning (DVCS) scenarios, in which changelists are copied from one server to another, and help you correlate the changelist's basic identity as it is copied. In such configurations, Perforce recommends using the **submit.identity** configurable to enable automatic generation of changelist identities by the **p4 submit**.

Options

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

`-f -d`	Forcibly delete a previously submitted changelist. Only a Perforce administrator or superuser can use this command, and the changelist must have had all of its files removed from the system with **p4 obliterate**.
`-i`	Read a changelist description from standard input. Input must be in the same format used by the **p4 change** form.
`-o`	Write a changelist description to standard output.
`-O`	If a changelist was renumbered on submit, and you know only the original changelist number, use `-O` and the original changelist number to view or edit the changelist.
`-I`	Specifies that the changelist number is the Identity field of a changelist.
`-s`	Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of `closed`. To leave a job unchanged, use the special status of `same`. On new changelists, the fix status is displayed as the special status `ignore`. (If the status is left unchanged, the job is not fixed by the submission of the changelist.) This option works in conjunction with the `-s` option to **p4 fix**, and is intended for use in conjunction with defect tracking systems.
`--serverid=`*`serverid`*	If you try to forcibly delete a changelist whose client is bound to another server, you need to specify the `--serverid` option and to specify the server id of the other server. This ensures that you do not accidentally delete the changelist, believing it to be on your own server. This variant of the **p4 change** command must be issued directly to the commit server.
`-t` *`type`*	Change a submitted changelist's *type* to either `restricted` or `public`.
`-u`	Update a submitted changelist. Only the `Jobs:`, `Description:`, or `Type:` fields can be updated, and only the submitter of the changelist can update the changelist. The `-u` and the `-f` options are mutually exclusive.
`-U user`	The `-U` *user* option changes the owner of an empty pending changelist. To reassign a changelist, you must either already be the changelist's owner, or a user with `admin` permissions and use the `-f` option. (Unlike manually editing the `User:` field in the **p4 change** form, this option is much more convenient for use within a trigger or script.)
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

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

Examples

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

Related Commands

To submit a changelist to the depot	**`p4 submit`**
To move a file from one changelist to another	**`p4 reopen`**
To remove a file from all pending changelists	**`p4 revert`**
To list changelists meeting particular criteria	**`p4 changes`**
To list opened files	**`p4 opened`**
To list fixes linked to particular changelists	**`p4 fixes`**
To link a job to a particular changelist	**`p4 fix`**

To remove a job from a particular changelist	**p4 fix -d**
To list all the files listed in a changelist	**p4 opened -c changelist**
To obtain a description of files changed in a changelist	**p4 describe changelist**

p4 changelist

Synopsis

Create or edit a changelist specification.

Syntax

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

Description

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

p4 changelists

Synopsis

List submitted and pending changelists.

Syntax

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

Description

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

p4 changes

Synopsis

List submitted and pending changelists.

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

Syntax

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

Description

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

By default, the format of each line is:

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

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

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

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

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

Revision specifications and revision ranges can be included in the file patterns. 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* options to limit output to only those changelists made from the named client workspace or the named user.
- Use the **-s** *status* option to limit output to only those changelists with the provided *status* (**pending**, **shelved**, or **submitted**) value.
- Use the **-e** *changelist#* option to display only changes that are greater or equal to the specified changelist number.

In a distributed configuration, changes that are pending or shelved on an Edge Server, are visible via the **p4 changes** command on other servers in the installation.

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

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

The `-u` global option has a different meaning than the `-u` option for the **p4 changes** command. For example, the following command means the **p4 changes** command is executed by user JOANNA, and it asks for a list of those changes made by JENNY.

```
$ p4 -u JOANNA changes -u JENNY
```

Options

`-c client`	List only changes made from the named client workspace.
`-f`	View restricted changes (requires `admin` permission)
`-i`	Include changelists that affected files that were integrated with the specified files.
`-l`	List long output, with the full text of each changelist description.
`-L`	List long output, with the full text of each changelist description truncated at 250 characters.
`-m max`	List only the highest numbered *max* changes.
`-s status`	Limit the list to the changelists with the given status (`pending`, `submitted`, or `shelved`)
`-t`	Display the time as well as the date of each change.
`-u user`	List only changes made from the named user.
`--me`	Equivalent to `-u $P4USER`.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- If **p4 changes** is called with multiple file arguments, the sets of changelists that affect each argument are evaluated individually. The final output is neither combined nor sorted; the effect is the same as calling **p4 changes** multiple times, once for each file argument.

- If files are not specified, **`p4 changes`** limits its report according to whether or not changes are public or restricted. Restricted **`submitted`** or **`shelved`** changes are not reported unless you either own the change or have **`list`** permission for at least one file in the change. Restricted **`pending`** (but unshelved) changes are visible only to the change owner.
- **`p4 changes`** ***`myfile`*****`#have`** accesses the db.have table lockless for the duration of the compute phase.

Examples

`p4 changes -m 5 //depot/project/...`	Show the last five submitted, pending, or shelved changelists that include any file under the **`project`** directory.
`p4 changes -m 5 -c eds_elm`	Show the last five submitted, pending, or shelved changelists from client workspace **`eds_elm`**.
`p4 changes -m 5 -s submitted -u edk`	Show the last five submitted changelists from user **`edk`**.
`p4 changes file.c@2010/05/01,2010/06/01`	Show any changelists that include file **`file.c`**, as mapped to the depot through the client view, during the month of May 2010.
`p4 changes -m 1 -s submitted`	Output a single line showing the changelist number of the last submitted changelist.
`p4 changes @2011/04/01,@now`	Display all changelists submitted from April 1, 2011 to the present.
`p4 changes @2011/04/01`	Display all changelists submitted *before* April 1, 2011.

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 clean

Synopsis

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

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

Syntax

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

Description

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

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

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

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

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

Options

-a	Added files: Find files in the workspace that have no corresponding files in the depot and delete them.
-d	Deleted files: Find those files in the depot that do not exist in your workspace and add them to the workspace.
-e	Edited files: Find files in the workspace that have been modified and restore them to the last file version that has synced from the depot.
-I	Do not perform any ignore checking; ignore any settings specified by P4IGNORE for added files.
-l	Display output in local file syntax with relative paths, similar to the workspace-centric view of **p4 status**.

`-n`	Preview the results of the operation without performing any action.
file	The files whose versions you want reconciled with their latest depot versions. If you omit this parameter, the files in your local working directory are used.
g-opts	See "Global Options" on page 571.

Usage Notes

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

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

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

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

Related Commands

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

p4 client

Synopsis

Create or edit a client workspace specification and its view.

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

Syntax

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

Description

A Perforce *client workspace* is a set of files on a user's machine that mirror a subset of the files in the depot. More precisely, it is a named mapping of depot files to workspace files. Use the **p4 client** command to create or edit a client workspace specification; invoking this command displays a form in which the user enters the information required by Perforce to maintain the workspace.

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

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 workspace are mapped to the depot, and vice-versa.

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

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

Form Fields

Field Name	Type	Description
Client:	Read-only	The client workspace name, as specified in the P4CLIENT environment variable or its equivalents.

Field Name	Type	Description
		When called without a *clientname* argument, **p4 client** operates on the workspace specified by the P4CLIENT environment variable or one of its equivalents. If called with a *clientname* argument on a **locked** workspace, the workspace specification is read-only.
Owner:	Writable, mandatory	The name of the user who owns the workspace. The default is the user who created the workspace. The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
Update:	Read-only	The date the workspace specification was last modified.
Access:	Read-only	The date and time that the workspace was last used in any way. The access time is only valid for the server where the client resides — where it was created or where it was moved to. This is an issue only in a commit-edge architecture. Reloading a workspace with **p4 reload** does not affect the access time.
Host:	Writable, optional	The name of the workstation on which this workspace resides. If included, operations on this client workspace can be run *only* from this host. If not set, access is allowed from any host. The hostname must be provided exactly as it appears in the output of **p4 info** when run from that host. This field is meant to prevent accidental misuse of client workspaces on the wrong machine. Providing a host name does not guarantee security, because the actual value of the host name can be overridden with the **-H** option to any **p4** command, or with the P4HOST environment variable. For a similar mechanism that does provide security, use the IP address restriction feature of **p4 protect**.
Description:	Writable, optional	A textual description of the workspace. The default text is **Created by owner**.
Root:	Writable, mandatory	The directory (on the local host) relative to which all the files in the **View:** are specified. The default is the current working directory. The path must be specified in local file system syntax. If you change this setting, you must physically relocate any files that currently reside there. On Windows client machines, you

Field Name	Type	Description
		can specify the root as `null` to enable you to map files to multiple drives.
`AltRoots:`	Writable, optional	Up to two optional alternate client workspace roots. Perforce applications use the first of the main and alternate roots that match the application's current working directory. Use the **`p4 info`** command to display the root being used. This enables users to use the same Perforce client workspace specification on multiple platforms, even those with different directory naming conventions. If you are using multiple or alternate workspace roots (the `AltRoots:` field), you can always tell which root is in effect by looking at the `Client root:` reported by **`p4 info`**. If you are using a Windows directory in any of your workspace roots, you must specify the Windows directory as your main workspace root and specify your other workspace roots in the `AltRoots:` field. For example, an engineer building products on multiple platforms might specify a main client root of `C:\Projects\Build` for Windows builds, and an alternate root of `/staff/`*`userid`*`/projects/build` for any work on UNIX builds.
`Options:`	Writable, mandatory	A set of seven switches that control particular workspace options. See "Usage Notes" on page 61 for a listing of these options.
`SubmitOptions:`	Writable, mandatory	Options to govern the default behavior of **`p4 submit`**. • `submitunchanged` All open files (with or without changes) are submitted to the depot. This is the default behavior of Perforce. • `submitunchanged+reopen` All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist. • `revertunchanged` Only those files with content, type, or resolved changes are submitted to the depot. Unchanged files are reverted. • `revertunchanged+reopen`

Field Name	Type	Description
		Only those files with content, type, or resolved changes are submitted to the depot and reopened in the default changelist. Unchanged files are reverted and *not* reopened in the default changelist. • `leaveunchanged` Only those files with content, type, or resolved changes are submitted to the depot. Any unchanged files are moved to the default changelist. • `leaveunchanged+reopen` Only those files with content, type, or resolved changes are submitted to the depot. Unchanged files are moved to the default changelist, and changed files are reopened in the default changelist. This option is similar to **`submitunchanged+reopen`**, except that no unchanged files are submitted to the depot.
`LineEnd:`	Writable, mandatory	Configure carriage-return/linefeed (CR/LF) conversion. See "Usage Notes" on page 61 for a listing of these options.
`Stream:`	Writable, optional	Associates the workspace with the specified stream. Perforce generates the view for stream-associated workspaces: you cannot modify it manually.
`StreamAtChange`	Writable, optional	A changelist number that sets a back-in-time view of a stream. When **`StreamAtChange`** is set, running **`p4 sync`** (when called with no arguments) updates the workspace to files at this changelist revision, instead of the head revision. You cannot submit changes (**`p4 submit`** returns an error) when **`StreamAtChange`** is set, because the workspace view no longer reflects the current stream inheritance. This field is ignored unless the **`Stream`** field is also set to a valid stream.
`ServerID:`	Writable, optional	If set, restricts usage of the workspace to the named server. If unset, use is allowed on master server and on any replicas of the master other than Edge servers.
`View:`	Writable, multi-line	Specifies the mappings between files in the depot and files in the workspace. See **`p4 help views`** for more information. A new view takes effect on the next **`p4 sync`** operation.
`ChangeView:`	Writable, optional, multi-line	Restricts access to depot paths to a particular point in time. Files specified for the ChangeView field are read-only: they may be opened but not submitted. For example:

Field Name	Type	Description
		`//depot/path/...@1000` Revisions of the files in the specified path will not be visible if they were submitted after the specified changelist number. Files matching a ChangeView path may not be submitted.
`Type:`	Writable, optional	By default clients are **`writeable`**. Specify **`readonly`** for short lived clients used in build automation scripts. Such clients cannot edit or submit files, but this should not be an issue in build scripts. Specifying **`partitioned`** achieves the same thing as the **`readonly`** setting except that it additionally allows you to edit and submit files using that client. Using writeable clients in build automation scripts can lead to **`db.have`** table fragmentation, which is used to track what files a client has synced. If you are experiencing such issues, use a read-only or partitioned client instead. A client of type **`readonly`** or **`partitioned`** is assigned its own personal **`db.have`** database table. The location of this table must first be specified with the **`client.readonly.dir`** configurable by an administrator.
`Backup:`	Writable, optional	Not currently in use. Applies only to clients bound to cluster workspace servers. If a backup startup process is launched on a cluster workspace-server, by default all writable clients (see **`Type`** field) bound to that workspace-server are periodically backed up. To disable backup for a particular client, its owner must set the value of the **`Backup`** field to **`disable`**. Note: If the backup startup process is configured with auto-unload unused clients on, neither readonly clients nor those with backup disabled, will be considered for auto unload. For complete information, see version 15.1 of *Helix Versioning Engine Administrator Guide: Cluster Management*.

Options

`-d` *`clientname`*

Delete the specified client workspace whether or not the workspace is owned by the user. The workspace must be unlocked and must have no opened files or pending changes. (The **`-f`** option permits Perforce administrators to delete locked workspaces owned by other users.) Clients can be deleted even if they have shelved files (see **`-Fs`** option).

If you try to forcibly delete a client bound to another server, you need to specify the **`--serverid`** option and specify the server id of the other server.

	This ensures that you do not accidentally delete the client believing it to be connected to your own server.
`-f`	Allows the last modification date, which is normally read-only, to be set. Administrators can use the `-f` option to delete or modify locked workspaces owned by other users. Use of this option requires `admin` access granted by **`p4 protect`**.
`-Fs`	Allows the deletion of a client even when that client contains shelved changes. The client is deleted and the shelved changes are left intact. (You must use the `-f` option with the `-Fs` option.)
`-i`	Read the client workspace specification from standard input.
`-o`	Write the client workspace specification to standard output.
`-o -c` *`change`*	When used with `-S` *`stream`*, displays the workspace specification that would have been created for a *stream* at the moment the *change* was submitted.
`-s`	Switch workspace view. To switch the workspace view to a stream, specify `-S` *`stream`*. To switch the view defined for another workspace, specify `-t` *clientname*. Switching views is not allowed in a client that has opened files. The `-f` option can be used with `-s` to force switching with opened files. View switching has no effect on files in a client workspace until **`p4 sync`** is run.
`--serverid=`*`serverid`*	If you try to forcibly delete a client bound to another server, you need to specify the `--serverid` option and specify the server id of the other server. This ensures that you do not accidentally delete the client believing it to be connected to your own server. This variant of the **`p4 client`** command must be issued directly to the commit server.
`-S stream`	Associates the workspace with the specified stream, which is used to generate its workspace view.
`-t` *`clientname`*	Copy client workspace *clientname*'s view and options into the `View:` and `Options:` field of this workspace. If you specify a default client template using the `template.client` configurable, you do not have to specify this option.
`-T type`	By default clients are `writeable`. You can also set `type` to `readonly` or `partitioned`. See the `Type` Field Name description in "Form Fields" on page 55 for details.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

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

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

  Use lowercase drive letters when specifying workspaces across multiple drives.

Options field

The `Options:` field contains six values, separated by spaces. Each of the six options have two possible settings; the following table provides the option values and their meanings:

Option	Choice	Default
`[no]allwrite`	If set, unopened files in the workspace are left writable. If `allwrite` is set and no `noclobber` is specified, a safe synchronization is performed. A setting of `allwrite` leaves unopened files writable by the current user; it does not set filesystem permissions to	`noallwrite`

Option	Choice	Default
	ensure that files are writable by any user of a multi-user system.	
`[no]clobber`	If set, a **p4 sync** overwrites ("clobbers") writable-but-unopened files in the workspace that have the same name as the newly-synced files. If **allwrite** is set and no **noclobber** is specified, a safe synchronization is performed.	`noclobber`
`[no]compress`	If set, the data stream between the user's workstation and the Perforce service is compressed. The compress option speeds up communications over slow links by reducing the amount of data that has to be transmitted. Over fast links, the compression process itself may consume more time than is saved in transmission. In general, compress should be set for line speeds under T1, and should be left unset otherwise.	`nocompress`
`[un]locked`	Grant or deny other users permission to edit or delete the workspace specification. (To make a **locked** workspace truly effective, the workspace's owner's password must be set with **p4 passwd**.) If **locked**, only the owner is able to use or edit the workspace specification. Perforce administrators can override the lock by using the **-f** (force) option with **p4 client**.	`unlocked`
`[no]modtime`	For files *without* the **+m** (modtime) file type modifier: • If **modtime** is set, the modification date (on the local filesystem) of a newly synced file is the datestamp *on the file* when the file was last modified. • If **nomodtime** is set, the modification date is the date and time *of sync*, regardless of version. For files *with* the **+m** (modtime) file type modifier: the modification date (on the local filesystem) of a newly synced file is the datestamp on the file when the file was submitted to the depot, regardless of the setting of modtime or nomodtime on the client. Files with the **modtime** (**+m**) type are primarily intended for use by developers who need to preserve original timestamps on files. The use of **+m** in a file type overrides the workspace's **modtime** or **nomodtime** setting. For a	**nomodtime** (date and time of sync) for most files. Ignored for files with the **+m** file type modifier.

Option	Choice	Default
	more complete discussion of the `+m` modifier, see "File Types" on page 585.	
`[no]rmdir`	If set, **`p4 sync`** deletes empty directories in a workspace if all files in the directory have been removed. 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 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 with your work.	`normdir`

Processing line endings

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

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

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

Option	Meaning
	The `share` option implicitly edits the file(s) during a submit. As a consequence, if you have set the `LineEnd` field to `share` in your client spec, the **`p4 resolve`** command may prompt you to edit the file before resolving.

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

http://answers.perforce.com/articles/KB/3096/

Working with Streams

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

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

Working with build servers

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

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

Using read-only clients

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

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

To set up a read-only client:

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

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

`/perforce/1/server.dbs/client/`*`hashdir`*`/db.myroc`

2. Set the `Type` field of the client spec to `readonly`.

Examples

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

Related Commands

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

p4 clients

Synopsis

List all client workspaces currently known to the system.

Syntax

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

Description

p4 clients lists all the client workspaces known to the Perforce versioning service. 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 2009/02/19 root /usr/src 'Joe's client'
```

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

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

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

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

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

Options

-a	List all client workspaces, not just workspaces bound to this server.
-e *filter*	List only client workspaces matching *filter* (case-sensitive).
-E *filter*	List only client workspaces matching *filter* (case-insensitive).
-m *max*	List only the first *max* client workspaces.
-s *serverID*	List only client workspaces bound to the specified *serverID*. On an edge server, the **-s** option defaults to the edge server's serverID.

`-S stream`	List client workspaces associated with the specified stream.
`-t`	Display the time as well as the date of the last update to the workspace.
`-u user`	List only client workspaces owned by *user*.
`--me`	Equivalent to `-u $P4USER`.
`-U`	List only client workspaces unloaded with **p4 unload**.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

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

p4 clone

Synopsis

Clone a new local Perforce server from a remote server.

Syntax

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

Description

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

Options

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

Usage Notes

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

Examples

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

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

Related Commands

To to initialize a Perforce server	`p4 init`

p4 configure

Synopsis

Set and manage server configuration variables.

Syntax

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

Description

Configuration variables are used to control and customize the behavior of the Perforce service. The configuration variables are described in **p4 help configurables**, in **p4 help environment**, and in "Configurables" on page 595. Configurable settings might affect the client, the server, or a proxy.

The **p4 configure** command provides one way to change the configuration of an active server. For information on how you set configurables that affect the client or the proxy, see "Configurables" on page 595.

You can set configurables in the following ways for the server; methods are shown in order of precedence:

- As command line options that are passed at server startup. For example:

  ```
  $ p4d -v net.keepalive.idle=2700
  ```

- Persistently, using the **p4 configure set** command.

  This method allows you to set the specified configurable for a named server or for any server.

- Using environment variables.

  When set using environment variables, certain server-related configurables are read-only; you cannot change P4ROOT or P4JOURNAL with **p4 configure**.

- On Windows using the **p4 set** command.

- On Unix, using the **export** command.

- Using default values, by taking no action.

Use **p4 configure show** to display the configuration state of the current server, a named server, or any configurable. Each configurable is displayed along with its value and an indication of what method was used to set it. Use **p4 configure unset** to unset the value of a configurable.

After installing Perforce, it is good practice to enable process monitoring by setting **monitor** to 1 or 2, require ticket-based authentication by setting **security** to 3 or 4, and preventing the automatic creation

of new users by setting `dm.user.noautocreate` to 1 or 2. Setting `dm.user.resetpassword` to 1 is also advisable; new users that you create (and to whom you assign an initial password) are forced to reset their passwords before they can issue commands.

Static, dynamic, and read-only configurables

Changes to dynamic configurables take effect the next time the server receives a new connection. Changes to static configurables, require a server restart. Values of read-only configurables may not be changed.

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

Changes to P4AUTH, P4PORT, the `startup.n` configurables used in replicated environments, `net.tcpsize`, and `net.backlog` require a restart.

To restart the server, use **p4 admin restart**.

Setting configurables in distributed environments

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

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

See *Helix Versioning Engine Administrator Guide: Multi-site Deployment* for details.

Accessing configurables when the server is down

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

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

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

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

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

For more information, see Accessing Server Configuration Variables:

http://answers.perforce.com/articles/KB_Article/Accessing-Server-Configuration-Variables

Options

`set` *`variable=value`*	Sets the named variable to the provided value.
`unset` *`variable`*	Unsets the named variable.
`show`	Shows the current configuration of the server currently specified by `P4PORT`.
`show allservers`	Shows the configuration variables for all servers known to the system.
`show` *`variable`*	Shows the setting of a specific configuration variable.
`show` *`P4NAME`*	If a Perforce server was invoked with `-In` *P4NAME* or with the `P4NAME` environment variable set to a server name, shows the settings of the named server.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To list all counters and their values — **`p4 counters`**

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

p4 copy

Synopsis

Copy files from one location in the depot to another.

Syntax

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

Description

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

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

Options

`-b` *branch*	Specify a branch view to be used to determine source and target files.
`-c` *change*	Open the files in the specified pending changelist rather than in the default changelist.
`-f`	Force the creation of extra revisions in order to explicitly record that files have been copied. Deleted source files are copied if they do not exist in the target, and files that are already identical are copied if they are not connected by existing integration records.
`-F`	Force copy operation; perform the operation when the target stream is not configured to accept a copy of the source. To determine a stream's expected flow of change, use **p4 istat**.
`-m` *max*	Specify the maximum number of files to copy, to limit the size of the operation.
`-n`	Preview the copy.
`-P` *parent*	Specify a target stream other than the parent of the source stream. Requires `-S`.

`-q`	Quiet mode; suppress normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.
`-r [toFile[rev] ...]`	Reverse the mappings in the branch view, integrating from the target files to the source files. Requires the `-b` option.
`-s fromFile[rev] [toFile ...]`	Treat *fromFile* as the source and both sides of the branch view as the target. To restrict the scope of the target further, specify the optional *toFile* parameter. Overrides the `-r` option, if specified. Requires `-b`.
`-S stream`	Specify the source stream. Changes are copied to its parent. You can use the `-r` option to reverse direction. To submit copied stream files, the current client must be switched to the target stream or to a virtual child stream of the target stream.
`-v`	Do not sync the target files. By default, **`p4 copy`** syncs the target files. If a large number of files is involved and you do not require the files to be present in your workspace, you can minimize overhead and network traffic by specifying `-v`.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
fromFile: Yes *toFile*: No	No	`read` access for *fromFile* `open` access for *toFile*

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

Examples

`p4 copy -S //projectX/dev/...`	Promote work from a development stream to the mainline.

Related Commands

Update a child stream with a more stable parent stream	**`p4 merge`**

Propagate changes	**p4 integrate**

p4 counter

Synopsis

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

Syntax

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

Description

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

The command includes the following variants:

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

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

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

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

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

- The variant **p4 counter -m *pair list*** defines multiple operations to be performed. Each operation is defined by a value pair in the pair list. To set a counter use a name and value; to delete a counter use a **-** (hyphen) followed by the name. See "Examples" on page 81.

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

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

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

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

See the **p4 counters** command for a list of Perforce counters. Superusers can use the **-f** option 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.

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

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

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

Options

-d *countername*	Delete variable *countername*.
-i *countername*	Increment variable *countername* by 1 and return the new value. This option can only be used with numeric counters.
-f	Set or delete counters that are reserved for use by Perforce (listed in **p4 help counters**). **Never** set the **change** counter to a value that is lower than its current value. Only operators or **super** users can use this flag.
-m *pair list*	Specify a list of operations to be performed. Each operation is defined by a value pair in the pair list. To set a counter, use a name and value; to delete a counter use a **-** (hyphen) followed by the name. See "Examples" on page 81.
-v	Display the previous value of the specified counter after the counter has been set or incremented.
g-opts	See "Global Options" on page 571.

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

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
		`super` to use the `-f` option

Examples

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

Related Commands

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

p4 counters

Synopsis

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

Syntax

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

Description

Perforce 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. These counters are currently used:

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

Options

-e *nameFilter*	List counters with a name that matches the *nameFilter* pattern, for example: **p4 counters -e 'mycounter-*'**
-m *max*	List only the first *max* counters.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

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

p4 cstat

Synopsis

Dump change/sync status for current client workspace.

Syntax

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

Description

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

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

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

A client workspace might have change 222 (that is, be synced to changelist 222), but depending on what others have done after the sync, could either need change 223 (if no files in changelist 223 have yet been synced), or have a partial sync of changelist 223 (if some, but not all, of the revisions in changelist 223 have been synced).

Options

g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

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

p4 dbschema

Synopsis

Report information about metadata in the Perforce database.

Syntax

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

Description

The **p4 dbschema** command reports information about the database structure in which the Perforce versioning service stores metadata.

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

The results are returned as tagged output.

This command is intended for systems integrators.

Options

tablename	Restrict output to the specified table name.
tableversion	Restrict output to the specified table version.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Examples

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

p4 dbstat

Synopsis

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

Syntax

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

Description

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

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

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

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

Options

-a	Display statistics for all tables.
-f	Displays a page count, free pages, and percent free data for the specified table(s).
-h	Display a histogram showing distances between leaf pages.
-s	Report file sizes of database tables.
dbtable	Display statistics for the specified table (for instance, **db.have**, **db.user**, and so on.) As you are most likely to need data for a particular table when working with support, they will let you know the name of the table for which you need information.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 dbverify

Synopsis

Perform low-level verification of the database tables.

Syntax

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

Description

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

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

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

Although the presence of a non-zero unlock count does not positively indicate corruption (and the presence of a zero unlock count does not guarantee data integrity), **p4 dbverify -U** has minimal performance impact.

Options

-t db.*tablename*	Restrict verification to the specified table name.
-U	Perform a less-detailed validation
-v	Provide verbose information on the verification.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- **`p4 dbverify`** is equivalent to **`p4d -xv`**.

p4 delete

Synopsis

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

Syntax

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

Description

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

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

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

Options

`-c changelist`	Opens the files for **delete** within the specified changelist. If this option is not provided, the files are linked to the default changelist.
`-k`	Delete the file on the shared versioning service, but keep a copy of the deleted file in your workspace.
`-n`	Preview which files would be opened for delete, without actually changing any files or metadata.
`--remote=remote`	Opens the file for delete in your personal server, and additionally — if the file is of type **+l** — takes a global exclusive lock on the file in the shared server from which which you cloned the file. For more information, see the section Support for exclusive locking in the Fetching and Pushing chapter of *Using Helix for Distributed Versioning*.
`-v`	Delete a file that is not synced into the client workspace. To use this option, specify these files in depot syntax; because such files are not synced, client syntax or local syntax can introduce ambiguities in the list of files to delete. (If the files are synced, **p4 delete -v *file*** removes the files from your workspace in addition to opening them for deletion.)

	To delete a set of files without transferring them to your workstation when another version of these files already exists in your workspace, use **p4 sync -k file**, followed by **p4 delete -k *file***. This allows you to delete a file that is in the depot without affecting the file (by the same name) in your workspace. For example, if you have `myfile` version 5 in the depot and `myfile` version 6 in your workspace, this sequence of commands, allows you to keep version 6 but delete version 5. If you don't do this and just do **p4 delete -v**, it will delete version 5 in the depot and version 6 in the workspace.
g-opts	See "Global Options" on page 571.

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**.
- Using an **+S*n*** file modifier results in special behavior when you delete and read a file: no file reversions are deleted that were submitted before the add or delete. For example, if a file of type **+S2** is marked as deleted in revision 5, and then re-added with the same file type and modifier, revisions 3 and 4 are not purged.

Examples

p4 delete //depot/README	Opens the file called `README` in the depot's top level directory for deletion. The corresponding file within the workspace is immediately deleted, but the file is not deleted from the depot until the default changelist is submitted.
p4 delete -c 40 *file*	Opens ***file*** in the current workspace for deletion. The file is immediately removed from the client workspace, but won't be deleted from the depot until changelist 40 is committed with **p4 submit**.

Related Commands

To open a file for add	**p4 add**

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

p4 depot

Synopsis

Create or edit a depot specification.

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

Syntax

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

Description

The Perforce versioning service stores files in shared repositories called depots. By default, there is one depot on every Perforce installation, and its name is `depot`.

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

Specifying the `-t` option creates a depot spec that is custom-made for the depot type you specify. For example, the following command:

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

Creates the following depot spec:

```
Depot:       mystreamdepot
Owner:       user
Date:        2015/07/21 15:57:50
Description: Created by user.
Type:        stream
StreamDepth: //mystreamdepot/1
Map:         mystreamdepot/...
```

You can edit this spec or you can pipe the output stream to the **`p4 depot`** command as follows to create the depot:

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

A depot created with **`p4 depot`** is not physically created on disk until files have been added to it with **`p4 add`**. Users are not able to access a new depot created with **`p4 depot`** until permission to access the depot is granted with **`p4 protect`**.

Form Fields

Field Name	Type	Description
`Depot:`	Read-Only	The depot name as provided in **`p4 depot`** ***`depotname`***.
`Owner:`	Writable	The user who owns the depot. By default, this is the user who created the depot. The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Description:`	Writable	A short description of the depot's purpose. Optional.
`Type:`	Writable	`local`, `remote`, `spec`, `stream`, `unload`, `archive` or `tangent`. A local depot is writable, and is the default depot type. Files reside in the server's root directory and are managed directly by the server. A stream depot is also writable, but contains *streams*, a type of branch that includes hierarchy and policy. A remote depot references files that reside on other servers, and cannot be written to. See "Working with Remote Depots" on page 100 for more information. The spec depot, if present, automatically archives edited forms. See "Working with Spec Depots" on page 101 for more information. The unload depot, if present, holds infrequently-used metadata (about old client workspaces and labels) that has been unloaded with the **`p4 unload`** command. For more information, see the *Helix Versioning Engine Administrator Guide: Fundamentals*. An archive depot is used in conjunction with the **`p4 archive`** and **`p4 restore`** commands to facilitate offline (or near-line) storage of infrequently-accessed revisions, typically large binaries. A tangent depot defines a read-only location that holds tangents created by the **`p4 fetch -t`** command. The tangent depot named `tangent` is automatically created by **`p4 fetch -t`** if one does not already exist.
`Address:`	Writable	If the `Type:` is `remote`, the address should be the `P4PORT` address of the remote server. If the `Type:` is `local` or `spec`, this field is ignored.

Field Name	Type	Description
`Suffix:`	Writable	If the `Type:` is `spec`, this field holds an optional suffix for generated paths to objects in the spec depot. See "Working with Spec Depots" on page 101 for more information. The default suffix is `.p4s`. You do not need a suffix to use the spec depot, but supplying a file extension to your Perforce server's versioned specs enables users of GUI client software to associate Perforce specifications with a preferred text editor. If the `Type:` is `local` or `remote`, this field is ignored.
`StreamDepth:`	Writable	For stream depots, the optional depth to be used for stream paths in the depot, where depth specifies the number of slashes following the depot name of a stream. See "Working with Stream Depots" on page 101. This field is used when streams are being created. The default is 1, matching the traditional stream name. You cannot update this value once streams or archive data exist in a depot.
`Map:`	Writable	If the `Type:` is `local`, `spec`, or `archive`, set the map to point to the relative location of the depot subdirectory. The map must contain the `...` wildcard; for example, a `local` depot `new` might have a `Map:` of `new/...`. If the `Type:` is `remote`, set the map to point to a location in the remote depot's physical namespace, for example, `//depot/new/rel2/...`. This directory will be the root of the local representation of the remote depot. For more information, see "Providing Map Information" on page 100.
`SpecMap:`	Writable	For spec depots, an optional description of which specs should be saved, expressed as a view.

Options

`-d` *depotname*	Delete the depot *depotname*. The depot must not contain any files; the Perforce superuser can remove files with **`p4 obliterate`**. If the depot is `remote`, **`p4 obliterate`** must still be run: no files are deleted, but any outstanding client or label records referring to that depot are eliminated. The name of a depot may not be the same as the name of a branch, client workspace, or label.

`-f`	By default, when you delete a depot, the directory specified by the `Map:` field (typically under P4ROOT) must be empty. Use the `-f` option to remove all files even if the directory is not empty.
`-i`	Read a depot specification from standard input.
`-o` *depotname*	Write a depot specification to standard output.
`-t` *type*	The type of the depot: `local`, `remote`, `spec`, `stream`, `unload`, `archive` or `tangent`.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Providing Map Information

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

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

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

Working with Remote Depots

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

Remote depots are accessed by a virtual user named `remote` (or, if configured, by the service user configured for the service that originates the request), and by default, all files on any Perforce

installation can be accessed remotely. To limit or eliminate remote access to a particular server, use **p4 protect** to set permissions for user **remote** (or the accessing site's service user) on that server.

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

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

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

Neither service users nor the virtual **remote** user consume Perforce licenses.

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

Working with Spec Depots

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

//specdepotname/formtype/objectname[*suffix*]

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

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

After you have created the spec depot, use **p4 admin updatespecdepot** to pre-populate it with current set of client, depot, branch, label, typemap, group, user, and job forms. For more information about setting up a spec depot, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

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

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

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

Working with Stream Depots

You use a **stream** depot to store the streams you create and work with. You need to provide the following information when creating a stream depot: name, owner, date, type, and stream depth.

By default, the stream depth is 1, which specifies that streams are stored one level below the depot name. For example:

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

You can specify a different depth when you create the stream depot. This determines the subdirectory level where you want to root your streams. For example, if you specify 3 for `StreamDepth`, your depot might look like one of these:

```
//myStreamDepot/A/B/myStream1
//myStreamDepot/A/B/myStream2
//myStreamDepot/A/C/myStream3
```

You might want to increase the stream depth to identify the project to which your streams belong or to identify the organizational group that is responsible for those streams. In general, it is best to keep the depth as low as possible.

Observe the following limitations with respect to stream depth:

- All streams in a stream depot must be rooted at the same subdirectory level, which is given by the value of `StreamDepth`.
- You may not change the name of a stream.

  For example if a stream is named as follows:

  ```
  //myStreamDepot/A/B/myStream1
  ```

  You cannot rename it either to this (changing the depth of the stream):

  ```
  //myStreamDepot/A/myStream1
  ```

  or this:

  ```
  //myStreamDepot/A/B/myNewStream
  ```

Related Commands

To list all depots known to the Perforce versioning service	**p4 depots**
To populate a new depot with files	**p4 add**
To add mappings from an existing client workspace to the new depot	**p4 client**
To remove all traces of a file from a depot	**p4 obliterate**

To limit remote access to a depot	**p4 protect**
To archive files into an archive depot	**p4 archive**
To restore files from an archive depot	**p4 restore**

p4 depots

Synopsis

Display a list of depots known to the Perforce versioning service.

Syntax

p4 [*g-opts*] depots

Description

Lists all the remote and local depots known to the Perforce service, 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.

If a depot is excluded in the protections table for a given user, the user denied access will no longer see the depot in the output of **p4 depots**.

`Operator` users may now run the **p4 depots** command.

Options

g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

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

p4 describe

Synopsis

Provides information about changelists and the changelists' files.

Syntax

p4 [*g-opts*] describe [*-doptions*] [-s -S -f -O -I] *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. By default, this command does not expand keywords because keyword differences tend to obscure real differences.

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

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

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

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

Options

-doptions	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" on page 108 for an option listing.
-f	Force the display of descriptions for restricted changelists. This option requires **admin** permission.
-O	If a changelist was renumbered on submit, and you know only the original changelist number, use **-O** and the original changelist number to describe the changelist.
-I	Specifies that the changelist number is the Identity field of a changelist.
-s	Display a shortened output that excludes the files' diffs.

`-S`	Display files shelved for the specified changelist, including diffs of those files against their previous depot revision.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

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

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

Related Commands

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

p4 diff

Synopsis

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

Syntax

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

Description

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

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

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

By default, the diff routine used is the one built into the **p4** command-line application. To change this diff routine to an external diff program, set the P4DIFF environment variable to point to the new program.

Options

-*doptions*	Pass options to the underlying diff routine (see "Usage Notes" on page 110 for details).
-f	Force the diff (if no revision is specified, against the head revision), even when the client file is not open for **edit**.
-m *max*	Limit output to diffs (or status) of only the first *max* files, unless the **-s** option is used, in which case the **-m** option is ignored.
-Od	Limit output to only those files that differ.
-*soptions*	Pass display options to the underlying diff routine (see "Usage Notes" on page 110 for details).
-t	Diff the revisions even if the files are not of type **text**.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- The `-d`*`options`* supported by **`p4 diff`** are:

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

- The `-s`*`options`* supported by **`p4 diff`** are:

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

Option	Name
`-sr`	Show only the names of opened files in the client workspace that are identical to the revision in the depot.

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

  `p4 diff -dub` ***`file`***

  specifies a unified diff that ignores changes in whitespace.

- 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.

- If a revision is provided in the file specification, the **`-s`** options compare the file(s) regardless of whether they are opened in a changelist or the workspace has been synced to the specified revision.

Examples

`p4 diff file#5`	Compare the client workspace revision of file **`file`** to the fifth depot revision.
`p4 diff @1999/05/22`	Compare all open files in the client workspace to the revisions in the depot as of midnight on May 22, 1999.
`p4 diff -du file`	Run the comparison on file **`file`**, displaying output in a format suitable for the **`patch(1)`** utility.
`p4 diff -sr | p4 -x - revert`	Revert all open, unchanged files. This differs from **`p4 revert -a`** (revert all unchanged files, where resolving a file, even if no changes are made, counts as a change), in that it reverts files whose workspace content matches the depot content, including resolved files that happen to be identical to those in the depot. The first command shows all open, unchanged files. The second command (running **`p4 -x`** and taking arguments, one per line, from standard input, abbreviated as "-") reverts each file in that list. (This is the UNIX version of this command; it uses a pipe. Most operating systems have some equivalent way of performing these operations in series). For more information about the **`-x`** option to **`p4`**, see "Global Options" on page 571.

Related Commands

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

p4 diff2

Synopsis

Compare one set of depot files to another and display the difference. The source and the target files to be compared can be specified on the command line or through a branch view.

Syntax

```
p4 [g-opts] diff2 [-doptions] [-Od -q -t -u] file1[rev] file2[rev]
p4 [g-opts] diff2 [-doptions] [-Od -q -t -u] -b branch [[fromfile[rev]] tofile[rev]]
p4 [g-opts] diff2 [-doptions] [-Od -q -t -u] [-S stream] [-P parent]
                [[fromfile[rev]] tofile[rev]]
```

Description

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

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

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 shared Perforce service, and always uses the service'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 displays the *summary* as `<none>`.

Options

`-b branch from[rev] to[rev]`	Use a branch mapping to diff files in two branched codelines. The files that are compared can be limited by file patterns in either the ***from*** or ***to*** file specifications.
`-doptions`	Runs the diff routine with one of a subset of the standard UNIX diff options. See "Usage Notes" on page 114 for a listing of these options.
`-Od`	Limit output to only those files that differ.
`-q`	Quiet diff. Display only the header; if *file1* and *file2* are identical, display only "***file1* - no differing files**" as the output.
`-S stream [-P parent]`	Diff a stream with its parent. To diff the stream with a stream other than its configured parent, specify `-P`.
`-t`	Diff the file revisions even if the file(s) are not of type **text**.
`-u`	Generate unified output format, showing added and deleted lines with sufficient context for compatibility with the **patch(1)** utility. Only those files that differ are included. File names and dates remain in Perforce syntax.
g-opts	See "Global Options" on page 571.

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 options supported by **p4 diff2** are:

Option	Name
`-dn`	RCS output format, showing additions and deletions made to the file and associated line ranges.
`-dc[num]`	context output format, showing line number ranges and *num* lines of context around the changes.
`-ds`	summary output format, showing only the number of chunks and lines added, deleted, or changed.

Option	Name
`-du[num]`	unified output format, showing added and deleted lines with *num* lines of context, in a form compatible with the `patch(1)` utility.
`-dl`	ignore line-ending (CR/LF) convention when finding diffs
`-db`	ignore changes made within whitespace; this option implies `-dl`.
`-dw`	ignore whitespace altogether; this option implies `-dl`.

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

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

  specifies a unified diff that ignores changes in whitespace.

- The header line of a unified diff produced with the `-du` option for `patch(1)` use displays the diffed files in 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. Because the branch mapping maps *`fromfiles`* to *`tofiles`*, why would you specify both *`fromfile`* and *`tofile`* file patterns? You wouldn't, but this syntax allows you to specify a *`fromfile`* file pattern and a *`tofile`* revision, or a *`fromfile`* revision and a *`tofile`* file pattern.
- RCS keywords within files are not expanded with **`p4 diff2`**.

Examples

`p4 diff2 -ds file#1 file`	Compare the first revision of file `file` to its head revision, and display a summary of what chunks were added to, deleted from, or changed within the file.
`p4 diff2 file@34 file@1998/12/04`	Diff the revision of `file` that was in the depot after changelist 34 was submitted against the revision in the depot at midnight on December 4, 1998.
`p4 diff2 //depot/rel1/... //depot/rel2/...#4`	Compare the head revisions of all files under `//depot/rel1` to the fourth revision of all files under `//depot/rel2`.

`p4 diff2 //depot/rel1/* //depot/rel2/...`	Not allowed. The wildcards in each file pattern must match.
`p4 diff2 -b branch2 //depot/rel2/...#2 @50`	Compare the second revision of the files in `//depot/rel2/...` to the files branched from it by branch mapping `branch2` at the revision they were at in changelist 50.

Related Commands

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

p4 dirs

Synopsis

List the immediate subdirectories of specified depot directories.

Syntax

p4 [*g-opts*] dirs [-C -D -H] [-S *stream*] [-i]*depot_directory*[*revRange*]...

Description

Use **p4 dirs** to find the immediate subdirectories of any depot directories provided as arguments. Any directory argument must be provided in depot or local syntax and must end with the * wildcard.

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** option.

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.

Note	This command is meant to be used in scripts, and it is unlikely that you'll use it from the command line.

Options

-C	Display only those directories that are mapped through the current client view.
-D	Include subdirectories that contain only deleted files. By default, these directories are not displayed.
-H	Include only those directories that contain files on the current client workspace's **p4 have** list.
-S *stream*	List directories mapped for the specified stream.
-i	Ignore the case of the directory argument when listing directories in a case-sensitive server.
g-opts	See "Global Options" on page 571.

Usage Notes

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

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

Examples

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

Related Commands

To list all the files that meet particular criteria	**`p4 files`**
To list all depots known to the Perforce versioning service	**`p4 depots`**

p4 diskspace

Synopsis

Display disk space information on the server.

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

Syntax

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

Description

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

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

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

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

If no arguments are specified, disk space information is displayed for all objects.

By default, Perforce rejects commands when free space on the filesystems housing the P4ROOT, P4JOURNAL, P4LOG, or TEMP fall below 10 megabytes. To change this behavior, set the `filesys.P4ROOT.min` (and corresponding) configurables to your desired limits.

If the user account that runs the Perforce versioning service is subject to disk quotas, the `filesys.*.min` configurables reflect those quotas, regardless of how much physical space actually remains on the filesystem(s) in question.

Note

Server releases prior to 16.1 included reserved space in the number displayed for used space in the Size column for Unix filesystems. This has been changed so that the size given now excludes reserved space.

Options

depot	Report disk space available for filesystem holding the specified *depot*.
journalPrefix	Report disk space available for filesystem holding a non-standard journal location.
P4JOURNAL	Report disk space available for filesystem holding P4JOURNAL.
P4LOG	Report disk space available for filesystem holding P4LOG (server log).

`P4ROOT`	Report disk space available for filesystem holding `P4ROOT`.
`TEMP`	Report disk space available for filesystem holding temporary files. If not defined, uses `P4ROOT` on Windows, and `/tmp` on Unix.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To configure Perforce's behavior when diskspace is low — **`p4 configure`**

p4 edit

Synopsis

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

Syntax

```
p4 [g-opts] edit [-c changelist] [-k -n] [-t type] [--remote=remote] 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 committed with **p4 submit**.

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

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

By default, the specified files are added to the default changelist. Use **-c** to specify a different changelist. (Or use the **p4 change** command to move files from the default changelist to a numbered changelist.)

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

Options

-c *changelist*	Opens the files for edit within the specified changelist. If this option is not provided, the files are linked to the default changelist.
-k	Keep existing workspace files; mark the file as open for edit even if the file is not in the client view. Use **p4 edit -k** only in the context of reconciling work performed while disconnected from the shared versioning service.
-n	Preview which files would be opened for edit, without actually changing any files or metadata.
--remote=*remote*	Opens the file for edit in your personal server, and additionally — if the file is of type **+l** — takes a global exclusive lock on the file in the shared server from which you cloned the file. For more information, see the section Support for exclusive locking in the Fetching and Pushing chapter of *Using Helix for Distributed Versioning*.

`-t type`	Stores the new file revision as the specified type, overriding the file type of the previous revision of the same file. To forcibly re-detect a file's filetype (that is, to assign a file type as if the file were being newly added) upon editing a file, use **`p4 edit -t auto`**. See "File Types" on page 585 for a list of file types.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Because **`p4 edit`** turns local OS `write` permissions on for the specified files, this command should be given before the file is actually edited. The process is:

1. Use **`p4 edit`** to open the file in the client workspace,
2. Edit the file with any editor,
3. Submit the file to the depot with **`p4 submit`**.

To edit an older revision of a file, use **`p4 sync`** to retrieve the previously stored file revision into the client workspace, and then **`p4 edit`** the file. Because this file revision is not the head revision, you must use **`p4 resolve`** before the file can be stored in the depot with **`p4 submit`**.

By default, Perforce does not prevent users from opening files that are already open; its default scheme is to allow multiple users to edit the file simultaneously, and then resolve file conflicts with **`p4 resolve`**. To determine whether or not another user already has a particular file opened, use **`p4 opened -a file`**.

If you need to prevent other users from working on files you've already opened, you can either use the **`p4 lock`** command (to allow other users to edit files you have open, but prevent them from submitting the files until you first submit your changes), or you can use the `+l` (exclusive-open) filetype to prevent other users from opening the files for edit at all.

In older versions of Perforce, **`p4 edit`** was called **`p4 open`**.

Examples

`p4 edit -t text+k doc/*.txt`	Opens all files ending in `.txt` within the current directory's `doc` subdirectory for `edit`. These files are linked to the default changelist; these files are stored as type `text` with keyword expansion.

p4 edit -t +l //depotname/...	Implements pessimistic locking (exclusive-open) for all files in a depot. After this changelist is submitted, only one user at a time will be able to edit files in the depot named *depotname*.
p4 edit -c 14 ...	Opens all files anywhere within the current working directory's file tree for **edit**. These files are examined to determine whether they are **text** or **binary**, and changes to these files are linked to changelist 14.
p4 edit status%40jan1.txt	Open a file named **status@jan1.txt** for edit. For details about how to specify other characters reserved for use as Perforce wildcards, see "Limitations on characters in filenames and entities" on page 578.

Related Commands

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

p4 export

Synopsis

Extract journal or checkpoint records.

Syntax

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

Description

This command reports checkpoint and journal metadata from a Perforce server. With no options, the records are reported in tagged form.

Some fields are added to the tagged output to indicate either transactional consistency, or to indicate the end of the journal.

To filter database tables out of the exported data, use the `-T` option with a list of tables whose data you wish to exclude. To exclude data from multiple tables, separate the table names by spaces or commas. The table names must begin with "`db.`", following the naming convention used for database files in the server root directory. If you separate the table exclusion list with spaces, you must enclose the list in quotes.

Options

`-c`	Specifies a checkpoint number or position token of the form *`checkpointnum#byteoffset`*.
`-f`	Format the output so that non-textual datatypes are formatted appropriately.
`-F` *`filter`*	Limit output to records that match the specified *`filter`* pattern. For example, `-F "table = db.configure"`
`-j`	Specify a journal number or position token of the form *`journalnum/byteoffset`*.
`-J` *`prefix`*	Specifies a filename prefix for the journal, such as that used with **`p4d -jc`** ***`prefix`***
`-l` *`lines`*	Limit output to the specified number of *`lines`* of journal records.

`-P` *`filterpattern`*	Limit output to records that match the specified filter pattern. Multiple filter patterns can be specified with multiple `-P` options. Each *`filterpattern`* begins with two characters and a colon, and specifies either a client filter or a depot filter, as well as whether the pattern is to be included or excluded, using the syntax: • `-Pic://client/`*`pattern`* - client records to include • `-Pxc://client/`*`pattern`* - client records to exclude • `-Pif://depot/`*`pattern`* - depot records to include • `-Pxf://depot/`*`pattern`* - depot records to exclude The first character specifies whether the records are included or excluded ("`i`" or "`x`"), the second character specifies whether the records are client workspace-related or depot-file related ("`c`" or "`f`"), the colon is a separator, and the remainder of the *`filterpattern`* denotes either a client workspace view or a depot file path. The mechanism by which this filtering is implemented is the same as that which is used by the `ClientDataFilter:` and `RevisionDataFilter:` fields in the **`p4 server`** form.
`-r`	Display raw journal output; this argument applies to journals only.
`-T` *`tableexcludelist`*	Supply a list of database tables (for example, `db.have` and `db.client`) to exclude from export. Limit output to records that match the specified *`filter`* pattern. For example, `-T db.have,db.client` or `-T "db.have db.client"`
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- Compressed journals or checkpoints are not supported.

Examples

`p4 export -T "db.have db.working"`	Run **`p4 export`**, but ignore records in the `db.have` and `db.working` tables.

Related Commands

To replicate metadata from one server to another	**p4 replicate**
To pull journal records (and file content) from a master server to a replica server	**p4 pull**

p4 fetch

Synopsis

Copy files from a remote server into your local server.

Syntax

```
p4 [g-opts] fetch [-r remotespec] [-m depth] [-v -k] [-n | -t] [-Ox] [-S stream | filespec]
p4 [g-opts] fetch [-r remotespec] [-v] [-n] [-Ox] -s shelf
```

Description

The **p4 fetch** command copies the following items from the specified remote server to the local server:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes associated with the changelists, but only if the job that is linked by the fix is already present in the local server. If it is not, then the fix is not copied.
- all integration records that describe integrations to the files being fetched

A fetch is only allowed if the files being fetched fit cleanly into the server to which you're currently connected, building cleanly on a shared common history.

The second form of the command copies a shelved changelist, rather than one or more submitted changelists, in which case conflicts do not arise; the result is a new shelved change in the local server.

If there are no conflicts, the files and their changelists become new submitted changelists in the local server. Conflict handling is configurable, using the **-t** option. If **-t** is not specified, and there are any conflicts or gaps, the fetch is rejected. The **-t** option specifies that the conflicting changelists should be relocated to the tangent depot, and the remote work is then fetched. After the fetch completes, use **p4 resubmit** to resubmit the conflicting local changes.

When the changelists are added to the local server, they are given newly assigned change numbers but they retain the same description, user, date, type, workspace, and set of files. When the files are added to the local server, they are kept in their same changelists, as new revisions starting after the current head. The new revisions retain the same revision number, file type, action, date, timestamp, digest, and file size. Although the changelists are new submitted changelists in the local server, none of the submit triggers are run in the local server.

Note that once a particular revision has been copied to a local server, using **p4 attribute -f** to change the attributes on that revision will only affect the revision on that server, not on any other server to which it may have been copied.

Typically, the **p4 fetch** command specifies a remote spec, and the `DepotMap` field in the remote spec specifies which files are to be fetched. The **p4 fetch** command may also specify a filespec argument to further restrict the files to be fetched. If the remote spec uses differing patterns for the local and remote sides of the `DepotMap`, the filespec argument, if provided, must specify the files using the local filename syntax. If a particular changelist includes some files that match the filespec, and other files that do not, then only the matching files are included in the fetch. In order to ensure that a partial changelist is not fetched, an appropriate filespec should be specified (for example, `//...@change,#head`).

p4 fetch behaves differently if the remote spec's `ArchiveLimits:` field is set. This field regulates how many, if any, revisions of file archives are stored on the server you fetch to. For more information, see the section "Configure server to limit storage of archive revisions" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.

When **p4 fetch** copies integration records, they are adjusted in the local server to reflect the resulting changelist numbers and revision numbers of the local server. In order to fetch a set of files, you must have read access to those files in the remote server, and you must have write access to those same files in the local server; your local userid is used as the userid at the remote server and you must already be logged in to both servers prior to running the **p4 fetch** command.

By default, a server does not accept fetch requests from another server. In order to fetch from a server, an administrator of that server must enable fetching by setting `server.allowpush` to `1`.

The **p4 fetch** command is atomic: either all the specified files are fetched, or none of them are fetched.

Files with the filetype modifiers `+k`, `+l`, or `+S` have some special considerations. Files of type `+k` have their digests cleared when fetched. This means certain cross-server merge conflicts are not detected. To re-generate the digests after the fetch, use the **p4 verify** command. When fetching files of type `+l`, the new files are added to the server even if the files are currently open by a pending changelist in the server. When fetching files of type `+S`, old archives which exceed the specified limit are not purged by the fetch command.

The value of the `rpl.checksum.change` configurable determines the level of verification performed for the **p4 fetch** command. See "Configurables" on page 595.

Note	**p4 fetch** automatically performs a **p4 sync** as part of its operations.

Triggering on fetches

The following push trigger types may be invoked during the execution of the **p4 fetch** command:

- The `push-submit` trigger can customize processing during the phase of the **p4 fetch** command when metadata has been transferred but files have not yet been transferred.
- The `push-content` trigger can customize processing during that phase of the **p4 fetch** command when files have been transferred but their contents have not yet been committed.
- The `push-commit` trigger can do any clean up work or other post processing after changes have been committed by the **p4 fetch** command.

For more information, see the section "Triggering on pushes and fetches" in the scripting chapter of *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

With no options specified **p4 fetch** fetches files from the remote server named origin.

`-k`	Suppresses automatic sync of workspace to the head revision.
`-m depth`	Specifies that Perforce should perform a shallow fetch; only the last number of revisions specified in *depth* are fetched.
`-n`	Performs correctness checks but does not fetch any files or changelists from the remote server. In particular, Perforce checks for conflicts between work that's been done in the local server and working you're trying to fetch from the remote server. This tells you whether your personal server is up to date with the remote server.
`-Oc`	When set, the **p4 fetch** command outputs information about every changelist. The **-v** option must be set for this to take effect.
`-Of`	When set, the **p4 fetch** command outputs information about every file in every changelist. The **-v** option must be set for this to take effect.
`-Oi`	When set, the **p4 fetch** command outputs information about every integration in every file in every changelist. The **-v** option must be set for this to take effect.
`-r remotespec`	Specifies a remote spec containing the address of the remote server, and a file mapping which is to be used to re-map the files when they are fetched from the remote server. See also **p4 remote**.
`-s`	Specifies a shelved changelist to be fetched, instead of one or more submitted changelists.For more information, see the section "Fetch and push a shelved changelist" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.
`-t`	Specifies that conflicting changelists should be relocated to the tangent depot, automatically creating that depot if it does not exist. The relocated changes can then be resubmitted using **p4 resubmit**. If you don't specify a remote server with the **-r** option, the remote server defaults to **origin**. **Note** **p4 fetch -t** requires **admin** permission.
`-S stream`	Specifies a particular stream to fetch. If you specify a stream you cannot also specify a file or files.

`-v`	Enables verbose mode, which provides diagnostics for debugging. With verbose mode enabled, you can refine and control the precise level of verbosity. Specifically, you can indicate whether you want information about: • every changelist fetched (with the `-Oc` option) • every file in every changelist fetched (with the `-Of` option) • every integration of every file in every changelist fetched (with the `-Oi` option) You can specify any combination of these options, but must always include the `-O`. The default is to display information about every changelist.
`filespec`	Specifies which files to fetch. If you specify a file or files you cannot specify a stream with the `-S` option.

Usage Notes

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

Examples

`p4 fetch -m 5 -r dev`	Fetch the most recent `5` revisions of each file in the `dev` remote spec.

Related Commands

To push to a remote server	**`p4 push`**

p4 filelog

Synopsis

Print detailed information about files' revisions.

Syntax

p4 [*g-opts*] filelog [-c *change*] [-h -i -l -L -t -p -s] [-m *max*] *file*[*revRange*]

Description

p4 filelog describes each revision of the files provided as arguments. At least one file or file pattern must be provided as an argument. If the file specification includes a revision range, only the specified revisions are listed.

By default, the output consists of one line per revision in reverse chronological order. The format of each line is:

```
... #rev change chnum action on date by user@client (type) 'description'
```

where:

- *rev* is the revision number;
- *chnum* is the number of the submitting changelist;
- *action* is the operation the file was open for: **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** option is used).
- *user* is the name of the user who submitted the revision;
- *client* is the name of the client workspace from which the revision was submitted;
- *type* is the type of the file at the given revision; and
- *description* is the first 30 characters of the corresponding changelist's description.

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

Options

`-c change`	Display only files submitted at the specified changelist number.
`-h`	Display file content history instead of file name history. The revisions that are listed include revisions of other files that were branched/copied (using **p4 integrate** and **p4 resolve -at**) to the specified revision. Revisions that were replaced by copying or branching are not displayed, even if they are in the history of the specified revision.
`-i`	Follow file history across branches. If a file was created by integration (**p4 integrate**), Perforce describes the file's revisions and displays the revisions of the file from which it was branched (back to the branch point of the original file). File history inherited by renaming (**p4 move**) is always displayed, regardless of whether or not the `-i` option is used.
`-l`	List long output, with the full text of each changelist description.
`-L`	List long output, with the full text of each changelist description truncated at 250 characters.
`-m max`	List only the first *max* changes per file output.
`-p`	When used with the `-h` option, do not follow content of promoted task streams. This option is useful when there are many child task streams branched from the supplied *file* argument.
`-s`	Display a shortened form of output by ignoring non-contributory integrations (for example, integrations involving "branch into" or copy into" operations are not displayed)
`-t`	Display the time as well as the date.
g-opts	See "Global Options" on page 571.

Usage Notes

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

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

- Old revisions of temporary object files (file type modifier `+S`*`n`*) are displayed with an action of `purge`.

Examples

`p4 filelog //depot/proj1/...`	Display the revision history for every file under the depot's `proj1` directory.
`p4 filelog file1.c@100,@120`	Display the revision history for `file1.c` from changelists 100 through 120.
`p4 filelog file1.c#have,#head`	If you do not have the latest revision of `file1.c`, display revision history since your last sync.
`p4 filelog file1.c file1.h`	Show the revision history for files `file1.c` and `file1.h`, which reside locally in the current working directory.

Related Commands

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

p4 files

Synopsis

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

Syntax

```
p4 [g-opts] files [-a -A -e] [-i][-m max] file[revRange]...
p4 [g-opts] files -U unloadfile...
```

Description

This command lists each file that matches the file patterns provided as arguments. If a revision specifier is given, the files are described at the given revision. One file is listed per line, and the format of each line is:

```
depot-file-location#rev - action change changelist (filetype)
```

where:

- *depot-file-location* is the file's location relative to the top of the depot,
- *rev* is the revision number of the head revision of that file,
- *action* is the action taken at the head revision: `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`,
- *changelist* is the number of the changelist in which the revision was submitted, and
- *filetype* is the 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. If a file pattern on the command line is given in client syntax, only files in the client workspace are shown.

Options

`-a`	For each file, list all revisions within a specified revision range, rather than only the highest revision in the range.
`-A`	Limit output to files in archive depots.
`-e`	Exclude deleted, purged, or archived files; the files that remain are those available for syncing or integration.
`-i`	Ignore the case of the file argument when listing files in a case sensitive server.

`-m max`	Limit output to the first *max* files.
`-U unloadfile`	List only files in the unload depot. See **p4 unload** for details.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- The specified revision can be a revision range; in this case, only those files with revisions within the specified range are listed, and by default, only the highest revision in that range is listed. (To display information for all files within a revision range, use **p4 files -a**.)
- Because the output of **p4 files** can be quite large when called with highly non-restrictive file arguments (for example, **p4 files //depot/...** prints information about all the files in the depot), it may be subject to a `maxresults` limitation as set in **p4 group**.

Examples

p4 files //depot/...	Provides information about all files in the depot.
p4 files //clientname/...	Provides information about all depot files visible through the client view.
p4 files @2011/03/10	Provides information about all depot file revisions that existed on March 10, 2011.
p4 files @2011/03/31:08:00,@2011/03/31:17:00	Lists all files and revisions changed during business hours on March 31, 2011.
p4 files //depot/proj2/...@p2lab	Lists files and revisions under the directory `//depot/proj2/...` tagged by label `p2lab`.
p4 files //depot/file.c	Show information on the head revision of `//depot/file.c`. (that is, the *highest* revision in the implied range of `#1,#head`)
p4 files -a //depot/file.c	Show information on every revision of `//depot/file.c` (that is, *all* revisions in the implied range of `#1,#head`)
p4 files -A //arch/depot/proj/...	If an administrator has used **p4 archive** to transfer `//depot/proj/...` to an archive

	depot named `arch`, displays information about the files in the archived project.

Related Commands

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

p4 fix

Synopsis

Link jobs to the changelists that fix them.

Syntax

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

Description

The **p4 fix** command links jobs (descriptions of work to be done) to a changelist (a set of changes to files that does the work described by a job).

If the changelist has not yet been submitted, the job appears on the **p4 submit** or **p4 change** form for the changelist to which it's linked, and under normal circumstances, the status of the job is changed to **closed** when the changelist is submitted. If the changelist has already been submitted when you run **p4 fix**, the job's status is changed to a default status (typically **closed**) immediately.

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

Because described work can be fixed over multiple changelists, one job can be linked to multiple changelists. Because a single changelist might fix ten bugs, multiple jobs can be linked to the same changelist. You can do this in one command execution by providing multiple jobs as arguments to **p4 fix**.

Options

-c *changelist*	The changelist to mark as fixed.
-d	Delete the fix record for the specified job at the specified changelist. The job's status will not change.
-s *status*	Upon submission of the changelist, change the job's status to *status*, rather than the default value **closed** (or some other value as defined in the **Presets:** of field 102 of the **p4 jobspec** form). If the changelist to which you're linking the job been **submitted**, the status value is immediately reflected in the job's status. If the changelist is **pending**, the job status is changed on submission of the changelist, provided that the **-s** option is also supplied to **p4 submit** and the desired status appears next to the job in the **p4 submit** form's **Jobs:** field. To leave a job unchanged, use the special status of **same**.

g-opts	See "Global Options" on page 571.

Usage Notes

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

- Because the format of jobs can be changed from site to site, it is possible that the jobs on your system no longer have a `Status:` field. If so, you can still link jobs to changelists with **`p4 fix`**, but Perforce will not change any of the job fields' values when the changelist is submitted.
- You can change a fixed or unfixed job's status at any time by editing the job with **`p4 job`**.
- Another way to fix (or unfix) a job is to add it to (or delete it from) the `Jobs:` field of an unsubmitted changelist's **`p4 submit`** or **`p4 change`** form.
- You can't **`p4 fix`** a job to the default changelist; instead, add the job to the `Jobs:` field of the default changelist's **`p4 submit`** form when submitting it to the depot.
- If you use **`p4 fix -s status`** on a job, and then use the `-s` option with **`p4 submit`** or **`p4 change`**, the `Jobs:` field of the changelist's form will also require a status value (the default value being the one specified by **`p4 fix -s status`**). The job(s) will be assigned the specified *status* upon successful submission of the changelist. If no status value is specified in the form, the error message:

  ```
  Wrong number of words for field 'Jobs'.
  ```

  is displayed.

  `p4 fix -s status`, **`p4 submit -s`**, and **`p4 change -s`** are intended for use in conjunction with defect tracking systems.

  Under normal circumstances, end users do not use these commands, and use **`p4 submit`** and **`p4 change`** without the `-s` option. In this case, only the job number is required in the `Jobs:` field, and each job's status is set to a default value (typically `closed`) on completion of the submit.

Examples

`p4 fix -c 201 job000141 job002034`	Mark two jobs as being fixed by changelist 201. If changelist 201 is still `pending`, the jobs' status is changed to `closed` when the changelist is submitted.
`p4 fix -c 201 -s suspended job002433`	Mark `job002433` as `suspended`, rather than `closed`, when changelist 201 is submitted.

	Requires use of the **-s** option with **p4 submit**.

Related Commands

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

p4 fixes

Synopsis

List jobs and the changelists that fix them.

Syntax

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

Description

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

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

```
jobname fixed by change changelist on date by user (status)
```

You can limit the listed fixes by combining the following options when calling **p4 fixes**:

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

Note	This command now runs in lockless mode if **db.peeking** is set to r.

Options

-c *changelist*	Limit the displayed fixes to those that include the specified changelist.
-i	Include fixes made by changelists that affected files integrated into the specified files.
-j *jobname*	Limit the displayed fixes to those that include the specified job.
-m *max*	List only the first *max* fixes.

g-opts	See "Global Options" on page 571.

Usage Notes

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

Examples

`p4 fixes //depot/proj1/...`	Display all fixes made by submitted changelists that included any files under `//depot/proj1`.
`p4 fixes file.c`	Display all fixes made by submitted changes that included any and all revisions of `file.c`.
`p4 fixes file.c#5`	Display all fixes made by submitted changes that included revisions 1 through 5 of `file.c`.
`p4 fixes file.c#5,5`	Display only those fixes associated with the changelist in which `file.c#5` was submitted.
`p4 fixes -c 414`	Display all jobs fixed by pending or submitted changelist 414.

Related Commands

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

p4 flush

Synopsis

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

Syntax

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

Description

Warning

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

If you use **p4 flush** incorrectly, the versioning service'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**.

The **p4 flush** command 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 workspace's *have list* (which tracks which file revisions have been synced, and is managed by the Perforce service) is updated to reflect the new client workspace contents.

p4 flush performs only the *second* of these steps. Under most circumstances, this is not desirable, because a client workspace's have list should always reflect the workspace's true contents. However, if the workspace's contents are already out of sync with the have list, **p4 flush** can sometimes be used to bring the have list in sync with the actual contents. Because **p4 flush** performs no actual file transfers, this command is much faster then the corresponding **p4 sync**.

Use **p4 flush** only when you need to update the have list to match the actual state of the client workspace. The "Examples" on page 148 subsection describes two such situations.

Options

`-f`	Force the flush. Perforce performs the flush even if the client workspace already has the file at the specified revision.
`-L`	For scripting purposes, perform the flush on a list of valid file arguments in full depot syntax with a valid revision number.
`-n`	Display 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.

`-q`	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- Because **`p4 flush`** updates the have list without copying files, and **`p4 sync -f`** updates the client workspace to match the have list, **`p4 flush`** ***`files`*** followed by **`p4 sync -f files`** is almost equivalent to **`p4 sync files`**. This means that a bad flush can be almost entirely fixed by following it with a **`p4 sync -f`** of the same file revisions that were originally flushed.

  Unfortunately, this is not a complete remedy, because any file revisions that were deleted from the have list by **`p4 flush`** will remain in the client workspace even after the **`p4 sync -f`**. In this case, you will need to manually remove deleted file revisions from the client workspace.

- **`p4 flush`** is an alias for **`p4 sync -k`**.

Examples

- Ten users at the same site need to set up new, identical client workspaces from the same depot at a remote location over a slow link. The standard method calls for each user to run identical **`p4 sync`** commands, but if bandwidth is limited, 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.

    Because **`p4 flush`** moves no files across the slow link, the process can be much faster then running the same **`p4 sync`** command ten separate times.

- Joe has a client workspace called `joe` that has a `Root:` of

  ```
  /usr/joe/project1/subproj
  ```

  and a `View:` of

  ```
  //depot/joe/proj1/subproj/... //joe/...
  ```

  He decides that all the files under `/usr/joe/project1` need to be included in the workspace, and accomplishes this by using **`p4 client`** to change the `Root:` to

`/usr/joe/project1`

and the `View:` to

`//depot/joe/proj1/... //joe/...`

This keeps his current client workspace files in the same place, while extending the scope of the workspace to include other files. But when Joe runs his next **p4 sync**, he's surprised to see that Perforce deletes every non-open file in the client workspace and replaces it with an identical copy of the same file!

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

To make better use of Perforce, Joe might have performed a **p4 flush #have** instead. This would have updated his client workspace's have list to reflect the files' "new" locations without actually copying any files.

Related Commands

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

p4 fstat

Synopsis

Dump file info in format suitable for parsing by scripts.

Syntax

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

Description

The **p4 fstat** command dumps information about each file, with information for each field on a separate line. The output is best used within a Perforce API application where the items can be accessed as variables, but is also suitable for parsing by scripts.

The only argument required for the **p4 fstat** command, is the *file[rev]* argument. All other options relate to limiting the set of files operated on or controlling the amount and display of information for the selected files.

- To change the field on which output is sorted, use one of the `-Sx` options, and to reverse sort order, use the `-r` option.
- Use the `-m` *max* option to limit the output to the first *max* files.
- To filter the output on some function of the form fields (for example, all files larger than a certain size and with a certain filetype), use the `-F` *filter* option.
- To limit output to the set of fields specified in a *fields* argument, use the `-T` *fields* option. The list of field names can be separated by spaces or commas.

The head type fields, for example, `headTime`, return information for the file revision provided for the file argument. If no specific revision is given, it returns information for the head revision.

Form Fields

The fields shown will vary with the selected file.

Field Name	Description	Example/Notes
`attr-`*name*	Attribute value for name	`attr-myAttr critical`
`attrProp-`*name*	Set if `attr-`*name* is a propagating attribute	`attrProp-myAttr`
`clientFile`	Local path to file (in local syntax by default, or in Perforce syntax with the `-Op` option)	`/staff/userid/src/file.c` (or `//workspace/src/file.c` in Perforce syntax)

Field Name	Description	Example/Notes
	For files containing the special characters `@`, `#`, `*`, and `%`, the `clientFile` displays the special character.	
`depotFile`	Depot path to file . For files containing the special characters `@`, `#`, `*`, and `%`, the filename is displayed containing the ASCII expression of the character's hexadecimal value.	`//depot/src/file.c`
`movedFile`	Name in depot of moved to/from file	`//depot/src/file.c`
`path`	Local path to file	`//workspace/src/file.c`
`isMapped`	Set if the file is mapped to current client workspace	`isMapped`
`shelved`	Set if file is shelved	`shelved`
`headAction`	Action taken at head revision, if in depot	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`.
`headChange`	Head revision changelist number, if in depot	`124`
`headRev`	Head revision number, if in depot	`124`
`headType`	Head revision type, if in depot	`text`, `binary`, `text+k`, etc. (see "File Types" on page 585.)
`headCharset`	Head charset	for unicode files
`headTime`	Head revision changelist time, if in depot. Time is measured in seconds since 00:00:00 UTC, January 1, 1970	`919283152` is a date in early 1999
`headModTime`	Head revision modification time (the time that the file was last modified on the client before submit), if in depot.	`919280483` is a date in early 1999
`movedRev`	Head revision of moved file	`157`

Field Name	Description	Example/Notes
`haveRev`	Revision last synced to workspace, if on workspace	`23`
`desc`	Changelist description (if using `-e` ***`changelist`*** and if the file was part of ***`changelist`***)	A Perforce changelist
`digest`	MD5 digest of a file (requires `-Ol` option)	A 32 hexadecimal digit string. Based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.
`fileSize`	File length in bytes (requires `-Ol` option)	`63488` Based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.
`action`	Open action, if opened in your workspace	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`.
`type`	Open type, if opened in your workspace	A Perforce file type
`charset`	Open charset	(for unicode files)
`actionOwner`	User who opened the file, if open	A Perforce username
`workRev`	open revision, if opened	
`change`	Open changelist number, if opened in your workspace	`75331`
`resolved`	The number, if any, of resolved integration records	`5`
`unresolved`	The number, if any, of unresolved integration records	`2`

Field Name	Description	Example/Notes
`reresolvable`	The number, if any, of re-resolvable integration records	`1`

`otherOpen`	The number of other users who have the file open, blank if no other users have the file open	`1`, `2`, `3`… *n*, preceded by *n* records listing the users (`0` through *n*-`1`) with `otherOpen`*n*, `otherAction`*n*, and `otherLock`*n* fields as applicable. For example:
		`... otherOpen 3`
		`...... otherOpen0 user1@ws1`
		`...... otherOpen1 user2@ws2`
		`...... otherOpen2 user3@ws3`

`otherOpen`*n*	For each user with the file open, the workspace and user with the open file	`user123@workstation9`
`otherLock`	Present and set to null if another user has the file locked, otherwise not present	`otherLock`
`otherLock`*n*	For each user with the file locked, the workspace and user holding the lock	`user123@workstation9` Because only one user at a time can lock a file, if *n* is set, *n* is always `0`.
`otherAction`*n*	For each user with the file open, the action taken	one of `add`, `edit`, `delete`, `branch`, `move/add`, `move/delete`, `integrate`, `import`, `purge`, or `archive`.
`otherChange`*n*	For every changelist with the file open, the changelist	75612
`openattr-`*name*	For every changelist with the file open, the attribute value for name	`attr-`*name*
`openattrProp-`*name*	Set if `attr-`*name* is a propagating attribute	`attrProp-`*name*
`ourLock`	Present and set to null if the current user has the file locked, otherwise not present	`ourLock`
`resolveAction`*n* `resolveBaseFile`*n* `resolveBaseRev`*n*	Pending integration action, base file, base revision number,	For pending integration record information, use the `-Or` option.

Field Name	Description	Example/Notes
`resolveFromFile`*n* `resolveStartFromRev`*n* `resolveEndFromRev`*n*	from file, starting, and ending revision, respectively.	
`totalFileCount`	The number of files examined, if sorted.	Appears in the first file's output when you use the `-m` *`max`* option in conjunction with one of the `-S`*x* or `-r` sorting options.

Options

`-A` *`pattern`*	Restrict displayed attributes to those that match the specified *`pattern`*. For example, for the selected files, `-A` *`foo`*`*` displays only attributes whose name starts with *`foo`*.
`-c` *`change`*	Display only files affected after the given changelist number. This operation is much faster than using a revision range on the affected files.
`-e` *`change`*	Display only files affected by the given changelist number. This option is much faster than using a revision range on the affected files. When used with the `-Ro` option, only pending changes are considered to ensure that files opened for add are included. This option also displays the change description.
`-F` *`filter`*	List only those files that match the criteria specified by *`filter`*. See "Usage Notes" on page 157 for a discussion of filters. Filtering is not optimized with indexes for performance.
`-L`	For scripting purposes or automated reporting processes: report file information on a list of valid file arguments in full depot syntax with a valid revision number. File specifications that do not meet these requirements are silently ignored.
`-m` *`max`*	Produce fstat output for only the first *`max`* files.
`-Oa`	Output attributes set by **`p4 attribute`**.
`-Od`	Output the digest of an attribute.
`-Oe`	Output attribute values encoded as hex.
`-Of`	Output all revisions for the given files, suppressing the `other[...]` and `resolve[...]` fields.
`-Ol`	Output a `fileSize` field displaying the length of the file and a digest field for each revision. **`p4 fstat -e`** ***`shelvedChange`*** **`-Rs -Ol`** reports the file size and digest of files shelved at the specified change.

`-Op`	Display the `clientFile` in Perforce syntax, as opposed to local syntax.
`-Or`	Display pending integration record data for files open in the current workspace.
`-Os`	Shorten output by excluding client workspace data (for instance, the `clientFile` field).
`-r`	Sort the output in reverse order.
`-Rc`	Limit output to files mapped into the current workspace.
`-Rh`	Limit output to files on your have list; that is, to files synced to the current workspace.
`-Rn`	Limit output to files opened at revisions not at the head revision.
`-Ro`	Limit output to open files in the current workspace. You must use this option to get pending changes.
`-Rr`	Limit output to open files that have been resolved.
`-Rs`	Limit output to shelved files. Requires `-e` *`changelist`* option. **`p4 fstat -e` *`shelvedChange`* `-Rs -Ol`** reports the file size and digest of files shelved at the specified change.
`-Ru`	Limit output to open files that are unresolved.
`-Sd`	Sort by date.
`-Sh`	Sort by have revision.
`-Sr`	Sort by head revision.
`-Ss`	Sort by filesize.
`-St`	Sort by filetype.
`-T` *`fields`*	List only those fields that match the field names specified by *`fields`*. The list of field names can be separated by spaces or commas.
`-U`	Include files in the unload depot when displaying data. See **`p4 unload`** for details.
`g-opts`	See "Global Options" on page 571. The `-s` global option (which prefixes each line of output with a tag describing the type of output as `error`, `warning`, `info`, `text`, or `exit`) can be particularly useful when used with **`p4 fstat`**.

Usage Notes

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

Filters

Use **`p4 fstat -F`** ***`filter`*** to limit the list of files to those that meet certain criteria.

You can use logical operators on any of the form fields displayed by **`p4 fstat`**. The usual comparison operators (`=`, `>`, `<`, `>=`, and `<=`) are available. Regular expression matching is supported by the regular expression matching operator (`~=`).

The following filter expression filters for files of a certain size whose `headType` field is set to `text`.

```
-F "fileSize > 100000 & headType=text"
```

Filters used for `fstat` are case-sensitive. All alphanumeric strings (including words including embedded punctuation) separated by whitespace are indexed as words.

Spaces between search terms in a filter are treated as boolean AND operations. To find files that contain any of the key/value pairs (boolean OR), separate the terms with the "`|`" character.

Ampersands (`&`) can be used as boolean ANDs as well; the boolean operators bind in the order `&`, `|`, space (highest precedence to lowest precedence). Use parentheses to change the grouping order.

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

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

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

Date fields can be matched by expressing the filter date as ***`yyyy/mm/dd`*** or ***`yyyy/mm/dd:hh:mm:ss`***. If a specific time is not provided, the equality operator (`=`) matches the entire day.

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

The behavior of comparison operators depends on the type of field you're comparing against. All fields that `fstat` processes are text fields. The equality operator (`=`) or case-insensitive equality operator

(`~=`) matches the file if the word given as the value is found anywhere in the specified field. The relational operators are of limited use here, because they match the file if *any* word in the specified field matches the provided value. Relational operators are always case-sensitive. For example, if a changelist has a `text` field `desc` that contains the phrase `bug not fixed`, and the filter is "`desc<fixed`", the file matches the filter, because `bug<fixed`.

Examples

`p4 fstat file.c`	Displays information on `file.c`.
`p4 fstat //....c@20,@now`	Displays information on all `.c` files after the checking-in of files under changelist 20.
`p4 fstat -Os file.c`	No client workspace information lines (`clientFile`) are displayed.
`p4 fstat -Osl file.c`	No client workspace information lines are displayed, but the `fileSize` and `digest` lines are displayed.
`p4 fstat -Os -Ol file.c`	Equivalent to **`p4 fstat -Osl`**.
`p4 fstat -F "clientFile=c:\\ws\\file.c" //depot/main/...`	If a path contains backslashes, escape them with backslashes.
`p4 fstat -F "clientFile~=c:\\ws\\[Ff]ile.c" //depot/main/...`	Use the `~=` regular expression modifier to specify a regexp that matches `File.c` and `file.c`.
`p4 fstat -Ol -F "fileSize < 1024 & headType=text" //depot/main/...`	Display information on all text files under `//depot/main/...` that are smaller than 1024 bytes in length.
`p4 fstat -T 'depotFile, headRev' file.c`	Display only the `depotFile` and `headRev` fields for `file.c`.

Related Commands

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

p4 grep

Synopsis

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

Syntax

```
p4 [g-opts] grep [-a -i -n -s -t] [-v | -l | -L] [-F | -G] [-A num] [-B num] [-C num]
                 -e pattern file[revRange] ...
```

Description

The **p4 grep** command searches for lines that match a given regular expression.

By default, **p4 grep** operates on the head revision. If the file argument specifies a revision, all files as of that revision number are searched. If the file argument has a revision range, only those files selected by that revision range are searched, and the highest revision in that range is used for each file.

By default, **p4 grep** searches at most 10,000 revisions. This limit is controlled by the **dm.grep.maxrevs** configurable.

The following example shows you can find all occurrences of a whole word:

```
$ p4 grep -e "voodoo" //depot/main/myDir/...
```

Options

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

`-n`	Display a matching line number after the file revision number.
`-v`	Display files with non-matching lines.
`-l`	Display the name of each selected file from which output would have been displayed; scanning stops at the first match.
`-s`	Suppress error messages from files with more than 4096 characters in a single line. (By default, **`p4 grep`** abandons these files and reports an error)
`-t`	Treat binary files as text. (By default, only files of type text are selected for pattern matching.)
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

p4 group

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

Description

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

To delete a group, use **p4 group -d *groupname***, or call **p4 group *groupname*** and remove all the users from the resulting form. Use the `-F` option with the `-d` option to force deletion and to remove the group from the protections table and from all groups.

Form Fields

Field Name	Type	Description
`Group:`	Read-only	The name of the group, as entered on the command line.
`MaxResults:`	Writable	The maximum number of results that members of this group can access from the service from a single command. The default value is `unset`. See "Usage Notes" on page 165 for more details.
`MaxScanRows:`	Writable	The maximum number of rows that members of this group can scan from the service from a single command. The default value is `unset`. See "Usage Notes" on page 165 for more details.
`MaxLockTime:`	Writable	The maximum length of time (in milliseconds) that any one operation can lock any database table when scanning data. The default value is `unset`. See "Usage Notes" on page 165 for more details.

Field Name	Type	Description
`MaxOpenFiles:`	Writable	The maximum number of files that a member of a group can open using a single command. See "Usage Notes" on page 165 for more details.
`Timeout:`	Writable	The duration (in seconds) of the validity of a session ticket created by **`p4 login`**. The default value is 43,200 seconds (12 hours). To create a ticket that does not expire, set the `Timeout:` field to `unlimited`.
`PasswordTimeout:`	Writable	The length of time (in seconds) for which passwords for users in this group remain valid. To disable password aging, use a value of `unset`.
`LdapConfig`	Writable	The LDAP configuration to use when populating the group's user list from an LDAP query. For more information, see **`p4 ldapsync`**.
`LdapSearchQuery`	Writable	The LDAP query used to identify the members of the group. For more information, see **`p4 ldapsync`**.
`LdapUserAttribute`	Writable	The LDAP attribute that represents the user's username. For more information, see **`p4 ldapsync`**.
`Subgroups:`	Writable, multi-line	Names of other Perforce groups. To add all users in a previously defined group to the group you're presently working with, include the group name in the `Subgroups:` field of the **`p4 group`** form. Note that user and group names occupy separate namespaces, and thus, groups and users can have the same names. Every member of any previously defined group you list in the `Subgroups:` field will be a member of the group you're now defining.
`Owners:`	Writable, multi-line	Names of other Perforce users. Group owners without `super` access are permitted to administer this group, provided that they use the `-a` option. Group owners are not necessarily members of a group; if a group owner is to be a member of the group, the userid must also be added to the `Users:` field.

Field Name	Type	Description
		The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Users:`	Writable, multi-line	The Perforce usernames of the group members. Each user name must be typed on its own line, and should be indented.

Options

`-a`	Allow a (non-superuser) group owner to administer the group. The user must be listed in the `Owner:` field of the group.
`-A`	Allow a user with `admin` access to add a new group. Existing groups cannot be modified when this option is used.
`-d` *`groupname`*	Delete group *groupname*. The members of the group are affected only if their access level or `maxresults` value changes as a result of the group's deletion.
`-F` *`groupname`*	Used only with the `-d` option, forces the deletion of the specified group, and also removes the group from the protections table and from all groups.
`-i`	Read the form from standard input without invoking the user's editor. The new group specification replaces the previous one.
`-o`	Write the form to standard output without invoking the user's editor.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- Referring to a (nonexistent) user in a group definition does not create the user, nor does it consume a license; use the **`p4 user`** command to create users.
- Ticket `Timeout` and `PasswordTimeout` values for users who belong to multiple groups are calculated the same way as `maxresults` values: the largest `timeout` value for all the groups of which the user is a member (including `unlimited`, but ignoring `unset`). Users in no groups have the default ticket

`Timeout` value of `43200` and `PasswordTimeout` value of `unset`. To create a ticket that does not expire, set the `Timeout` to `unlimited`.

- If you are using the `PasswordTimeout:` field to implement password aging, a 30-day timeout is 2,592,000 seconds.

- As the number of files in the depot grows, certain commands can significantly slow down the service if called with no parameters, or if called with non-restrictive arguments. For example, **`p4 print //depot/...`** will print the contents of every file in the depot on the user's screen, and **`p4 filelog //depot/...`** will attempt to retrieve data on every file in the depot at *every revision*.

  The Perforce superuser can limit the amount of data that Perforce returns to the user by setting the `MaxResults` value for groups of users. The superuser can also limit the amount of data scanned (whether returned to the user or not) by setting the `MaxScanRows` value, and the length of time any database table can be locked in by any single operation by setting the `MaxLockTime` value. Equally, the `MaxOpenFiles` field can be set to specify the maximum number of files that a group member can open at any given time.

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

  If a user belongs to multiple groups, the service computes her `MaxResults` value to be the maximum of the `MaxResults` for all the groups of which the user is a member (removing the limit if it encounters a setting of `unlimited`, but ignoring any settings still at the default value of `unset`). If a particular user is not in any groups, her `MaxResults` value is `unset`. (The user's `MaxScanRows`, `MaxLockTime`, and `MaxOpenFiles` limits are computed in the same way.)

  The speed of most hardware should make it unnecessary to ever set a `MaxResults` value below 10,000, a `MaxScanRows` value below 50,000, or a `MaxLockTime` value below 1,000.

  A user can also set these limits by specifying them on a per-command basis for some commands. Values set for individual commands, override values set using **p4 group**. To disable overriding **p4 group** settings, set `server.commandlimits=2`.

- To unload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough that users will not need to ask for assistance with **`p4 unload`** or **`p4 reload`** operations.

- To display a group's `maxresults`, `maxscanrows`, `maxlocktime`, `MaxOpenFiles` and `timeout` limits, use **`p4 groups -v`** ***`groupname`***.

- Use **`p4 help maxresults`** to obtain the list of commands that are affected by any of the four limiting values.

Related Commands

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

To synchronize LDAP and Perforce groups	**p4 ldapsync**

p4 groups

Synopsis

List groups of users.

Syntax

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

Description

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

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

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

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

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

Options

`-g` *name*	List groups with the specified name.
`-i`	Display groups to which the *user* or *group* is an indirect member (that is, by means of inclusion in a subgroup).
`-m` *max*	List only the first *max* groups.
`-o` *name*	List groups owned by the named user.
`-u` *name*	List groups for whom the specified user is a member.
`-v`	Display verbose output: include `MaxResults`, `MaxScanRows`, `MaxLockTime`, and `Timeout` values.
g-opts	See "Global Options" on page 571.

Usage Notes

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

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

Examples

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

Related Commands

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

p4 have

Synopsis

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

Syntax

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

Description

List those files and revisions that have been copied to the client workspace with **p4 sync**. If file patterns are provided, the list is limited to those files that match one of the patterns, and to those files that are mapped to the client view.

p4 have lists the files, one per line, in the format:

depot-file#revision-number - local-path

- *depot-file* is the path to the file in *depot syntax*.
- *revision-number* is the *have revision*; the revision presently in the current client workspace
- *local-path* is the path as represented in terms of the local filesystem (that is, in *local syntax*).

Options

g-opts	See "Global Options" on page 571.

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.

  The have list does *not* include files that exist in your client workspace but not in the depot (nor does it include files at deleted revisions.)

  For instance, if you use **p4 add** to open a newly created file in your client workspace for add, or if you use **p4 integrate** to create a group of files in your client workspace, but haven't submitted them, the new files do not appear in the output of **p4 have**.

The set of all files in your client workspace is the union of the set of files listed by **p4 have** with the set of files listed by **p4 opened**.

- For files containing the special characters `@`, `#`, `*`, and `%`, the *depot-file* field shows the ASCII expression of the character's hexadecimal value, and the *local-path* shows the special character. For example:

```
//depot/status/100%25.txt#1 - /staff/status/100%.txt
```

Examples

```
p4 sync //depot/name...
p4 have //depot/name

p4 sync //depot/name/...#4
p4 have //depot/name
```

In each of these two pairs of commands:

The first **p4 have** shows that the highest revision of the file has been copied to the client workspace.

The second **p4 have** shows that the fourth revision is the revision currently in the client workspace.

Related Commands

To copy file revisions from the depot to the client workspace	**p4 sync**

p4 help

Synopsis

Provide on-line help for Perforce.

Syntax

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

Description

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

p4 help with no arguments lists all the available **p4 help** options. **p4 help *command*** provides help on the named *command*. **p4 help *keyword*** takes the following keywords as arguments:

Command and Keyword	Meaning	Equivalent Chapter in this Manual
p4 help administration	Help on specialized administration topics.	**p4 admin**
p4 help charset	Describes how to control Unicode translation.	P4CHARSET description.
p4 help commands	Lists all the Perforce commands.	*Table of Contents*
p4 help configurables	Describes all of the server configuration variables.	"Configurables" on page 595.
p4 help dvcs	Describes decentralized version control with Perforce.	(none)
p4 help environment	Lists the Perforce environment variables and their meanings.	"Environment and Registry Variables" on page 485
p4 help filetypes	Lists the Perforce filetypes and their meanings.	"File Types" on page 585
p4 help jobview	Describes Perforce jobviews.	**p4 jobs** description
p4 help legal	Legal and license information.	(none)
p4 help networkaddress	Help on network address syntax.	(none)

Command and Keyword	Meaning	Equivalent Chapter in this Manual
p4 help replication	Describes specialized replication topics.	(none)
p4 help revisions	Describes Perforce revision specifiers.	"File Specifications" on page 575
p4 help simple	Provides short descriptions of the eight most basic Perforce commands.	(none)
p4 help usage	Lists the six options available with all Perforce commands.	"Global Options" on page 571
p4 help views	Describes the meaning of Perforce views.	"Views" on page 581

Usage Notes

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

Related Commands

To view information about the current Perforce configuration	**p4 info**

p4 ignores

Synopsis

Displays the ignore mappings computed from the rules in `P4IGNORE` files.

Syntax

```
p4 [g-opts] ignores [-v] [path ...]
p4 [g-opts] ignores -i [-v] path ...
```

Description

It's not always easy to tell why a file is being ignored during add and reconcile operations. To help with that, **`p4 ignores`** lets you see the mappings in Perforce syntax that are applied for any specified path.

If you do not specify a path, the mappings for the current working directory are displayed.

Using the `-v` option outputs additional information about the source of each set of mappings: the rule that generated them and the line number in the file where the rule is defined.

The `-i` option checks whether a filepath would be ignored; if it would, the path is returned. With the `-v` option, a more verbose message is returned explaining whether the filepath would be ignored and if it matched any ignore rules.

Options

`-i`	Test to see if the specified filepath would be ignored.
`-v`	Report additional information about the `P4IGNORE` rules in effect. When listing mappings, include the rules from which they were generated. When testing a filepath, include the rule that affected the output.
`path`	The path to be tested. Specify the path using local syntax.
`g-opts`	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	none

Examples

Note that in the following table, the commands in the lefthand column were run in the `c:\workspace` directory.

`p4 ignores`	Display a list of mappings generated from `P4IGNORE` rules.
	
	`.../.p4root/...`
	`.../.p4root`
	`.../.p4config/...`
	`.../.p4config`
	`c:/workspace/.../builds/...`
	`c:/workspace/builds/...`
	`c:/workspace/.../builds`
	`c:/workspace/builds`

`p4 ignores -v`	Display a list of mappings generated from `P4IGNORE` rules, including the rules that they were generated from.
	
	`#FILE - defaults`
	`#LINE 2:**/.p4root`
	`.../.p4root/...`
	`.../.p4root`
	`#LINE 1:**/.p4config`
	`.../.p4config/...`
	`.../.p4config`
	`#FILE c:\workspace\.p4ignore`
	`#LINE 1:builds`
	`c:/workspace/.../builds/...`
	`c:/workspace/builds/...`
	`c:/workspace/.../builds`
	`c:/workspace/builds`

`p4 ignores -i file1 builds/file2`	Test to see whether `file1` and `file2` will be ignored. If a file is ignored, it will be returned. Otherwise, there will be no output. `c:\workspace\builds\file2 ignored`
`p4 ignores -i -v file1 builds/file2`	Test to see whether `file1` and `file2` will be ignored. If a file is ignored then it will be returned with information about the `P4IGNORE` rule that it matched. If it is not, it will be returned with a message saying that it is not ignored; if it matched a `P4IGNORE` exclusionary rule, information about that rule will also be returned.

```
c:\workspace\file1 not ignored
c:\workspace\builds\file2 ignored by
  c:\workspace\.p4ignore:1:builds
```

p4 info

Synopsis

Display information about the current Perforce application and the shared versioning service.

Syntax

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

Description

The **p4 info** command displays information about the Perforce application and the shared versioning service.

Here's an example of the output from **p4 info**. If the server were a replica of another server, that information would be supplied.

```
Client name: myserver-24-n102
Client host: myserver-24-n102.dhcp.perforce.com
Client unknown.
Current directory: /Users/jbujes
Peer address: 10.0.102.24:52492
Client address: 10.0.102.24
Server address: someaddress.perforce.com:1999
Server root: /depots/p4-1999
Server date: 2015/07/13 14:52:59 -0700 PDT
Server uptime: 147:34:34
Server version: P4D/LINUX26X86_64/2015.2.MAIN-TEST_ONLY/1199094 (2015/07/07)
ServerID: Master1999
Server services: standard
Server license: 500 users (expires 2017/01/31)
Server license-ip: qaplay.perforce.com
Case Handling: sensitive
```

To obtain the version of the Perforce application (**p4**), use **p4 -V**.

Options

-s	Shortened output: exclude information (for example, the workspace root) that requires a database lookup.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To read Perforce's help files	**p4 help**
To display Perforce Proxy connection information	**p4 proxy**
To view version information for your Perforce application	**p4 -V**

p4 init

Synopsis

Initializes a new Perforce server.

Syntax

p4 [-u *user*] [-d *dir*] [-c *client*] init [-h -q] [-c *stream*] [-C*x*] [-xi -n] [-p]

Description

Initialize a new personal (local) Perforce server.

In order to run **p4 init**, you must have up-to-date and matching versions of the **p4** and **p4d** executables in your operating system path. You can download these executables from www.perforce.com.

Perforce stores its database files in the directory named `.p4root`. Perforce stores configuration settings in the `P4CONFIG` and `P4IGNORE` files at the top level of your directory. It is not necessary to view or update these files, but you should be aware that they exist.

After initializing your new server, run **p4 reconcile** to mark all of your source files to be added to Perforce, then **p4 submit** to submit them.

Options

`-c` *stream*	Specifies the stream to use as the mainline stream instead of the default `//stream/main`.
`-C`*x*	Sets the case sensitivity of the installation. If *x* is set to `0`, your installation is case-sensitive, if set to `1` your installation is case-insensitive. Your client must match the case sensitivity of the server you're fetching from or pushing to.
`-d` *directory*	Specifies the directory in which Perforce initializes the server. Without this option, Perforce initializes the server in the current directory.
`-h`	Display help for this command, as it operates on the client.
`-n`	Configures the installation without unicode support. This is useful because the unicode capability of the local server must match that of the server you fetch from and push to.
`-q`	Suppresses informational messages.
`-p`	Specifies the address of a remote server whose case sensitivity and unicode settings you want to discover. Specifying this information makes your local server compatible with the remote server.

`-u username`	Specifies your Helix user name.
`-xi`	Configures the installation with unicode support.
g-opts	See "Global Options" on page 571.

Without `-xi` or `-n`, unicode support is detected by finding a `P4CHARSET` setting.

Usage Notes

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

Examples

`p4 init`	Initializes a new Perforce personal server.

Related Commands

Clone a new server	**`p4 clone`**

p4 integrate

Synopsis

Open files for branching or merging.

p4 integrate can be abbreviated as **p4 integ**.

Syntax

```
p4 [g-opts] integrate [options] fromFile[revRange] toFile
p4 [g-opts] integrate [options] -b branch [-r] [toFile[RevRange] ...]
p4 [g-opts] integrate [options] -b branch -s fromFile[revRange] [toFile ...]
p4 [g-opts] integrate [options] -S stream [-r] [-P parent] [file[revRange] ...]

options: -c changelist -Di -f -h -O options -n -m max -Roptions -q
```

Description

When you've made changes to a file that need to be propagated to another file, start the process with **p4 integrate**. The command includes four syntax variants, depending on whether the source and target files are specified using files, branches, or streams.

The simplest syntax variant is **p4 integrate** *fromFile toFile*; this lets the versioning service know that changes in *fromFile* need to be propagated to *toFile*, and has the following effects:

- If *toFile* does not yet exist, *fromFile* is copied to *toFile*, then *toFile* is opened for **branch** in the client workspace.
- If *toFile* exists, and shares a common ancestor with *fromfile* as above, then *toFile* is opened for **integrate**. You can then use **p4 resolve** to propagate all of, portions of, or none of the changes in *fromFile* to *toFile*.

  The **p4 resolve** command uses *fromFile* as *theirs*, *toFile* as *yours*, and the file with the most edits in common as the base.
- If *fromFile* was deleted at its last revision (and all previous changes have already been integrated between *fromFile* and *toFile*), *toFile* is opened for **delete** in the client workspace.
- Whether you move files using **p4 move**, or whether you use native OS commands to rename files within your workspace (using **p4 reconcile** or **p4 status** to update your changelist to reflect the moves you made), **p4 integrate** automatically detects these actions, adjusts the source-to-target mappings appropriately, and schedules a filename resolve for each remapped file pair.

(Some of the available options modify this behavior. See "Options" on page 184 for details.)

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

Note	If you integrate from a classic branch or other stream depot to a task stream, the files are not copied up to the parent unless they are edited and submitted first.

To specify multiple files, use wildcards in ***fromFile*** and ***toFile***. Any wildcards used in ***fromFile*** must match identical wildcards in ***toFile***. 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***.

By default, files that have been opened for **branch** or **integrate** with **p4 integrate** are read-only in the client workspace. You can edit these files before submitting them using **p4 edit** to reopen the file for **edit**.

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

In most cases, **p4 integrate** performs a lazy copy; the contents of the file are not duplicated on the server, because the integration record contains sufficient information to reproduce the file. Integrations performed on temporary object files (**+S** and **+S*n***) do not produce a lazy copy; the integrated **tempobj** file consumes additional diskspace on the server.

Alternatives to **p4 integrate** include the following:

- **p4 populate** to quickly branch files without opening them in a workspace.
- **p4 copy** to open files to be copied without scheduling any resolves.
- **p4 merge** to open files to be merged and schedule resolves for all changes.

Options

Because some of the integration options add complexity to the integration process, we've divided the options into "Basic Integration Options" on page 184 and "Advanced Integration Options" on page 185.

Basic Integration Options

-b ***branchname*** **[***toFiles*** ...]**	Integrate the files using the ***sourceFile***/***targetFile*** mappings included in the branch view of ***branchname***. If the ***toFiles*** argument is included, include only those target files in the branch view that match the pattern specified by ***toFiles***. If a revision range is supplied with ***toFiles***, the range refers to source revisions, not target revisions.
fromFiles toFiles	***fromFiles*** are called the *source files*; ***toFiles*** are called the *target files*.

	Any *toFiles* that **p4 integrate** needs to operate on must be included in the **p4 client** view.
`-n`	Display the integrations this command would perform without actually performing them.
`-c` *changelist*	Open the *toFiles* for `branch`, `integrate`, or `delete` in the specified pending changelist. If this option is not provided, the files are opened in the default changelist.
`-q`	Quiet mode; suppresses normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.
g-opts	See "Global Options" on page 571.

Advanced Integration Options

`-b` *branchname* `-s` *fromFile*[*revRange*] [*toFiles* `...`]	In its simplest form, **p4 integrate -b** ***branchname*** **-s** ***fromFile*** allows you to integrate files using the source/target mappings included in the branch view of ***branchname***, but include only those source files that match the patterns specified by ***fromFile***. In its more complicated form, when both ***fromFile*** and ***toFile*** are specified, integration is performed bidirectionally: first, integration is performed from ***fromFile*** to ***toFile***; then integration is performed from ***toFile*** to ***fromFile***. This variation of **p4 integrate** was written to provide some needed functionality to graphical Perforce applications; it is unlikely that you'll need to use this more complex form.
`-b` *branchname* `-r` [*toFiles* `...`]	Reverse the mappings in the branch view, integrating from the target files to the source files.
`-Di`	The `-Di` option modifies the way deleted revisions are treated. If the source file has been deleted and re-added, revisions that precede the deletion will be considered to be part of the same source file. By default, re-added files are considered

	to be unrelated to the files of the same name that preceded them. This flag is intended to be used for the documented purposes only, otherwise it might produce unintended integration results. For example, using the `-Di` option when a source file has been moved or renamed will not propagate the moved file correctly. The move/add and move/delete revisions are propagated as `branch` and `delete` revisions instead.
`-f`	Force the integration on all revisions of *`fromFile`* and *`toFile`*, even if some revisions have been integrated in the past. Best used with a revision range.
`-h`	Don't automatically sync target files to the head revision before integrating. Use the have revision instead.
`-m` *`max`*	Limit the command to integrating only the first *`max`* files.
`-Ob`	The `-Ob` option outputs the base revision for the merge (if any).
`-Or`	The `-Or` option outputs the resolves that are being scheduled.
`-Rb`	The `-Rb` option schedules a branch resolve instead of branching the target files automatically.
`-Rd`	The `-Rd` option schedules a delete resolve instead of deleting the target files automatically.
`-Rs`	The `-Rs` option skips cherry-picked revisions that have already been integrated. Using this option can improve merge results, but can also cause multiple resolves per file to be scheduled.
`-S` *`stream`* [`-P` *`parent`*]	Integrates a stream to its parent. To reverse the direction of the mapping, use the `-r` option.

	To override the configured parent and integrate to a different target stream, specify `-P`. To submit integrated stream files, the current client must be switched to the target stream or to a virtual child stream of the target stream.

Usage Notes

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

Examples

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

Related Commands

To create or edit a branch mapping	**`p4 branch`**
To view a list of existing branch mappings	**`p4 branches`**
To view a list of integrations that have already been performed and submitted	**`p4 integrated`**

To propagate changes from one file to another after opening files with **p4 integrate**	**p4 resolve**
To view a history of all integrations performed on a particular file	**p4 filelog**

p4 integrated

Synopsis

Show integrations that have been submitted.

Syntax

p4 [*g-opts*] integrated [-b *branchname* [-r]] *file* ...

Description

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

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

where:

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

Integrate Action	What the User Did During the **p4 resolve** Process
branch from	*file* did not previously exist; it was created as a copy of *partner-file*.
branch into	*partner-file* did not previously exist; it was created as a copy of *file*.
merge from	*file* was integrated from *partner-file*, accepting *merge*.
merge into	*file* was integrated into *partner-file*, accepting *merge*.
moved from	*file* was integrated from *partner-file*, accepting *theirs* and deleting the original.
moved into	*file* was integrated into *partner-file*, accepting *theirs* and creating *partner-file* if it did not previously exist.
copy from	*file* was integrated from *partner-file*, accepting *theirs*.
copy into	*file* was integrated into *partner-file*, accepting *theirs*.
ignored	*file* was integrated from *partner-file*, accepting *yours*.
ignored by	*file* was integrated into *partner-file*, accepting *yours*.
delete from	*file* was integrated from *partner-file*, and *partner-file* had been previously deleted.

Integrate Action	What the User Did During the p4 resolve Process
`delete into`	*file* was integrated into *partner-file*, and *file* had been previously deleted.
`edit from`	*file* was integrated from *partner-file*, and *file* was edited within the **p4 resolve** process. This allows you to determine whether the change should ever be integrated back; automated changes (`merge from`) needn't be, but original user edits (`edit from`) performed during the resolve should be.
`edit into`	*file* was integrated into *partner-file*, and *partner-file* was reopened for `edit` before submission.
`add from`	*file* was integrated from a deleted *partner-file*, and *partner-file* was reopened for `add` (that is, someone restored a deleted file by syncing back to a pre-deleted revision and adding the file).
`add into`	*file* was integrated into previously nonexistent *partner-file*, and *partner-file* was reopened for `add` before submission.

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

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

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

Options

`-b` *branchname*	Only files integrated from the source to target files in the branch view are listed. Qualified files are listed, even if they were integrated without using the branch view.
`-r`	Reverses the mappings in the branch view, swapping the target files and source files. The `-b` branch flag is required.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- When using tagged output with **`p4 integrated`** be warned that the `toFile` and `fromFile` values will be the opposite of how they are presented in the non-tagged output. The following example shows how the output varies:

```
$ p4 integrated //depot/main/revisions.h
//depot/main/revisions.h#1 - add into //depot/dev/revisions.h#1

$ p4 -ztag integrated //depot/main/revisions.h
... toFile //depot/main/revisions.h
... fromFile //depot/dev/revisions.h
... startToRev #none
... endToRev #1
... startFromRev #none
... endFromRev #1
... how add into
... change 12345
```

See "Global Options" on page 571 for information on how to enable tagged output.

Related Commands

To see a list of integrations that have not yet been resolved	**`p4 resolve -n`**
To view a list of integrations that have been resolved but not yet submitted	**`p4 resolved`**
To perform an integration	**`p4 integrate`**
To view the actions taken for all revisions of a particular file (including all the files from which that particular file was integrated)	**`p4 filelog [-i]`** ***`file`***

p4 interchanges

Synopsis

Report changes not yet integrated.

Syntax

```
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] fromFile[revRange] toFile
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] -b branchname [toFile[revRange] ...]
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] -b branchname -s fromFile[revRange]
                         [toFile ...]
p4 [g-opts] interchanges [-f -l -r -t -F] [-u user] -S stream [-P parent] [file[revRange]]
                         [toFile ...]
```

Description

The **p4 interchanges** command lists changes that have not been integrated from a set of source files to a set of target files. The command also reports changes that consist solely of ignored integrations if those changes have not yet been integrated into the target.

Options

`-b branchname`	Use the source and target as defined by the specified branch specification.
`-b branchname -s fromFile[RevRange] [toFiles ...]`	Preview bidirectional integrations (used by Perforce applications; see **p4 integrate** for details.)
`-f`	List files that require integration. For partially integrated changelists, files might be listed even if they were integrated individually.
`-F`	Used with `-S`, ignores a stream's expected flow. It can also force it to generate a branch view based on a virtual stream; the mapping itself refers to the underlying real stream.
`-l`	Long form: include full text of the changelist description.
`-r`	Reverse source and target (that is, reverse the direction of the integration).
`-S stream [-P parent]`	Display integrations pending between the stream and its parent. To treat another stream as the parent, specify `-P`.

`-t`	Display full date and time that changelist was submitted. By default, only the date is displayed.
`-u` *`user`*	Limit results to those submitted by the specified user.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

p4 istat

Synopsis

Check for integrations needed for a stream.

Syntax

p4 [*g-opts*] istat [-a -c -r -s] *stream*

Description

Check for integrations that are needed with respect to the parent stream. (Primarily for Perforce applications that checking this status in order to render it in human-readable format.)

In a distributed environment, this command is run directly against an edge server; it is not forwarded to the commit server.

Options

`-a`	Check for all integrations, to and from the parent stream
`-c`	Clear cached information before checking integration history. Intended for diagnostic use.
`-r`	Check for integrations required from the parent stream.
`-s`	Display the status of a stream and generate cache data without executing database queries.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- The `-c` option is intended for diagnostic and cache consistency checks associated with P4V, the Perforce Visual Client.

Related Commands

To display changes/sync status for the current workspace.	**p4 cstat**

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 is not working correctly") or an enhancement request (for instance, "please add a flag that forces a certain operation to occur") or anything else requiring a change to some files under 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. Because 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 installation to another.

When **p4 job** is called with no arguments, a new job named **job***NNNNNN* is created, where *NNNNNN* is a sequential six-digit number. You can change the job's name within the form before quitting the editor. If **p4 job** is called with a *jobname* argument, a job of that name is created; if that job already exists, it is edited.

Once a job has been created, you can link the job to the changelist(s) that fix the job with **p4 fix**, **p4 change**, or **p4 submit**. When a job is linked to a changelist, under most circumstances the job's status is set to **closed**. (See "Usage Notes" on page 198 for more information).

Form Fields

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

Field Name	Type	Description
Job:	Writable	The job's name. For a new job, this is **new**. When the form is closed, this is replaced with the name **job***NNNNNN*, where *NNNNNN* is the next six-digit number in the job numbering sequence.

Field Name	Type	Description
		Alternately, you can name the job anything at all by replacing the text in this field.
`Status:`	Writable Value	The value of this field must be `open`, `closed`, or `suspended`. When the job is linked to a changelist, the value of this field is set to `closed` when the changelist is submitted.
`User:`	Writable	The name of the user who created the job.
`Date:`	Writable	The date the job was modified.
`Description:`	Writable	An arbitrary text description of the job.

Options

`-d jobname`	Delete job *`jobname`*, but only if it has no associated pending or submitted fixes.
`-f`	Force option. Allows Perforce administrators to edit read-only fields.
`-i`	Read the job form from standard input without invoking an editor.
`-o`	Write the job form to standard output without invoking an editor.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- If the Perforce superuser has eliminated field ID# `102` (the `Status:` field) with **`p4 jobspec`**, Perforce is unable to close jobs when the changelists to which they are linked are submitted. See the **`p4 jobspec`** command and the *Helix Versioning Engine Administrator Guide: Fundamentals* for more information.
- After a job has been created or changed, Perforce indexes the job so that **`p4 jobs -e`** can locate the job quickly. The index keys are ***`word`***, ***`fieldname`*** where ***`word`*** is a case-insensitive alphanumeric word. Values in date fields are stored as the number of seconds since January 1, 1970, 00:00:00.

Examples

`p4 job`	Create a new job; by default, its name is of the form `job`*`NNNNNN`*.
`p4 job job000135`	Edit job `job000135`.

Related Commands

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

p4 jobs

Synopsis

List jobs known to the Perforce versioning service.

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 in Perforce. You can limit the output of the command by specifying various criteria with options and arguments. If you specify a file pattern, the jobs listed will be limited to those linked to changelists affecting particular files. The **-e** option can be used to further limit the listed jobs to jobs containing certain words.

Jobs are listed in alphanumeric order (or, if you use the **-r** option, in reverse alphanumeric order) by name, one job per line. The format of each line is:

jobname on *date* by *user* **status* *description*

The *description* is limited to the first 31 characters, unless the **-l** (long) option is used.

If any of the **date**, **user**, **status**, or **description** fields have been removed by the Perforce superuser with **p4 jobspec**, the corresponding value will be missing from each job's output.

To limit the list of jobs to those that have been fixed by changelists that affected particular files, use **p4 jobs *filespec***. The files or file patterns provided can contain revision specifiers or a revision range.

Options

-e *jobview*	List only those jobs that match the criteria specified by *jobview*. See "Usage Notes" on page 202 for a discussion of job views.
-i *files ...*	Include jobs fixed by changelists that affect files integrated into the named files.
-l	Output the full description of each job.
-m *max*	Include only the first ***max*** jobs, sorted alphanumerically. If used with the **-r** option, the last ***max*** jobs are included.
-r	Display jobs in reverse alphabetical order by job name.
-R	Rebuild the job table and re-index each job. Re-indexing the table is necessary either when upgrading from version 98.2 or earlier, or when upgrading from 99.1 to 2001.1 or higher and you wish to search your body of existing jobs for strings containing punctuation.

g-opts See "Global Options" on page 571.

Usage Notes

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

Job Views

Use **`p4 jobs -e`** ***`jobview`*** to limit the list of jobs to those that contain particular words. You can specify that the search terms be matched only in particular fields, or anywhere in the text of the job. You can use job views to match jobs by values in date fields, though there are fewer options for dates than there are for text. Job fields of type **`bulk`** are not indexed for searching.

Text matching is case-insensitive. All alphanumeric strings (including words including embedded punctuation) separated by whitespace are indexed as words.

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

Spaces between search terms in job views act as boolean AND operations. To find jobs that contain any of the terms (boolean OR), separate the terms with the "`|`" character.

Ampersands (**`&`**) can be used as boolean ANDs as well; the boolean operators bind in the order **`&`**, **`|`**, space (highest precedence to lowest precedence). Use parentheses to change the grouping order.

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

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

Date fields can be matched by expressing the job view date as ***`yyyy/mm/dd`*** or ***`yyyy/mm/dd:hh:mm:ss`***. If a specific time is not provided, the equality operator (`=`) matches the entire day.

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

Additionally, you can use the NOT operator (`^`) to negate the sense of some comparisons. (See Limitations below for details).

Regular expression matching is supported by the regular expression matching operator (`~=`).

To search for words containing characters that are job search expression operators, escape the characters with a backslash (`\`) character. To match the backslash character, escape it with an additional backlash (`\\`).

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

Field Type	Use of Comparison Operators in Job Views
`word`	The equality operator (`=`) must match the value in the word field exactly. The relational operators perform comparisons in ASCII order.
`text`	The equality operator (`=`) matches the job if the word given as the value is found anywhere in the specified field. The relational operators are of limited use here, because they match the job if *any* word in the specified field matches the provided value. For example, if a job has a `text` field `ShortDescription` that contains only the phrase `gui bug`, and the job view is "`ShortDesc<filter`", the job matches the job view, because `bug<filter`.
`line`	As for field type `text`, above.
`select`	The equality operator (`=`) matches a job if the value of the named field is the specified word. The relational operators perform comparisons in ASCII order.
`date`	Dates are matched chronologically. If a specific time is not provided, the operators `=`, `<=`, and `>=` match the entire day.

If you're not sure of a field's type, run **p4 jobspec -o**, which outputs the job specification used at your site. The **p4 jobspec** field called `Fields:` contains the job fields' names and datatypes. See **p4 jobspec** for a discussion of the different field types.

Other Usage Notes

- The **p4 user** form has a `JobView:` field that allows a job view to be linked to a particular user. After a user enters a job view into this field, any changelists he creates automatically list jobs that match the job view in this field. The jobs that are fixed by the changelist can be left in the form, and the jobs that are not should be deleted.
- **p4 jobs** sorts its output alphanumerically by job name, which also happens to be the chronological order in which the jobs were entered. If you use job names other than the standard 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 job view "*field*=*"; any non-null value for *field* matches.
- When querying for jobs using the `-e` *jobview* option, be aware of your operating system and command shell's behavior for parsing, quoting, and escaping special characters, particularly when using wildcards, logical operators, and parentheses.

Limitations

- Job views cannot be used to search for jobs containing null-valued fields. In other words, if a field has been deleted from an existing job, then the field is not indexed, and there is no job view that matches this "deleted field" value.
- The job view NOT operator (^) can be used only after an AND within the job view. Thus, the job views "`gui ^name=joe`" and "`gui&^name=joe`" are valid, while the job views "`gui|^name=joe`" and "`^name=joe`" are not.
- The * wildcard is a useful way of getting around both of these limitations.

  For instance, to obtain all jobs without the string "`unwanted`", query for '`job=* ^unwanted`". All jobs will be selected by the first portion of the job view and logically ANDed with all jobs NOT containing the string "unwanted".

  Likewise, because the job view "*`field`*=*" matches any *non*-null value for *`field`*, (and the `job` field can be assumed not to be null), you can search for jobs with null-valued fields with "`job=* ^`*`field`*=*"
- You cannot currently search on space-delimited fields with conditionals. For example, instead of using **`p4 jobs -e "field=word1 word2"`**, you must use **`p4 jobs -e "field=word1 field=word2"`**.

Examples

`p4 jobs //depot/proj/file#1`	List all jobs attached to changelists that include revisions of `//depot/proj/file`.
`p4 jobs -i //depot/proj/file`	List all jobs attached to changelists that include revisions of `//depot/proj/file` or revisions of files that were integrated into `//depot/proj/file`.
`p4 jobs -e gui`	List all jobs that contain the word `gui` in any field.
`p4 jobs -e "gui Submitted-By=joe"`	List all jobs that contain the word `gui` in any field and the word `joe` in the `Submitted-By:` field.
`p4 jobs -e "gui ^Submitted-By=joe"`	List all jobs that contain the word `gui` in any field and any value *other than* `joe` in the `Submitted-By:` field.
`p4 jobs -e "window*"`	List all jobs containing the word "`window`", "`window.c`", "`Windows`", in any field. The quotation marks are used to prevent the local shell from expanding the "*" on the command line.
`p4 jobs -e window.c`	List all jobs referring to `window.c` in any field.

p4 jobs -e "job=* ^unwanted"	List all jobs not containing the word **unwanted** in any field.
p4 jobs -e "(fast|quick)&date>1998/03/14"	List all jobs that contain the word **fast** or **quick** in any field, and have a **date:** field pointing to a date on or after **3/14/98**.
p4 jobs -e "fast|quick" //depot/proj/...	List all jobs that have the word **fast** or **quick** in any field, and that are linked to changelists that affected files under **//depot/proj**.

Related Commands

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

p4 jobspec

Synopsis

Edit the jobs template.

Syntax

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

Description

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

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

Form Fields

Field Name	Description
`Fields:`	A list of field definitions for your site's jobs, one field per line. Each line is of the form *code name datatype length fieldtype*. • *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 "Usage Notes" on page 209 for more details. • *name*: the name of the field. This can be changed at any time, while the code should not change once jobs have been created. Field names must not contain spaces. • *datatype*: the datatype of the field. Possible values are: • **word**: a single arbitrary word (a string with no spaces) • **date**: a date/time field • **select**: one of a fixed set of words • **line**: one line of text • **text**: a block of text, starting on the line underneath the fieldname. • **bulk**: like **text**, but not indexed for searching with **p4 jobs -e**.

Field Name	Description
	• *length*: recommended length for display boxes in GUI clients accessing this field. Use a value of `0` to let a Perforce application choose its own value. • *fieldtype*: does the field have a default value? Is it required? Is it read-only? Possible values are: • `optional`: field can take any value or be erased. • `default`: a default value is provided; it can be changed or erased. • `required`: a default value is provided; it can be changed but the user must enter a value. • `once`: read-only; the field value is set once to a default value and is never changed. • `always`: read-only; the field's value is set to a new default when the job is edited. This is useful only with the `$now` and `$user` variables; it allows you to change the date a job was modified and the name of the modifying user.
`Values:`	Contains a lists of fields and valid values for `select` fields. Enter one line for each field of datatype `select`. Each line must contain the fieldname, a space, and the list of acceptable values separated by slashes. For example: `JobType bug/request/problem`.
`Presets:`	Contains a list of fields and their default values for each field that has a fieldtype of `default`, `required`, `once`, or `always`. Each line must contain the field name and the default value, separated by a space. For example: `JobType bug` Any one-line string can be used, or one of three built-in variables: • `$user`: the user who created the job • `$now`: the current date • `$blank`: the phrase `<enter description here>` When users enter jobs, any fields in your jobspec with a preset of `$blank` must be filled in by the user before the job is added to the system. See "Usage Notes" on page 209 for special considerations for field 102.
`Comments:`	Textual comments that appear at the top of each **`p4 job`** form. Each line must begin with the comment character `#`.

Field Name	Description
	See "Usage Notes" on page 209 for special considerations for these comments if your users need to enter jobs through P4V, the Perforce Visual Client.

Options

`-i`	Read the jobspec form from standard input.
`-o`	Write the jobspec form to standard output.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- Do not attempt to change, rename, or redefine fields 101 through 105. These fields are used by 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 applications. Although it is possible to rename or delete these fields, it is highly undesirable to do so. Perforce applications may continue to set the value of field 102 (the `Status:` field) to `closed` (or some other value defined in the `Presets:` for field 102) upon changelist submission, even if the administrator has redefined field 102 to for use as a field that does not contain `closed` as a permissible value, leading to unpredictable and confusing results.

- The information in the `Comments:` fields is the only information available to your users to tell them how to fill in the job form, and is also used by P4V, the Perforce Visual Client, to display tooltips. Please make your comments complete and understandable.

- The `Presets:` entry for the job status field (field 102) has a special syntax for providing a default fix status for **`p4 fix`**, **`p4 change`** **`-s`**, and **`p4 submit`** **`-s`**.

  By default, a job's status is set to closed after you use **`p4 fix`**, **`p4 change`**, or **`p4 submit`**. To change the default fix status from `closed` to some other ***`fixStatus`*** (assuming that you have defined the ***`fixStatus`*** as a valid `select` setting in the `Values:` field), use the special syntax of ***`jobStatus`*****`,fix/`*****`fixStatus`** in the `Presets:` field for field 102 (job status). To change the behavior of **`p4 fix`**, **`p4 change`**, and **`p4 submit`** to leave job status unchanged, use the special ***`fixStatus`*** of `same`.

- See the "Customizing Perforce: Job Specifications" chapter of the *Helix Versioning Engine Administrator Guide: Fundamentals* 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 journalcopy

Synopsis

Copies journal data from a master server to the local file system of a standby replica. The copy is identical, byte-for-byte, to the original.

Displays information about the current copy position.

Syntax

```
p4 [g-opts] journalcopy -l
p4 [g-opts] journalcopy [-i n] [-b wait]
```

Description

The **p4 journalcopy** provides two syntax variants:

- The first variant displays information about the current copy position from the master's journal to the replica's journal.
- The second variant copies journal data to the local file system of a standby replica.

Obtaining the current copy position

An operator may run the **p4 journalcopy -l**, **p4 pull -l -j**, and **p4 pull -l -s** commands. This makes it possible for an operator to confirm the state of a replica.

The output of the **p4 journalcopy -l** command is shown below. The sequence number indicates the offset position in the journal that the copy has reached.

```
Current replica persisted journal state is: Journal 2, Sequence 6510347
```

You can compare this offset with the output for the **p4 pull -l** command for the replica. The latter indicates the offset position in the journal that has been written to the database.

Copying journal records

In order for a standby replica to take over for the master server, it must have a copy of the master server's metadata and access to versioned files. To do this, you must run the following commands:

- The **p4 journalcopy** command to copy the master's journal file to the local file system of the standby replica. This command does not apply the copied journal records to the replica's database.
- The **p4 pull -L** command to retrieve the journal records from journal files created by the **p4 journalcopy** and to apply these to the replica's database.

The combination of the **p4 journalcopy** command and the **p4 pull -L** command accomplish the same result as using the **p4 pull** command, with the following differences:

- Journal files are copied before the database is updated.
- Copied files are physically identical to the master's journal, not just logically equivalent.

  Having an identical copy of the journal files is useful for failover because other servers in your installation, which will have stored physical byte offsets into the master journal files in their state files, do not have to adjust those offset positions during failover; they just switch their target to the new master and continue replicating from it.
- Applying journal records to the database occurs in parallel with fetching new records from the master, rather than sequentially as is done in a plain **p4 pull** command.
- An additional statefile is used to coordinate the **p4 journalcopy** and the **p4 pull -L** commands.

The **p4 journalcopy** command runs very quickly, so journal records can be transferred from the master server to the standby replica with very little lag and with very little overhead on the master server.

By default when **p4 journalcopy** detects an error with its connection to the master, it sleeps for one second and then attempts to reconnect and resume transferring journal records. You can set the default value when it sets a **p4 journalcopy** startup thread like this:

```
$ p4 -p depot_master_p4port configure set standby#startup.1 = journalcopy -i 0 -b 1+
```

Options

`-b wait`	Wait the specified number of seconds to retry the **p4 journalcopy** command after a failed attempt.
`-i n`	Repeat the **p4 journalcopy** command every *n* seconds. If you do not use this option, the command runs once. If you set *n* to `0`, the command will run as frequently as possible rather than sleeping between runs.
`-l`	Report on the current position in the copied journal.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Examples

`p4 journalcopy -l`	Show information about the current copy position from the master's journal to the replica's journal.

Related Commands

To make a copy of the master's versioned files.	**`p4 pull -u`**
To get information about replication status from the point of view of the master server, use the `-J` option of the **`p4 servers`** command.	**`p4 servers`**

p4 journaldbchecksums

Synopsis

Write journal notes with table checksums.

Syntax

```
p4 [g-opts] journaldbchecksums [-t tableincludelist | -T tableexcludelist] [-l level]
p4 [g-opts] journaldbchecksums -u filename -t tablename [-v version] [-z]
p4 [g-opts] journaldbchecksums -s -t tablename [-b blocksize] [-v version]
p4 [g-opts] journaldbchecksums -c changelist
```

Description

The **p4 journaldbchecksums** command provides a set of tools for ensuring data integrity across a distributed or replicated installation.

The Perforce service automatically performs an integrity check whenever you use the **p4 admin checkpoint** or **p4 admin journal** commands, or when you use **p4 journaldbchecksums** to manually perform an integrity check.

To use this command, structured logging (see **p4 logparse**) must be enabled, and at least one structured log must be capturing events of type `integrity`.

When an integrity check is performed, the Perforce service writes records to the journal that contains the checksums of the specified tables (or, if no tables are specified, for all tables). Replica servers, upon receiving these records, compare these checksums with those computed against their own database tables, as they would with **p4 dbstat**. Results of the comparisons are written in the replica's log.

You can control which tables are checked, either by including and excluding individual tables with the `-t` and `-T` options, or by using one of three levels of verification.

Verification levels are controlled by the `rpl.checksum.auto` configurable or the `-l level` option.

- Level 1 corresponds to the most important system and revision tables.
- Level 2 includes all of level 1 as well as certain metadata that is not expected to differ between replicas.
- Level 3 includes all metadata, including metadata that is likely to differ between replicas, particularly build farms and edge servers.

When checking individual changelists and individual tables, the `rpl.checksum.change` and the `rpl.checksum.table` configurables control when events are written to the log.

For more information, including a list of database tables associated with each level of verification, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

Options

`-b` *`blocksize`*	When scanning tables, scan *blocksize* records per block. The default is 5,000. For each block, the server computes a block checksum and writes it as a journal note. Replica servers automatically verify these blocks when processing these notes. This option can be used with large tables on a production system as the table is unlocked between each block. Inspecting the results of the block verifications will reveal the location of damage that affects only part of a database table.
`-c` *`changelist`*	Compute a checksum for an individual submitted changelist. The checksum is written as a journal note, and replica servers automatically verify the checksum of the change when they process these notes.
`-l` *`level`*	Specify a level for checksumming; each level corresponds to a larger set of tables. These levels correspond to the levels used by the **`rpl.checksum.auto`** configurable.
`-s -t` *`tablename`*	Scan the specified database table.
`-t` *`tables`*	Specify the table(s) for which to compute checksums. To specify multiple tables, double-quote the list and separate the table names with spaces. The table names must start with "**`db.`**". Table names can also be separated by commas.
`-T` *`tableexcludelist`*	Compute checksums for all tables except those listed.
`-u` *`filename`* `-t` *`tablename`*	Unload the specified database table to a file. This command also writes a journal note that documents this action, and instructs replica servers to automatically unload the same table to the same file when processing these notes.
`-v` *`version`*	When unloading or scanning tables, specify the server version number to use. If no server version number is specified, the current server version is used.
`-z`	Compress the file when unloading a table.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- For more about administering Perforce in distributed or replicated environments, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

p4 journals

Synopsis

Display history of checkpoint and journal activity for the server.

Syntax

p4 [*g-opts*] journals [-F *filter*] [-T *fields*] [-m *max*]

Description

Use the **p4 journals** command to display information from the **db.ckphist** table, which holds historical information about checkpoint and journal activity. A server uses this table to record the following checkpoint and journal events:

- the server takes a checkpoint
- the server rotates a journal
- the server replays a journal
- a replica schedules a checkpoint

Each server in a multi-server installation has its own, unique **db.ckphist** table. That is, the table is not replicated to replicas. This table is not part of the main server database; it's not journaled, and it does not need to be backed up. It is not included in checkpoints. If anything goes wrong, it can be thrown away.

Here's an example of the output from **p4 journals**.

```
mbp-jbujes:~ jbujes$ p4 -p qaplay:20141 journals
... start 1381278576
... startDate 2013/10/08 17:29:36
... end 1381278576
... endDate 2013/10/08 17:29:36
... pid 19960
... type checkpoint
... flags
... jnum 19
... jfile checkpoint.19
... jdate 1381278576
... jdateDate 2013/10/08 17:29:36
... jdigest E4EB1FF5B589D05E9F5A8EE1F8183A86
... jsize 27183115
... jtype text
... failed 0
... errmsg

... start 1381278576
... startDate 2013/10/08 17:29:36
... end 1381278576
... endDate 2013/10/08 17:29:36
... pid 19960
... type checkpoint
... flags
... jnum 18
... jfile journal.18
... jdate 1381278575
... jdateDate 2013/10/08 17:29:35
... jdigest 00000000000000000000000000000000
... jsize 15737
... jtype text
... failed 1
... errmsg

... start 1374629669
... startDate 2013/07/23 18:34:29
... end 1374629669
... endDate 2013/07/23 18:34:29
... pid 14700
... type replay
... flags -r . -j r
... jnum -1
... jfile basis.ckp
... jdate 1366076427
... jdateDate 2013/04/15 18:40:27
... jdigest 00000000000000000000000000000000
... jsize 27181640
... jtype text
... failed 1
... errmsg
```

This command displays full error message text for a failed checkpoint.

Use the global `-F` option to format the output from the **`p4 journals`**; for example:

`p4 -F "%jfile% %jnum%" journals -F type=checkpoint`

The meaning of each field is described in the following table. Output entries are listed from newest event to oldest event.

`start`	Starting Unix timestamp of the command that ran. See type field to determine which command was executed.
`startDate`	Human-readable form of start value.
`end`	Ending Unix timestamp of the command that ran.
`endDate`	Human-readable form of end value.
`pid`	The process id of the command whose execution produced this record. This value can be useful in searching for related entries in other logs.
`type`	Indicates the command whose execution produced this record. Types include the following: • `journal`: refers to the **`p4 -J`** command or the **`p4 admin journal`** command. • `checkpoint`: refers to the **`p4 -jc`** command or **`p4 admin checkpoint`** command. • `replay`: refers to the **`p4d -jr`** command • `schedule`: refers to taking a checkpoint on a replicated server. This produces two records: one for the checkpoint and another for the schedule.
`flags`	Flags passed to the command implied by the type filed.
`jnum`	The checkpoint number. A value of -1 indicates that the journal number is unknown.
`jfile`	The name of a journal or checkpoint file that was input to the command implied by the type field.
`jdate`	Unix timestamp when the filed specified by `jfile` was created.
`jdateDate`	Human-readable format of `jdate` value.
`jdigest`	The digest of the checkpoint file.
`jsize`	The size of the file specified by `jfile`.
`jtype`	The type of the file specified by `jtype`.

Options

`-F` *`filter`*	List only the records that satisfy the filter expression. For instructions on constructing the filter expression, see "Job Views" on page 202.

`-m max`	Limit output to the specified number of records.
`-T fields`	Limit output to the specified fields. Separate fields using a comma or a space.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To create a checkpoint.	**`p4 admin checkpoint`**
To create a journal.	**`p4 admin journal`**

p4 key

Synopsis

Display, set, or delete a key/value pair.

Syntax

```
p4 [g-opts] key name
p4 [g-opts] key [-v]name value
p4 [g-opts] key [-d] name
p4 [g-opts] key [-i -v] name
p4 [g-opts] key [-m] [pair list]
p4 [g-opts] key --from oldvalue --to newvalue name
```

Description

Keys allow you to store name-value pairs for use in scripts. These user-managed keys are stored in a table named `db.nameval`.

The command includes the following variants:

- The variant **`p4 key name`** returns the value of key *name*.
- The variant **`p4 key name value`** sets the value of the key *name* to *value*, and if *name* does not already exist, it is created.

  Specifying **`-v`** displays the previous value of the specified key after the key has been set or incremented.
- The variant **`p4 key -d name`** deletes the specified key.
- The variant **`p4 key -i name`** increments a numeric key.

  Specifying **`-v`** displays the previous value of the specified key after the key has been set or incremented.
- The variant **`p4 key [-m] pair list`** defines multiple set and delete operations to be performed. Each operation is defined by a value pair in the pair list. To set a key, use a name and value, to delete a key, use a **`-`** (hyphen) followed by the name. See "Examples" on page 224.

  This variant is useful in distributed environments where running individual commands is likely to introduce unwanted latency.
- The final variant (**`--from`** … **`--to`**) sets the specified key to the new value only if the current value of the key is *oldvalue*. A key that has never been set or that has been deleted cannot be set using this syntax variant.

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

If the `dm.keys.hide` configurable is set to `2`, `admin` access is required.

Options

`-d` *name*	Delete key *name* from the Perforce service.
`-i` *name*	Increment key *name* by 1 and return the new value. This option can only be used with numeric keys.
`-m` *name value ...*	Perform multiple key value operations in one command. See "Examples" on page 224.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list` to display a key's value; (`admin` if `dm.keys.hide` is set to 2) `review` to set a new value

Examples

`p4 key` ***`mykey`*** **`12`**	Set the value of ***`mykey`*** to `12`. If ***`mykey`*** does not exist, it is created. Requires `review` access.
`p4 key` ***`mykey`***	Display the value of ***`mykey`***. If ***`mykey`*** does not exist, its value is displayed as `0`. Requires `list` access.
`p4 key -m` ***`mykey`*** **`5`** ***`mynewkey`*** **`4`**	Set two keys.
`p4 key -m -` ***`mykey`*** **`-`** ***`mynewkey`***	Delete two keys.
`p4 key -m` ***`mykey`*** **`6 -`** ***`mynewkey`***	Set one key; delete one key.

Related Commands

To list all keys and their values	**`p4 keys`**

p4 keys

Synopsis

Display list of known key/value pairs.

Syntax

p4 [*g-opts*] keys [-e *nameFilter*] [-m *max*]

Description

The Perforce versioning service holds a user-accessible store of key/value pairs. These user-managed keys are stored in a table named `db.nameval`.

If the `dm.keys.hide` configurable is set to 1 or 2, `admin` access is required.

`p4 keys` provides the current list of keys, along with their values.

Options

`-e` *nameFilter*	List keys with a name that matches the *nameFilter* pattern, for example: **`p4 keys -e 'mycounter-*'`**
`-m max`	List only the first *max* keys.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`list` (`admin` if `dm.keys.hide` is set to 1 or 2)

Related Commands

To view or change the value of a key — **`p4 key`**

p4 label

Synopsis

Create or edit a label specification and its view.

Syntax

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

Description

Use **p4 label** to create a new label specification or edit an existing label specification. A *labelname* is required.

Running **p4 label** allows you to configure the mapping that controls the set of files that are allowed to be included in the label. After configuring the label, use **p4 labelsync** or **p4 tag** to tag files with the label.

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

Only the `Owner:` of an `unlocked` label can use **p4 labelsync** or **p4 tag** to tag files with that label. The owner of a group may be a single user or a group.

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

Form Fields

Field Name	Type	Description
`Label:`	Read-only	The label name as provided in the invoking command.
`Owner:`	Writable, optional	The label's owner. By default, the user who created the label. Only the owner of a label can update which files are tagged with the label. The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
`Update:`	Read-only	The date the label specification was last modified.

Field Name	Type	Description
`Access:`	Read-only	The date and time the label was last accessed, either by running **`p4 labelsync`** on the label, or by otherwise referring to a file with the label revision specifier **`@label`**. (Note: Reloading a label with **`p4 reload`** does not affect the access time.)
`Description:`	Writable, optional	An optional description of the label's purpose.
`Options:`	Writable	Options to control behavior and storage location of labels • **`locked`** or **`unlocked`**. If the label is **`locked`**, the list of files tagged with the label cannot be changed with **`p4 labelsync`**. • **`autoreload`** or **`noautoreload`**. For static labels, if **`noautoreload`** is set, the label is stored in **`db.label`**, and if **`autoreload`** is set, it is stored in the unload depot. This option is ignored for automatic labels. Storing labels in the unload depot can improve performance on sites that make extremely heavy use of labels.
`Revision:`	Writable	An optional revision specification for an automatic label. If you use the **`#`** character to specify a revision number, you must use quotes around it in order to ensure that the **`#`** is parsed as a revision specifier, and not as a comment field in the form.
`View:`	Writable	A list of depot files that can be tagged with this label. No files are actually tagged until **`p4 labelsync`** is invoked. Unlike client views or branch views, which map one set of files to another, label views consist of a simple list of depot files. See "Views" on page 581 for more information.
`ServerID:`	Writable, optional	If set, restricts usage of the label to the named server. If unset, this label may be used on any server.

Options

`-d` [**`-f`**]	Delete the named label if it's **`unlocked`**. The **`-f`** option forces the deletion even if the label is **`locked`**. (Deleting a **`locked`** label requires **`admin`** or **`super`** access.)
`-f`	Allow the **`Update:`** field's date to be set. Can be used with either the **`-i`** option or the **`-t`** option for the same purpose.
`-g`	In distributed environments, use the **`-g`** option to control whether the label is local to an edge server, or globally available from the commit server.
`-i`	Read the label definition from standard input without invoking the editor.

`-o`	Write the label definition to standard output without invoking the editor.
`-t` *`template`*	Copy label *`template`*'s view and options into the `View:` and `Options:` fields of this label. You can specify a default label template using the `template.label` configure variable. If you do so, you do not have to specify this option.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- To create an automatic label, fill in the `Revision:` field of the **`p4 label`** form with a revision specifier. When you sync a workspace to an automatic label, the contents of the `Revision:` field are applied to every file in the `View:` field.
- With a distributed Perforce service, labels may be local or global. Local labels are restricted to a single edge server, and cannot be used on other servers. Global labels are created and updated on the commit server, and are visible to all servers. However, global labels may only be used with global (unbound) client workspaces.

  By default, labels are local to your edge server, and you use the `-g` option to access global labels on the commit server. If your administrator has set `rpl.labels.global` to `1`, labels are global by default, and the meaning of the `-g` option is inverted to allow updating of local labels.

Examples

**`p4 files @`*`labelname`*	List the file revisions tagged by *`labelname`*.

Related Commands

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

p4 labels

Synopsis

Display list of defined labels.

Syntax

```
p4 [g-opts] labels [-t] [-u user | --me] [[-e|-E] filter] [-m max] [file[revrange]]
p4 [g-opts] labels [-t] [-u user | --me] [[-e|-E] filter] [-m max] [-a | -s serverID]
p4 [g-opts] labels -U
```

Description

p4 labels lists all the labels known to the Perforce service in the form:

```
Label labelname date description
```

Use the `-t` option to display the time of the last update to the label.

```
Label labelname date time description
```

To see a list of loaded static labels that tag specific files, specify a file pattern, with an optional revision range. (Because automatic labels refer to all files in the label view at a specified revision range, automatic labels are not shown when you use **p4 labels** with a file pattern.)

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

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

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

Options

`-a`	List all labels, not just labels bound to this server. This option may not be used with a file specification.
`-e filter`	List only labels matching *filter* (case-sensitive).
`-E filter`	List only labels matching *filter* (case-insensitive).
`-m max`	List only the first *max* labels.
`-s serverID`	List only those labels bound to the specified *serverID*. This option may not be used with a file specification.

`-t`	Display the time as well as the date of the last update to the label.
`-u` *`user`*	List only labels owned by *`user`*.
`--me`	Equivalent to `-u $P4USER`.
`-U`	List labels in the unload depot. For details, see **`p4 unload`**.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- To see a list of files tagged by a particular label, use **`p4 files`** **`@`*`labelname`***.
- In a distributed environment, users connected to an edge server receive only those labels that are bound to their edge server, unless they explicitly request otherwise by specifying the `-a` or `-s` *`serverID`* options.

Examples

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

Related Commands

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

p4 labelsync

Synopsis

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

Syntax

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

Description

p4 labelsync causes the named label to reflect the current contents of the client workspace by tagging the last revision of each file synced into the workspace with the label name. The label name can subsequently be used in a revision specification as **@*label*** to refer to the revision of the file that was tagged with the label.

Without a file argument, **p4 labelsync** causes the label to reflect the contents of the client workspace by adding, deleting, and updating the set of files tagged with the label.

If a file is given, **p4 labelsync** updates the tag for only that named file. If the file argument includes a revision specification, the client view is ignored; the specified revision is used instead of the revision existing in the workspace. If the file argument includes a revision range, then only the highest revision in that range is used.

Only the `Owner:` of an `unlocked` label can use **p4 labelsync** to tag files with that label.

A label that has its `Options:` field set to `locked` cannot be updated with **p4 labelsync**.

Options

`-a`	Add the label to files that match the file pattern arguments; no files are deleted from the label.
`-d`	Delete the label tag from the named files.
`-g`	In distributed environments, use the `-g` option to specify whether the label being applied is local to an edge server, or is globally available from the commit server. To update a global label, the client workspace must also be an unbound (global) workspace.
`-l` *labelname*	Specify the label to be applied to file revisions.
`-n`	Display what **p4 labelsync** would do without actually performing the operation.
`-q`	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- By default, **`p4 labelsync`** operates on the revisions of files last synced to your client workspace. To tag the head revisions of files (or the highest revision in a specified range), use **`p4 tag`**.
- To see which files are tagged by the label, use **`p4 files @`*`labelname`***.
- With a distributed Perforce service, **`p4 labelsync`** works with a label local to the edge server (to which you are sending a request). The **`-g`** option can be used to apply a global label, but only with an unbound (global) client workspace.

  By default, labels are local to your edge server, and you use the **`-g`** option to access global labels on the commit server. If your administrator has set **`rpc.labels.global`** to **`1`**, labels are global by default, and the meaning of the **`-g`** option is inverted to allow updating of local labels.

Related Commands

To create or edit a label	**`p4 label`**
To list all labels known to the system	**`p4 labels`**
To create a label and tag files with the label	**`p4 tag`**

p4 ldap

Synopsis

Create, edit, or delete an LDAP configuration specification, or test an existing LDAP configuration.

This command may not be run from a read-only or build replica.

Syntax

```
p4 [gopts] ldap configname
p4 [gopts] ldap -i
p4 [gopts] ldap -o configname
p4 [gopts] ldap -d configname
p4 [gopts] ldap -t username configname
```

Description

The **p4 ldap** command includes five syntax variants:

- The first variant allows you to create or edit an LDAP configuration.
- The **p4 ldap -i** command allows you to read an LDAP configuration from standard input.
- The **p4 ldap -o** command allows you to display the specified LDAP configuration.
- The **p4 ldap -d** command allows you to delete the specified LDAP configuration.
- The **p4 ldap -t** command allows you to test an existing LDAP configuration.

Creating an LDAP Configuration

The LDAP configuration you create with the **p4 ldap** command defines an Active Directory or other LDAP server against which the Perforce server can authenticate users.

To create an LDAP configuration specification, you provide values that specify the host and port of the AD/LDAP server, bind method information, and security parameters. Bind methods can be one of the following:

- **Simple**: Uses a template based on the user's name to produce a distinguished name that the Perforce server attempts to bind against, validating the user's password. For example:

  ```
  uid=%user%,ou=users,dc=example,doc=org
  ```

- **Search**: Uses an LDAP search query to locate the user record. The search relies on a known base DN and an LDAP search query; you provide these using the `SearchBaseDN`, `SearchFilter`, and `SearchScope` fields of the LDAP configuration specification. This method might also required the full distinguished name and password of a known read-only entity in the directory. You supply these using the `SearchBindDN` and `SearchPasswd` fields of the LDAP configuration. Here is a sample search query:

```
BaseDN: ou=users,dc=example,dc=org
LDAP query: (uid=%user%)
```

- **SASL**: If the AD/LDAP server supports `SASL DIGEST-MD5`, this method defers the user search to the AD/LDAP server and does not require a distinguished name to be discovered before the bind is attempted. The user provides a user name, a password, and an optional realm.

In addition to creating the LDAP configuration, you must use the following configurables to enable the configuration and to further define the authentication process:

- `auth.ldap.order.N` - enables an AD/LDAP server and specifies the order in which it should be searched.
- `auth.default.method` - specifies whether new users should be authenticated by Perforce or using LDAP.
- `auth.ldap.userautocreate` - specifies whether new users should be automatically created on login when using LDAP authentication.
- `auth.ldap.timeout` - time to wait before giving up on a connection.
- `auth.ldap.cafile` - the path to a file used for certification when the AD/LDAP server uses SSL or TLS.
- `auth.ldap.ssllevel` - level of SSL certificate validation.

For more information, see "Configurables" on page 595.

Note	LDAP configurations are stored in the new `db.ldap` table. This table is journaled, so LDAP configurations are now included in checkpoints and are replicated.

Authentication is user-based:

- The LDAP authentication method is selected for each existing user with the `AuthMethod` field of the user specification. For more information, see the **`p4 user`** command.
- The authentication method applied to auto-created users (LDAP or Perforce) is determined by the `auth.userautocreate` configurable. For more information, see "Configurables" on page 595.

Here is a sample LDAP configuration:

```
Name:     sleepy
Host:     openldap.example.com
Port:     389
Encryption:    tls
BindMethod:    search
Options: nodowncase nogetattrs norealminusername
SimplePattern: someuserid
SearchBaseDN:  ou=employees,dc=example,dc=com
SearchFilter:  (cn=%user%)
SearchScope:    subtree
GroupSearchScope:  subtree
```

Testing an LDAP Configuration

You can use a command like the following to test an LDAP configuration:

```
$ p4 ldap -t userX myConfig
```

The command prompts you for a password and returns successfully if **userX** can be found. If the AD/LDAP server specified by **myConfig** is down, if the user can't be found, or if the password you supply is incorrect, the command returns a detailed error message. For example:

```
c:\temp> p4 -p 1666 ldap -t userX sleepy
Enter password:
Authentication as cn=userX,ou=employees,dc=example,dc=com
failed. Reason: Invalid Credentials
```

Form Fields

Field Name	Type	Description
`Name:`	Read only	The name of the LDAP configuration. Relevant to bind method: all
`Host:`	Writable	Fully qualified domain name of AD/LDAP server. The default is **localhost**. Relevant to bind method: all
`Port:`	Writable	The port to connect on. The default is **389**. Relevant to bind method: all
`Encryption:`	Writable	One of **none**, **ssl**, and **tls**. The default is **tls**. Relevant to bind method: all
`BindMethod:`	Writable	One of **simple**, **search**, and **sasl**. See "Creating an LDAP Configuration" on page 235 above for more details. Relevant to bind method: all
`Options`	Writable	Modifies the behavior of the LDAP integration that is specific to this configuration. Choose from the following: • `[no]downcase` specifies whether **p4 ldapsync -g** should downcase user names from the directory. For example, if user names in LDAP are ABrown, and SMITH, they are added to the group as **abrown** and **smith**.

Field Name	Type	Description
		• `[no]getattrs` specifies whether the `Fullname` and `Email` fields for users auto created with **`p4 login`** should be populated from the directory. This requires that you set the `AttributeName` and/or `AttributeEmail` fields in the ldap spec. See below. • `[no]realminusername` specifies whether the realm should be taken from the SASL username if it is in UNC or UPN format. That is, if your user names look like this: `user@realm` or `realm\user`, the user and realm are separated and passed separately. By default, these options are not set.
`SimplePattern:`	Writable	The distinguished name used to bind against to validate the user's credentials. The `%user%` placeholder is replaced with the user's `userId`. Relevant to bind method: simple
`SearchBaseDN:`	Writable	The distinguished name from which to start the search for the user object. Relevant to bind method: search
`SearchFilter:`	Writable	The LDAP query filter that identifies the user object to bind against. The `%user%` placeholder is replaced with the user's `userId`. Relevant to bind method: search
`SearchScope:`	Writable	One of the following: • `baseonly` - search just the `BaseDN` object. • `children` - search the `BaseDN` object and its direct children. • `subtree` - search the `BaseDN` object and all objects below it. Relevant to bind method: search
`SearchBindDN:`	Writable	The distinguished name to bind against in order to search the directory. Relevant to bind method: search
`SearchPasswd:`	Writable	The password for the `BindDN` record. You may quote this field; this allows special characters, like `#` to be used in the password.

Field Name	Type	Description
		Relevant to bind method: search
`SaslRealm:`	Writable	The optional realm to use when authenticating the user using SASL. Relevant to bind method: sasl
`GroupSearchFilter:`	Writable	The filter to use for the group search. Relevant to bind method: all
`GroupBaseDN:`	Writable	The search base for performing a group search. The default is the value of `SearchBaseDN`. Relevant to bind method: all
`GroupSearchScope`	Writable	One of the following, to be used when performing a group search. • `baseonly` - search just the `BaseDN` object. • `children` - search the `BaseDN` object and its direct children. • `subtree` - search the `BaseDN` object and all objects below it. Relevant to bind method: all
`AttributeUid`	Writable	The name of the attribute in the user object that contains the user's UID.
`AttributeName`	Writable	The name(s) of the attribute(s) in the user object that contains the user's full name. If multiple attributes are required to form the full name, specify each one surrounded by % symbols, so that expanding these forms the user's full name. The `getattrs` option must be enabled for this field to be populated using the value specified in the LDAP `AttributeName` field.
`AttributeEmail`	Writable	The name of the attribute in the directory's user object that contains the users' email addresses. The `getattrs` option must be enabled for this field to be populated using the value in the LDAP `AttributeEmail` field.

Options

`-d` *`config`*	Deletes the specified LDAP configuration.

`-i`	Read the LDAP specification from standard input.
`-o` *`config`*	Writes the specified LDAP configuration to standard output.
`-t` *`username config`*	Specifies a username to authenticate against the specified LDAP configuration; it is provided for testing purposes. The command returns a success message or a detailed error message. You do not have to enable the configuration to run this test.

Usage Notes

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

Examples

`p4 ldap myLdap`	Create the `myLdap` configuration.
`p4 ldap -o myLdap`	Write the `myLdap` configuration to standard output.
`p4 ldap -t Joanna myLdap`	Authenticate the user `Joanna` against the server specified by the `myLdap` configuration.
`p4 ldap -d myLdap`	Delete the `myLdap` configuration.

Related Commands

To view a list of all LDAP configurations.	**`p4 ldaps`**
To define LDAP-related configurables.	**`p4 configure`**

p4 ldaps

Synopsis

Display a list of LDAP configurations or attempt to authenticate a user against active configurations.

Syntax

```
p4 [g-opts] ldaps [-A]
p4 [g-opts] ldaps -t username
```

Description

The **p4 ldaps** command includes two syntax variants:

- The first variant allows you to display existing LDAP configurations; the **-A** option lists active configurations according to the priority set for them with the **auth.ldap.order.*n*** configurable.
- The second variant allows you to attempt to authenticate the specified user against all active configurations. This command tests each configuration whether the authentication succeeds or fails. That is, testing does not stop with the first successful authentication.

Listing configurations

If you do not use the **-A** option, **p4 ldaps** returns information about all configurations. If a configuration has not been assigned a priority using the **auth.ldap.order.*n*** configurable, it is shown to be disabled. Output includes the configuration name, the host and port of the AD/LDAP server, the bind method used, and whether the server is enabled.

```
c: \temp>p4 -p 1666 ldaps
bashful localhost:389 simple (disabled)
dopey localhost:389 sasl (enabled)
sneezy localhost:389 search (enabled)
```

If you use the **-A** option, only enabled servers are shown, and they are listed in the order in which they will be searched. For example:

```
c: \temp>p4 -p 1666 ldaps -A
sneezy localhost:389 search (enabled)
dopey localhost:389 sasl (enabled)
```

The order of the servers shown above are determined by the setting of the **auth.ldap.order.*n*** configurable; for example:

```
c: \temp>p4 -p 1666 configure show
auth.ldap.order.1=sleepy (configure)
auth.ldap.order.2=dopey (configure)
```

Testing active configurations

Using the `-t` option allows you to test all active configurations. A test might fail because a server is unavailable, because the user could not be found, or because the wrong credentials were submitted.

- Here is output from a successful authentication:

```
c:\temp> p4 -p 1666 ldaps -t myUser
Enter password:
Testing authentication against LDAP configuration sneezy
Authentication successful
Testing authentication against LDAP configuration dopey.
Authentication successful
```

- Here is output from a test that failed because the AD/LDAP servers were unavailable:

```
c:\temp> p4 -p 1666 ldaps -t myUser
Enter password:
Testing authentication against LDAP configuration sneezy.
Failed to initialize TLS: Server Down
Testing authentication against LDAP configuration dopey.
Failed to initialize TLS: Server Down
```

- Here is output when a bad password is given:

```
c:\temp> p4 -p 1666 ldaps -t myUser
Enter password:
Testing authentication against LDAP configuration sneezy
Authentication as abrown failed. Reason: Invalid Credentials
Testing authentication against LDAP configuration dopey
Authentication as abrown failed. Reason: Invalid Credentials
```

Options

`-A` *`config`*	Display command output according to the priority set with the `auth.ldap.order.`*`n`* configurable. This limits the configurations displayed to those that have been assigned a priority. If you omit this option, all active configurations are listed in alphabetical order.
`-t` *`username`*	Specifies a user name to authenticate against all active LDAP configurations; this option is provided for testing purposes.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Examples

`p4 ldaps`	List all configurations.
`p4 ldaps -t Joanna`	Authenticate the user `Joanna` against all active configuration.

Related Commands

To create, edit, delete, or test an LDAP configuration.	**`p4 ldap`**
To define LDAP-related configurables.	**`p4 configure`**

p4 ldapsync

Synopsis

Synchronize Perforce group memberships with LDAP groups.

Syntax

p4 [*gopts*] ldapsync -g [-n] [-i *N*] [*group* ...]

Description

The **p4 ldapsync** command updates the users in the specified Perforce groups to match the members in the corresponding LDAP groups. The correspondence between a Perforce group and an LDAP group is defined in the Perforce group spec. If you do not specify a group name, all groups with LDAP configurations are updated.

You can use the `[no]downcase` option in the ldap spec `options` field to specify whether **p4 ldapsync** should downcase user names (found by the group query) when updating user names.

For information about using the Perforce group spec to associate an LDAP group with a Perforce group, see "Authorization using LDAP groups" in the *Helix Versioning Engine Administrator Guide: Fundamentals*.

You can synchronize once or at a given interval. To enable periodic synchronization, you must add the **p4 ldapsync** command as a startup command as follows:

1. Check that the Perforce server has its server id set. Use the **p4 serverid** command to check.
2. If the server has no server id, assign one using a command like the following:

   ```
   $ p4 serverid my-server
   ```

3. Use the **p4 configure show** command to check which startup configurables are already being used. Select the next available number. For example, if six startup configurables are being used, you can set the startup configurable that runs the **p4 ldapsync** command as number 7:

   ```
   $ p4 configure set "my-server#startup.7=ldapsync -g -i 1800"
   ```

   This command will update all groups with valid LDAP synchronization configurations every 1800 seconds (30 minutes).

Options

`-g`	Required to specify groups.

`-i N`	Execute the command every *N* seconds. If this option is not specified, the command executes once and exits.
`-n`	Preview the operation and show the groups that would be affected without taking any action.
group	The name of a Perforce group that must be updated when changes to the corresponding LDAP group take place. If no group names are specified, all groups with LDAP configurations are updated.

Usage Notes

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

Examples

`p4 ldapsync -g`	Updates all groups for which LDAP configurations have been defined.

Related Commands

To view a list of all LDAP configurations	**`p4 ldaps`**
To create or edit an LDAP configuration	**`p4 ldap`**
To define LDAP-related configurables	**`p4 configure`**
To define LDAP configurations for a Perforce group spec	**`p4 group`**

p4 license

Synopsis

Update or display the license file.

Syntax

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

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 service without having to shut down the service and manually copy the license file into the server root.

Most new license files obtained from Perforce can be installed with **p4 license**, (or by copying over the existing license file) except for when the service's IP address or port has changed. If either the server IP address or port number has changed, restart the service with **p4 admin restart**.

For releases 2016.1 and later, without a valid license, the versioning service limits itself to either 5 users and 20 client workspaces (and unlimited files), or to an unlimited number of users and workspaces (but with a limit of 1,000 files).

Versions of the server prior to 2016.1 will continue to support the 20/20 usage allowance: 20 users and 20 client workspaces (and unlimited files), or an unlimited number of users and workspaces (but with a limit of 1,000 files)

Options

`-o`	Display the current license file on the standard output.
`-i`	Read in a new license file from the standard input.
`-u`	Report license limits and show how many entities (users or files) are in use with respect to these limits.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
		(`admin` for `-u` option)

Examples

`p4 license -o`	Display the current license file on the standard output.
`p4 license -i`	Read in a new license file from the standard input.

p4 list

Synopsis

Create a temporary list of files that can be used as a label.

Syntax

```
p4 [g-opts] list [-l labelname] [-C -M] file[revRange] ...
p4 [g-opts] list -l labelname -d [-M]
```

Description

This command is intended for use by systems integrators and third-party developers.

p4 list builds an in-memory temporary list of files that can be used as a label for the duration of the single **p4** command session that created it. The list exists only as long as the connected session; the temporary list created by running **p4 list** from the command line is not available to subsequent **p4** commands.

By default, the head revision is listed. If the file argument specifies a revision, all files at that revision are listed. If the `file` argument specifies a revision range, the highest revision in the range is used for each file.

The `-d` option is handy for long-running processes that need to use and reuse lists within the scope of one session without exhausting the server's process memory.

Options

`-C`	Limits any depot paths to those that can be mapped through the client workspace.
`-d` *labelname*	Delete the specified list.
`-l` *labelname*	Specify the label to be applied to file revisions. If a label of that name already exists, the in-memory name has precedence over the stored one. If you do not use this option, the **p4 list** command assigns a unique name to the temporary list and returns the name as output.
`-M`	When run against a forwarding replica, forward the **p4 list** command to the master server.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 lock

Synopsis

Lock an opened file against other users submitting changes to the file.

Syntax

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

Description

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

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

In an edge/commit architecture, use the **-g** flag to lock the files locally and globally. This syntax variant may only be used from an edge server, and it must be used with the **-c** *changelist* option. This lock is removed by the **p4 unlock -g** command or by any submit command for the specified changelist.

Options

-c *changelist*	Lock only files included in changelist *changelist*
-g	Lock files in an edge/commit architecture. See "Description" above.
g-opts	See "Global Options" on page 571.

Usage Notes

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

If the server configurable **server.locks.global** is set to **1**, when **p4 lock** is issued on an edge server it takes global locks on the commit server by default.

Related Commands

To unlock locked files	**p4 unlock**

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

p4 lockstat

Synopsis

Report lock status of database tables.

Syntax

```
p4 [g-opts] lockstat [-c client | -C]
```

Description

By default, the **p4 lockstat** command reports any database tables that are currently locked for a read or write operation.

Options

`-c client`	Report whether or not the specified client workspace is locked for a read or write operation.
`-C`	Report all client workspaces that are locked for read/write operations.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 logappend

Synopsis

Add a line to any user log files.

Syntax

p4 [*g-opts*] logappend [-a *args...*]

Description

The **p4 logappend** command appends a line to any structured log file that includes user log events. Up to 25 arguments may be supplied per line.

Options

`-a` *`args...`*	Up to 25 arguments to be appended to the `user` log.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- At present, the only two log files that include user log events are `user.csv` and `all.csv`.

p4 logger

Synopsis

Report changed jobs and changelists.

Syntax

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

Description

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

Options

-c *sequence#*	List all events happening after this sequence number.
-t *countername*	List all events after this counter number.
-c *sequence#* **-t** *countername*	Update the supplied counter with the current sequence number and clear the log; as this clears the log regardless of which counter name is specified, only one user can make use of this option.
g-opts	See "Global Options" on page 571.

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.
- In distributed environments, **p4 logger** commands should be issued to the Commit Server, not to an Edge Server. If you are using P4DTG or other third-party tools that make use of this command, ensure that your installation is properly configured.

Related Commands

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

To see full information about a particular changelist	**p4 describe**
To see a list of all changelists, limited by particular criteria	**p4 changes**

p4 login

Synopsis

Log in to the Perforce service by obtaining a ticket.

Syntax

```
p4 [g-opts] login [-a -p] [-h host] [user]
p4 [g-opts] login [-s]
p4 [g-opts] login [-r remote spec]
```

Description

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

By default, tickets are valid for 12 hours. This default value is defined on a per-group basis in the **p4 group** form.

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

Login attempts, whether successful or not, are logged to a structured log file. If the login fails, the reason for failure is included in the log. In the case of authentication triggers or LDAP authentication, the parts of the error message that contain user data are sanitized as needed.

If you use LDAP authentication, you can set the `getattrs` option in the ldap spec `options` field to specify whether the `Fullname` and `Email` fields for users created by **p4 login** should be populated from the directory.

Options

`-a`	Obtain a ticket that is valid for all IP addresses.
`-h host`	Request a ticket that is valid for the specified host IP address.
`-p`	Display the ticket, rather than storing it in the local ticket file.
`-s`	Display the status of the current ticket, if one exists.
`-r`	Specify the remote server to which the login should be forwarded. If the remote spec passed in contains a `RemoteUser` entry, the login is performed for that user.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- To create tickets that do not expire, set the timeout value to **`unlimited`** in the **`p4 group`** form.
- By default, after three failed login attempts, a user must wait 10 seconds before logging in again. Perforce superusers can control the number of login attempts by setting the **`dm.user.loginattempts`** configurable.
- To extend a ticket's lifespan, use **`p4 login`** while already logged in. Your ticket's lifespan is extended by 1/3 of its initial timeout setting, subject to a maximum of your ticket's initial timeout setting.
- Perforce superusers can obtain login tickets for users other than themselves without entering passwords. Non-superusers who attempt to log in as other users must use the **`p4 -u`** ***`username`*** **`login`** form of the command, and correctly supply the other user's password.
- Tickets are stored in the file specified by the **`P4TICKETS`** environment variable. If this variable is not set, tickets are stored in **`%USERPROFILE%\p4tickets.txt`** on Windows, and in **`$HOME/.p4tickets`** on other operating systems.
- The **`-h`** option causes the service to issue a ticket that is valid on the specified host IP address. This option is typically used with **`-p`** to display a ticket that can subsequently be used on another machine.
- In replicated environments, logging in to the master server does *not* log you in to any replica servers.

Examples

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

Related Commands

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

p4 logout

Synopsis

Log out of Perforce by removing or invalidating a ticket.

Syntax

p4 [*g-opts*] logout [-a] [*username*]

Description

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

If you use **p4 logout -a**, the ticket remains in the ticket file, but is invalidated on the service: all users of the ticket are logged out simultaneously. You can also remove a single user's ticket with the **-a** *username* option.

Options

`-a`	Log out all users of the ticket by invalidating the ticket on the service. If a username is specified, that user is logged out. You must have super user access to be able to log out a user other than yourself.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`, `super` to use `-a` *username*

- Tickets are stored in the file specified by the P4TICKETS environment variable. If this variable is not set, tickets are stored in `%USERPROFILE%\p4tickets.txt` on Windows, and in `$HOME/.p4tickets` on other operating systems.
- In replicated environments, logging out of the master server with **p4 logout -a** also logs you out of any replica servers.

Examples

p4 logout	Log out of Perforce by removing the local session ticket.

p4 logout -a	Log out of Perforce by removing the local session ticket and instructing the Perforce service to invalidate the ticket on all other workstations from which they were logged in.

Related Commands

To start a login session (to obtain a ticket)	**p4 login**
To display tickets	**p4 tickets**

p4 logparse

Synopsis

Parse a structured log file and return log data.

Syntax

p4 [*g-opts*] logparse [-e] [-T *fields...*] [-F *filter*] [-s *offset*] [-m *max*] *logfile*

Description

The **p4 logparse** command parses the indicated structured *logfile* and returns the log data in tagged format.

Structured logs differ from the basic error log (P4LOG) and audit log (P4AUDIT). To read the basic error log, use the **p4 logtail** command.

Valid names for structured log files are shown in the table below:

`all.csv`	All loggable events (commands, errors, audit, etc.)
`audit.csv`	Audit events (audit, purge)
`auth.csv`	Information about user login attempts.
`commands.csv`	Command events (command start, command compute, command end)
`errors.csv`	Error events (errors-failed, errors-fatal)
`track.csv`	Command tracking (track-usage, track-rpc, track-db)
`user.csv`	User events; one record every time a user runs **p4 logappend**.
`events.csv`	Server events (startup, shutdown, checkpoint, journal rotation, etc.)
`integrity.csv`	Major events that occur during replica integrity checking.

To enable structured logging, set the `serverlog.file.`*n* configurable(s) to the name of the file. For example:

```
$ p4 configure set serverlog.file.2=commands.csv
$ p4 configure set serverlog.file.3=errors.csv
$ p4 configure set serverlog.file.5=audit.csv
```

Numbers provided for the configurables do not have to be consecutive. A given number may not exceed 500; this assignment will return an error:

```
$ p4 configure set serverlog.file.666=commands.csv
```

Structured log files are automatically rotated on checkpoint, journal creation, overflow of associated `serverlog.maxmb.n` limit (if configured), and the **`p4 logrotate`** command.

Options

`-e`	Display special characters as hex-encodings.
`-F filter`	Limit output to records that match the filter pattern.
`-m max`	Limit the number of lines returned.
`-s f_offset`	Start parsing at the given file offset as returned in the `f_offset` field.
`-T fields...`	Limit displayed fields to those listed.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 logrotate

Synopsis

Rotate one or more structured log files.

Syntax

p4 [*g-opts*] logrotate [-l *logname*]

Description

The **p4 logrotate** command rotates the named logfile, or rotates all structured logs if the **-l *logname*** option is not supplied.

If the relevant configurables are set, structured log files automatically rotate when they grow to **serverlog.maxmb.*n*** megabytes in length, and the past **serverlog.retain.*n*** log files are preserved.

By default, structured logs have no maximum size limit, and automatically rotate only on checkpointing and journaling events.

Options

-l *logname*	Rotate the log named *logname*.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 logschema

Synopsis

Describe the schema of a log record type.

Syntax

```
p4 [g-opts] logschema [-a | recordtype]
```

Description

The **p4 logschema** command returns a description of the specified log record type, in tagged format.

Options

`-a`	Display the specification of every known log record type.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 logstat

Synopsis

Report size of journal, error log, and/or audit log files; or report size of the specified structured file.

Syntax

p4 [*g-opts*] logstat [-s | -l *logname*]

Description

If no options are specified the **p4 logstat** command reports the sizes of the journal, error log (if it exists), and audit log (if it exists).

- Use the **-l** *logname* option to display the file size of the specified log.
- Use the **-s** option to report the file size for each of the structured log files defined for this server.

The following two sample commands illustrate the different output due to the use of the **-s** option.

```
C:\temp\logs> p4 logstat
journal 2591 bytes
out 126 bytes
C:\temp\logs> p4 logstat -s
journal 2591 bytes
out 255 bytes
all.csv 13599 bytes
commands.csv 11321 bytes
path/to/log/files/all.csv 13599 bytes
```

See the *Helix Versioning Engine Administrator Guide: Fundamentals* for more information about structured logs.

Options

-l *logname*	Display the file size of the named *logname*. Valid values for *logname* are **journal**, **errorLog**, and **auditLog**, or any of the **serverlog.file.***n* filenames associated with structured logs.
-s	Report the file size for each of the structured log files defined for this server.
g-opts	See "Global Options" on page 571.

Usage Notes

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

p4 logtail

Synopsis

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

Syntax

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

Description

The **p4 logtail** command displays the last block(s) of the error log, and the offset for the next block, when available.

Output consists of a series of lines in tagged format. The first line is "`... file LOG`", followed by multiple blocks of log data. By default, all blocks from the *start_offset* are output until the end of the file. The data is returned in blocks of size *blocksize*, each of which is tagged with "`... data`". The last line is "`... offset`*next_offset*", where *next_offset* is the offset in the logfile from which the next block of data is to be retrieved.

If you specify the name of an error log that has an associated counter, the **p4 logtail** command returns the current value of that counter. It also returns the current size of the log, at the end of the output (along with the ending offset in the log). The size and offset are the same if the command reads to the end of the log. For more information about counters, see "Logging and structured files" in the chapter "Administering Perforce: Superuser Tasks" in the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

`-b` *blocksize*	The block size, in bytes. The default is 8192 bytes.
`-l` *log*	If specified, the name of the log to display.
`-m` *maxBlocks*	The maximum number of blocks to output. Ignored unless `-s` is also specified.
`-s` *start*	The offset (from the beginning of the file), in bytes.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- For more about administering Perforce, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Examples

`p4 logtail -b 1024 -m 2`	Display the last two kilobytes of the log file, as two separate blocks of 1,024 bytes each.

p4 merge

Synopsis

Merge one set of files into another.

Syntax

```
p4 [g-opts] merge [-c change] [-m max] [-n -Ob -q -F] [--from stream]
[toFile[fromRevRange]]
p4 [g-opts] merge [-c change] [-m max] [-n -Ob -q] fromFile[fromRevRange] toFile
```

Description

The **p4 merge** command is a simplified form of the **p4 integrate** command: it merges a set of changes from source to target files. The command outputs the scheduled resolves. This command is intended for use with streams and distributed version control, but is also perfectly usable for traditional Perforce branches.

- Use **p4 resolve** to resolve all changes. Then use **p4 submit** to commit merged files to the depot. Unresolved files may not be submitted.
- Use **p4 shelve** to shelve merged files or **p4 revert** to delete them.
- Use the **p4 integrated** and **p4 filelog** to display merge history.

Using the client workspace as a staging area, **p4 merge** schedules all affected target files to be resolved per changes in the source. Target files outside of the current client view are not affected. Source files need not be within the client view.

Each file in the target is mapped to a file in the source. Mapping adjusts automatically for files that have been moved or renamed, as long as **p4 move** was used to move or rename the files. The scope of source and target files sets must include both old-named and new-named files for mappings to be adjusted. Moved source files may schedule moves to be resolved in target files. You can limit the revisions to be merged using the *fromRevRange* parameter. If the scope does not include both old and new files, for example, if you run the merge on a single file that is either the move/add or move/delete of the move pair of actions, an error message is shown.

With streams, you use **p4 merge** to keep a child stream up to date with a more stable parent stream. This ensures that when you promote changes back to the stable parent, you do not inadvertently overwrite any other changes that were checked into the parent. Files are opened in a pending changelist and scheduled for resolve as required. To update the parent stream, resolve and submit. By default, **p4 merge** merges changes into the current stream from its parent, or from another stream specified by the **--from** option. The source and target can also be specified on the command line as a pair of file paths. More complex merge mappings can be specified using branch specifications as with **p4 integrate**. Use the **-F** option to force merging against a stream's expected flow. You can also use this option to force the generation of a branch view based on a virtual stream; the mapping itself refers to the underlying real stream.

In most cases, you can use the **p4 merge** and **p4 copy** commands to propagate changes between streams (or branches).

If you specify no arguments for the command, the target defaults to the current stream, and the source defaults to the current stream parent. You can specify a different source with `--from stream_name`, which is an alias for the `-P` option. You can specify the stream as a directory name relative to the current stream depot: for example, `--from main` instead of `--from //Ace/main`.

The **`p4 integrate`** and **`p4 merge`** commands select (as the base) the revision with the most edits in common with the source and the target.

Options

`-c change`	Specifies an existing pending changelist in which the files are to be opened.
`-F`	Force merge operation; perform the operation when the target stream is not configured to accept a merge from the source. To determine a stream's expected flow of change, use **`p4 istat`**.
`--from stream`	Specifies a stream other than the parent stream to merge from.
`-m max`	Limits the number of files merged. This option is useful for scripts that integrate large number of files; it enables them to batch the integrations and minimize the locking-related impact to other users of the shared versioning service.
`-n`	Preview the merge.
`-Ob`	The `-Ob` option displays the base revision for the merge (if any) along with each scheduled resolve.
`-q`	Quiet mode; suppresses normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Examples

```
p4 merge -r -S //projectX/bruno_dev
p4 resolve
p4 submit -d "Update dev w latest changes"
```

Incorporate the latest changes from main (parent) into Bruno's development stream (child).

Related Commands

Promote changes to more stable neighbor stream	**p4 copy**
Propagate changes	**p4 integrate**
Resolve file conflicts	**p4 resolve**

p4 monitor

Synopsis

Display Perforce process information; control long-running tasks.

Syntax

```
p4 [g-opts] monitor show [-a -l -e -L -s R | T | P | B | F | I ]
p4 [g-opts] monitor terminate id
p4 [g-opts] monitor clear [id | all]
p4 [g-opts] monitor pause id
p4 [g-opts] monitor resume id
```

Description

p4 monitor allows a system administrator to observe and control Perforce-related processes running on a Perforce server machine. Processes are tracked using a dedicated table that is constantly updated; this has a minor impact on server performance.

To use **p4 monitor**, you must enable monitoring on the Perforce service by setting the `monitor` configurable with **p4 configure**. You control process monitoring by setting the `monitor` configurable to one of the following values:

- 0 — disable monitoring
- 1 — enable monitoring of active processes
- 2 — enable monitoring of both idle connections and active processes
- 3 — enable idle connections stuck at Init(), idle connections and active commands.
- 5, 10, and 25 — obtain lock information. See the description of the `L` option for more information.

Changes to the `monitor` configurable affect all new **p4** processes that connect to the server; restarting the server is not required.

Command syntax variants provide the following alternatives:

- To list current process information, use **p4 monitor show**. By default, all processes are listed, but only the command (for example, **sync**, **edit**, **submit**) is shown, without arguments. This form of **p4 monitor** requires `list` level access. Use the **-s** *status* option to restrict the display to processes in the specified state.

  To show the list of arguments associated with each command, use the **-a** (arguments) option or **-l** (long) option. For additional information from the user environment, use the **-e** (environment) option. These options require `super` level access. Use the **-L** option to show locked files.

- To mark a process for termination, use **p4 monitor terminate** *id*. This command requires that the user be an operator or have `super` level access.

The **p4 monitor terminate** command will not mark a process for termination unless the process has been running for at least ten seconds. Some commands, such as **p4 obliterate**, cannot be terminated.

- To 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. (If a command terminates prematurely on the server side, it may be erroneously listed as running. You can clear such processes with **p4 monitor clear**.)

  Requires that the user be an operator or have **super** access.

  Processes marked as running continue to run to completion even if removed from the monitor table with **p4 monitor clear**.

- To control long-running tasks such as **p4 verify** or **p4 pull**, use the **p4 monitor pause** and **p4 monitor resume** commands.

  p4 monitor pause and **p4 monitor resume** require that the user be an operator or have **super** access.

Output format

Each line of **p4 monitor** output consists of the following fields:

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

pid	The process ID under Unix (or thread ID under Windows)
status	**R**, **T**, **P**, **B**, **F**, or **I**, depending on whether the process is *R*unning, marked for *T*ermination, *P*aused, *B*ackground, *F*inish or *I*dle. *F*inish and *B*ackground occur only in replica servers.
owner	The Perforce user name of the user who invoked the command.
hh:mm:ss	The time elapsed since the command was called.
command [*args*]	The command and arguments as received by the Perforce service.

For example, consider the following output to the **p4 monitor show -L** command, which displays information about locked files:

```
8764 R user 00:00:00 edit
     [server.locks/clients/88,d/ws4(W),db.locks(R),db.rev(R)]
8766 R user 00:00:00 edit
     [server.locks/clients/89,d/ws5(W),db.locks(R),db.rev(R)]
8768 R user 00:00:00 monitor
```

Following pid, status, owner, and time information, this shows two edit commands that have various files locked, including the client workspace lock in exclusive mode for the workspaces **ws4** and **ws5**, and db.locks and db.rev tables in read-only mode.

If you have enabled idle process monitoring (by setting the `monitor` configurable to 2), idle processes appear with a *`status`* of `R`, but with a *`command`* of `IDLE`.

Some commands (for instance, **`p4 submit`**) invoke multiple processes. For example, `dm_CommitSubmit` or `dm_SubmitChange` may appear in the output of **`p4 monitor`** as two separate phases of the **`p4 submit`** command.

Getting pull thread information for replicas

If you are running a replica with monitoring enabled and you have not configured the monitor table to be disk-resident, you can run the following command to get more precise information about what pull threads are doing. (Remember to set `monitor.lsof`).

```
$ p4 monitor show -sB -la -L
```

Command output would look like this:

```
31701 B uservice-edge3 00:07:24 pull sleeping 1000 ms
    [server.locks/replica/49,d/pull(W)]
```

Options

You must be an operator or have `super` access to use the following options.

`-a`	Show all arguments associated with the process (for example, `edit file.c`, or `sync -f //depot/src/...`). Perforce user names are truncated to 10 characters, and each line is limited to a total of 80 characters of output. Requires that the user be an operator or have `super` access.
`-e`	Show environment information including invoking Perforce application (if known), host IP address, and workspace name. Requires that the user be an operator or have `super` access.
`-l`	Show all arguments in long form; that is, without truncating user names or the list of command line arguments. Requires that the user be an operator or have `super` access.
`-L`	Show information about locked files. The information is collected only for the duration of the **`p4 monitor`** command, and is not persisted. Requires that the user be an operator or have `super` access. Pre-requisites for using this option vary with the platform on which the server is running.

- On Unix platforms, you must set the `monitor.lsof` configurable to the following value:

  ```
  $ path/lsof -F pln
  ```

  The value for *path* varies with the version of Unix you are using. For example, `/usr/bin/lsof`.

  There are circumstances in which `monitor.lsof` might not work for you: your Linux machine does not support `lsof`, the version of `lsof` might not work with the Perforce server, or the administrator might not be willing to run the `lsof` command for security reasons. If this is the case, you can still get information about locked files by setting the `monitor` configurable, described next.

- On non-Unix platforms or if `monitor.lsof` cannot be used, you must set the `monitor` configurable to 5, 10, or 25:

  `5`: monitor both active commands and idle connections, including a list of the files locked by the command for more than one second.

  `10`: monitor both active commands and idle connections, including a list of the files locked by the command for more than one second, with lock wait times included in the lock information.

  `25`: monitor both active commands and idle connections, including a list of the files locked by the command for any duration, with lock wait times included in the lock information.

  Using the `monitor` level to display information about locked files has a non-trivial impact on performance; the `monitor.lsof` option is preferred for Unix platforms.

You can use the `-z tag` option with this option. In that case, the **`p4 monitor show`** command will return one lockinfo tag for each file that the process has locked.

`-s` *`status`*	Restrict the display to processes in the `R`unning, `T`erminated, `P`aused, `B`ackground, `F`inish, or `I`dle states. Requires that the user be an operator or have `super` access.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Examples

p4 monitor show	Show Perforce processes information (commands only). Requires **list** access only.
p4 monitor show -l	Show arguments and commands, without limits on line length. Requires **super** access.
p4 monitor show -a	Show arguments and commands, limited to 80 characters per line of output. Requires **super** access.
p4 monitor terminate 123	Instruct the Perforce service to mark process 123 for termination. Requires **super** access.
p4 monitor clear all	Clears the monitor table of all entries. Requires **super** access.

Related Commands

To turn on monitoring	**p4 configure set monitor=1**
To turn off monitoring	**p4 configure set monitor=0**

p4 move

Synopsis

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

Syntax

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

Description

The **p4 move** command takes a file already opened for edit or add and moves it to the destination provided.

An open file can be moved many times before it is submitted; moving a file back to its original location undoes the pending move, leaving it open for edit. Using **p4 revert** on a moved file both undoes the move and reverts the unsubmitted content.

Options

-c *change*	If a changelist number is provided, the files are opened in the numbered pending changelist.
-t *filetype*	If a filetype is specified, the file is reopened as the new filetype.
-f	Force a move to an existing target file. The file must be synced, but not opened. The originating source file will no longer be synced to the workspace. If you use **p4 move -f**, you will need to resolve the move before submitting the changelist.
-k	Keep existing workspace files by bypassing the renaming in the client workspace. Use **p4 move -k** only in the context of reconciling work performed while disconnected from the Perforce service.
-n	Preview the move that would be performed, without actually moving files.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
fromFile: Yes *toFile*: No	No	**read** access for *fromFile* **write** access for *toFile*

- Files must be open for `add` or open for `edit` before they can be moved.
- To move and resolve a file that is open for edit but has been renamed at the head revision, you can use the `-f` option to force the move.

Examples

`p4 move file1.c file2.c`	Assuming that `file1.c` is open for add or edit, move `file1.c` to `file2.c`.
`p4 move //depot/d1/... //depot/d2/...`	Moving open files from directory `d1` to directory `d2`.

p4 obliterate

Synopsis

Removes files and their history from the depot.

Syntax

```
p4 [g-opts] obliterate [-y -A -b -a -h] file[revRange] ...
```

Warning

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

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

Description

`p4 obliterate` can be used by Perforce administrators to permanently remove files from the depot. The file is removed from the Perforce service, along with all associated metadata, including references to the file in labels, the have list, and so on. After **`p4 obliterate`** completes, it appears to the service as if the affected file(s) had never existed. Copies of files in user 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`** option 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.

Note	**`p4 obliterate myfile`** does not obliterate a shelve of the file (archive or metadata). When you attempt to unshelve a file that has been obliterated, you will get an error. To recover the content of that file, print the file. To get rid of the shelve, delete the shelf.

`p4 obliterate` with a file spec does not obliterate a shelve of the file.

Options

`-y` *`filespec`*	Perform the obliterate operation. Without this option, **`p4 obliterate`** merely reports what it would do.

`-A`	Obliterate a revision marked for archive. By default, archived revisions are skipped.
`-b`	Restrict files in the argument range to those that are branched, and to files that are both the first revision and the head revision. This option is useful for removing old branches (where only one revision exists) while preserving files that have been modified post-branch. You can greatly improve the performance of **`obliterate -b`** by using the `-a` option with `-b`.
`-a`	Skip the (potentially resource-intensive) search of **`db.archmap`** and do not remove the file from the server; only its metadata. Although the file is not removed from disk, you can use `-a` in conjunction with `-b` to speed up obliteration of branched files known to exist only as lazy copies.
`-h`	Skip the search of **`db.have`** when looking for matching records to delete. The next time a client workspace that refers to these files is synced, any such files in the workspace will also be removed from the workspace. (This is often the desired behavior, for example, in client workspaces on build machines.)
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- **`p4 obliterate`** is one way to reclaim disk space from files that are no longer required, or to clean up mistakes made by users who create file hierarchies in the wrong place. Do not use operating system commands (**`erase`**, **`rm`**, and their equivalents) to remove files from the Perforce server root by hand.
- A better way to save disk space is to relocate infrequently-accessed files onto lower-performance (or detachable) storage. Instead of obliterating files outright, consider using **`p4 archive`** and **`p4 restore`** in conjunction with an archive depot. With archive depots, file history is preserved and available to all users, and file contents may be moved to offline or near-line storage.
- Obliterating files can alter the behavior of user commands. Syncing to an obliterated file does not remove the file from your client workspace, because the file is no longer part of any client workspace. Syncing to an obliterated revision of a file will either report that the file does not exist (if all revisions were obliterated), or provide you with the most recent non-obliterated revision of the file.
- Obliterating files in revision ranges can 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 2011/04/18 by user@dev1 (binary) 'Fixed'
... #1 change 1231 add on 2011/04/12 by user@dev1 (binary) 'First try'
```

In this case, 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 encounter difficulty.

Examples

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

Related Commands

To mark a file deleted at its head revision but leave it in the depot. This is the normal way of deleting files.	**p4 delete**
Instead of obliterating files, you can save disk space on a local depot by archiving some of its revisions to an archive depot. History of changes to these files is preserved.	**p4 archive**
To restore archived revisions from an archive depot. (You cannot restore obliterated files, but you can restore archived files.)	**p4 restore**

p4 opened

Synopsis

List files that are open in pending changelists.

Syntax

```
p4 [g-opts] opened [-a -c change] [-C workspace] [-u user -m max -s -g] [file ...]
p4 [g-opts] opened [-a -x] [-m max] [file ...]
```

Description

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

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

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

```
depot-file#rev - 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
- *type* is the type of the file at the given revision.
- If the file is locked (see **p4 lock**), a warning that it is `*locked*` appears at the line's end
- Files with filetypes that use the `+l` modifier are exclusively-locked (see the example for **p4 typemap**) and are displayed with a lock status of `*exclusive*`.

You can use the `-s` option to provide shortened output that omits the *#rev* number and the (*type*) of the file. This form of the command typically runs faster than the default.

The `-u` option limits output to files opened by a particular user. Otherwise, the command applies to all users.

The `-g` option lists files that are opened on a commit server in a distributed installation.

The following examples show how user details are included in command output. In the following, information is about the current workspace and the current user is not identified. However, other users with files opened are identified.

```
$ p4 opened
//depot/file-1.txt#1 - edit default change (text) by adam@scratch
//depot/file-2.txt#1 - edit default change (text)
```

In this example, which asks for all workspaces, user names are always displayed:

```
$ p4 opened -a
//depot/file-1.txt#1 - edit default change (text) by adam@scratch
//depot/file-2.txt#1 - edit default change (text) by normal@scratch
```

Options

`-a`	List opened files in all client workspaces. In distributed environments, this option lists files opened on all workspaces on your edge server.
`-c` *`change`*	List the files in pending changelist *`change`*. To list files in the default changelist, use **`p4 opened -c default`**.
`-C` *`workspace`*	List only files that are open in the specified client *`workspace`*.
`-g`	List files that are opened on the commit server in a distributed installation. This allows you to track files that might be locked globally on the commit server from an edge server. Because the command query runs on the commit server, if you provide a file spec argument, it must be in depot syntax. The `-g` option implies the `-a` option.
`-m` *`max`*	List only the first *`max`* open files.
`-s`	Short output; do not output the revision number or file type. This option is more efficient, particularly when using the `-a` (all-workspaces) option at large sites.
`-u` *`user`*	List only those files that were opened by *`user`*. Note that this option lists files opened in any workspace for the specified user name, not just the current workspace.
`-x`	In distributed environments, list all open files that have the `+l` filetype (exclusive open) over all servers. This option implies the `-a` option.
`g-opts`	See "Global Options" on page 571.

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**.

- **p4 opened -a** can have a performance impact on large sites; unless you need the exact revision number or file type of an opened file, best practice is to use **p4 opened -as**.
- **p4 opened** does not show files in shelved changelists. To display shelved changelists, use **p4 changes -s shelved**, and then use **p4 describe -s -S *changelist*** to display the files in the selected changelist(s).

Examples

p4 opened -c 35 //depot/main/...	List all files in pending changelist 35 that lie under the depot's `main` subdirectory.
p4 opened -a -c default	List all opened files in the default changelists for all client workspaces.

Related Commands

To open a file in a client workspace and list it in a changelist	**p4 add** **p4 edit** **p4 delete** **p4 integrate**
To move a file from one changelist to another	**p4 reopen**
To remove a file from all changelists, reverting it to its previous state	**p4 revert**
To create a new, numbered changelist	**p4 change**

To view a list of changelists that meet particular criteria	**p4 changes**

p4 passwd

Synopsis

Change a user's Perforce password.

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** option. To prevent another user from impersonating you, use **p4 passwd** to set your password.

After you have set a password, you can authenticate with the password by providing it whenever in one of three ways:

- Set the environment variable P4PASSWD to the password value;
- Create a setting for P4PASSWD within the P4CONFIG file;
- Use the **-P** *password* option on the 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 the security level configured for your installation.

For Perforce applications on Windows and OS X that connect to Perforce services at security levels 0 and 1, **p4 passwd** stores the password by using **p4 set** to store the MD5 hash of the password in the registry or system settings. When connecting to Perforce services at security levels 2, 3, or 4, password hashes are neither stored in, nor read from, these locations.

You can improve security by using ticket-based authentication instead of password-based authentication. To authenticate with tickets instead of passwords, first set a password with **p4 passwd**, and then use the **p4 login** and **p4 logout** commands to manage your authentication.

You can further improve security by assigning users to groups and setting the **PasswordTimeout:** field in the **p4 group** form. If a user belongs to more than one group, the largest **PasswordTimeout** value applies.

Perforce superusers can reset the passwords of individual users (or all users site-wide) with the **p4 admin resetpassword** command. You can also set the **dm.user.resetpassword** configurable (set with **p4 configure**) to require that any newly-created users reset the password you assigned them when you created their account.

For more about how user authentication works, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Certain combinations of security level and Perforce applications releases require users to set "strong" passwords. A password is considered strong if it is at least `dm.password.minlength` (by default, 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 (by default) considered strong. For information about how higher security levels work, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

`-O` *`oldpassword`*	Avoid prompting by specifying the old password on the command line. This option is not supported if your site is configured to use security level 2, 3, or 4. If you use the `-O` option, you must use the `-P` option.
`-P` *`newpassword`*	Avoid prompting by specifying the new password on the command line. This option is not supported if your site is configured to use security level 2, 3, or 4.
`user`	Superusers can provide this argument to change the password of another user.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- Passwords can be up to 1,024 characters in length. As of Release 2013.1, password length is configurable by setting the `dm.password.minlength` configurable. To require passwords to be at least 16 characters in length, a superuser can run:

  ```
  $ p4 configure set dm.password.minlength=16
  ```

  The default minimum password length is eight characters.

- The **`p4 passwd`** command never sends plaintext passwords over the network; a challenge/response mechanism is used to send the encrypted password to the service.

- Passwords can contain spaces; command line use of such passwords requires quotes. For instance, to pass the password `my passw`, to Perforce, use **`p4 -P "my passw"`** ***`command`***.

- If a user forgets his or her password, a Perforce superuser can reset it by specifying the username on the command line: **p4 passwd *username***
- To delete a password, set the password value to an empty string. Depending on your site's security level, your Perforce service may not permit you to set a null password.
- If you are using ticket-based authentication, changing your password automatically invalidates all of your tickets and logs you out; that is, changing your password is equivalent to **p4 logout -a**.

Related Commands

To change other user options	**p4 user**
To change users' access levels	**p4 protect**
To log in using tickets instead of passwords	**p4 login**
To force password reset	**p4 admin resetpassword**

p4 ping

Synopsis

Test network performance.

Syntax

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

Description

p4 ping simulates Perforce network traffic by sending messages from the versioning service to the Perforce application and back, and times the round trips. Round-trip times are reported in milliseconds. Because the round-trip time is typically too fast to measure for a single message, you can specify a message *count* per test.

Options

`-c count`	Number of messages per test.
`-f`	Flood mode: the service transmits continuously, sending the next message without waiting for the Perforce application to confirm receipt of the prior message.
`-i iterations`	Repeat the test for the specified number of *iterations*.
`-p pausetime`	Pause for *pausetime* seconds between tests, up to 120 seconds. To disable pausing, specify a pausetime of 0.
`-r receivesize`	Size of the user-to-service message, up to a maximum value of 100,000 bytes.
`-s sendsize`	Size of the service-to-user message, up to a maximum value of 10,000,000 bytes.
`-t transmittime`	Transmit data for *transmittime* (maximum 6,000) seconds.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- Like the operating system's counterpart, **`p4 ping`** can flood the network with traffic.

p4 populate

Synopsis

Branch a set of files as a one-step operation.

Syntax

```
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] fromFile[rev] toFile
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] -b branch [-r] [toFile[rev]]
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] -b branch -s fromFile[rev]
                     [toFile]
p4 [g-opts] populate [-d description] [-f -n -o] [-m max] -S stream [-P parent] [-r]
                     [toFile[rev]]
```

Description

The **p4 populate** command branches a set of files (the source) into another depot location (the target) in a single step. The new files are created immediately, without requiring a **p4 submit** or a client workspace.

The execution of the **p4 populate** command now fires a `change-submit` trigger to allow interested parties to perform some validation before submission. As with change-content triggers, a temporary pending change record is created so that the description can be accessed, but no results are returned to `change-submit` triggers from **p4 opened** because files branched with **p4 populate** are never opened. If no description is given, the command line arguments are used for a description.

If the **p4 populate** command fails after the `change-content` stage succeeds, a `change-fail` trigger is enabled.

The value of the `rpl.checksum.change` configurable will determine the level of verification performed for this command.

Options

`-b` *branch*	Use the view in a user-defined *branch* specification; the source is the left side of the branch view and the target is the right side of the branch view.
`-d` *description*	Provide a description for the automatically-submitted changelist. If no description is provided, the command line arguments are used for a description.
`-f`	Force deleted files to be branched into the target. (By default, deleted files are treated as nonexistent and are skipped.)
`-m` *max*	Limit the action to the first *max* files.
`-n`	Preview the operation without actually doing anything.

`-o`	Display a list of files created by the **`p4 populate`** command
`-P` *`parent`*	When used with `-S` *`stream`*, specify a parent stream other than the stream's actual parent.
`-r`	Reverse direction of integration (from target to source, rather than from source to target)
`-s`	If used with `-b` *`branch`*, treat *`fromFile`* as the source, and use both sides of the user-defined branch view as the target. (Optional *`toFile`* arguments may be given to further restrict the scope of the target file set.) The `-r` option is ignored when `-s` is used.
`-S` *`stream`*	Use a stream's view; the source is the stream itself, and the target is the stream's parent.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

p4 print

Synopsis

Print the contents of a depot file revision.

Syntax

```
p4 [g-opts] print [-a -A -k -q] [-m max] [-o outfile] file[revRange] ...
p4 [g-opts] print -U unloadfile ...
```

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

Any file in the depot can be printed, subject to permission limitations as granted by **p4 protect**. If the file argument does not map through the client view, you must provide it in depot syntax.

By default, the file is written with a header that describes the location of the file in the depot, the revision number of the printed file, and the number of the changelist that the revision was submitted under. To suppress the header, use the `-q` (quiet) option.

By default, RCS keywords are expanded. To suppress keyword expansion, use the `-k` (keyword) option.

By default, the local depot is searched for the specified file. If you specify the `-U` option, the unload depot is searched instead.

Options

`-a`	For each file, print all revisions within a specified revision range, rather than only the highest revision in the range.
`-A`	Attempt to print a file stored in an archive depot.
`-k`	Suppress RCS keyword expansion.
`-m max`	Print only the first *max* files.
`-o outfile`	Redirect output to the specified output file on the local disk, preserving the same file type, attributes, and/or permission bits as the original file in the depot.
`-q`	Suppress the one-line file header normally added by Perforce.

`-U`	Look for the specified file or files in the unload depot. Data about an unloaded client, label, or task stream can be printed.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- Because most terminals are unable to display UTF16 content, the default behavior of the **`p4 print`** command is to return UTF8 content. You can override this behavior by bypassing terminal output entirely and specifying an output file, for example:

  ```
  $ p4 print -q -o outputfile //depot/file
  ```

  If your terminal supports UTF16 output, specify standard output as the output file:

  ```
  $ p4 print -q -o - //depot/ file
  ```

- **`p4 print`**'s file arguments can take a revision range. By default, only the highest revision matched by any particular file is printed (that is, when no range is specified, the implied range is `#1,#head`, and the highest revision is `#head`). To print all files in a specified (or implied) range, use the `-a` option.

- Because **`p4 print`**'s output can be quite large when called with highly non-restrictive file arguments (for instance, **`p4 print //depot/...`** prints the contents of all files in the depot), it may be subject to a `maxresults` limitation as set in **`p4 group`**.

- In many cases, redirecting **`p4 print`**'s output to a file via your OS shell will suffice.

  The `-o` option is intended for users who require the automatic setting of file type and/or permission bits. This is useful for files such as symbolic links (stored as type `symlink`), files of type `apple`, automatically setting the execute bit on UNIX shell scripts stored as type `text+x`, and so on.

Related Commands

To compare the contents of two depot file revisions	**`p4 diff2`**
To compare the contents of an opened file in the client workspace to a depot file revision	**`p4 diff`**

p4 property

Synopsis

Add, delete, or list property values.

Syntax

```
p4 [g-opts] property -a -n name -v value [-s sequence] [-u user | -g group]
p4 [g-opts] property -d -n name [-s sequence] [-u user | -g group]
p4 [g-opts] property -l [-A] [-n name [-s sequence] [-u user | -g group]]
                    [-F filter] [-T taglist] [-m max]
```

Description

The **p4 property** command can be used by administrators to view and update property definitions stored in the Perforce service. The service does not use the property definitions; it provides this capability for other Perforce applications, such as P4V.

The Perforce service offers three ways of storing metadata: counters/keys, attributes, and properties.

If your application requires only the flat storage of simple key/value pairs, and attempts to implement no security model, use the **p4 counters** and **p4 keys** commands.

If your application's metadata is associated with particular files, use **p4 attribute**.

If your application's metadata is not associated with files, and if you have a requirement to restrict its visibility to users, groups, and/or to control the precedence of multiple values using sequence numbers, use **p4 property**.

When specifying multiple property values for the same property, use distinct sequence numbers to specify the precedence order. A value with a higher sequence number is ordered before a value with a lower sequence number. Values with the same sequence number have an undefined ordering relationship.

Options

`-a`	Update a property value, or add a property value if it is not yet present. Requires `admin` access.
`-A`	List properties for all users and groups, as well as the property sequence number of each property value. Requires `admin` access.
`-d`	Delete a property value. Requires `admin` access.
`-F` *`filter`*	Limit the properties displayed to those that match the *`filter`* pattern. Syntax is that used by **p4 fstat**.
`-g` *`group`*	Specify the user group to which this property applies.

`-l`	List one or more property values. Performance is substantially improved when you supply a `-n` *`name`* argument.
`-m` *`max`*	Limit output to the first max matching properties.
`-n` *`name`*	Specify the name of the property.
`-s` *`sequence`*	Specify the sequence number of the property. If not specified, the default value is `1`.
`-T` *`taglist`*	Limit the fields that are displayed to the fields listed in *`taglist`*. Syntax is that used by **`p4 fstat`**.
`-u` *`user`*	Specify the user to whom this property applies.
`-v` *`value`*	Specify the value of the property.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- Perforce administrators can use **`p4 property`** to centrally manage P4V's performance settings and selectively enable/disable features.

  See the *Helix Versioning Engine Administrator Guide: Fundamentals* for details.

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 --convert-p4admin-comments -o | -i
```

Description

For a new Perforce installation, anyone who wants to use Perforce is allowed to connect to the service, 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.

Use **p4 protect** to control Perforce permissions. You can use **p4 protect** to do the following:

- Control which commands and/or files particular users can access
- Grant permissions to groups of users, as defined with **p4 group**
- Grant access to the **p4 protect** command for a particular path, to a user or group
- Grant or deny specific access rights to users by using the **=read**, **=open**, **=write**, and **=branch** rights, without having to re-grant lesser permissions
- Limit access to particular IP addresses, so that only users at the specified IP addresses can run Perforce.

The command provides three syntax variants:

- The first variant allows you to edit the protections table in a text form.
- The second variant writes the protection table to standard output
- The third variant reads the protection table from standard input. You can also use this variant to convert comments in the p4admin-generated protection table. For more information, see "Comments" on page 308.

In the course of normal operation, you'll primarily grant users **list**, **read**, **write**, **owner**, and **super** access levels. The **open** and **review** access levels are used less often.

In general, you typically grant an access level to a user or group, after which, if finer-grained control is required, you can selectively deny one or more specific rights.

For detailed information, see the "Security" chapter of the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Permission levels and access rights are described in the table below:

Permission Level / Right	What the User Can Do
`list`	The user can access all Perforce metadata, but has no access to file contents. The user can run all the commands that describe Perforce objects, such as **`p4 files`**, **`p4 client`**, **`p4 job`**, **`p4 describe`**, **`p4 branch`**, etc. Those commands that list files, such as **`p4 describe`**, will only list those files to which the user has at least `list` access.
`read`	The user can do everything permitted with `list` access, and also run any command that involves reading file data, including **`p4 print`**, **`p4 diff`**, **`p4 sync`**, and so on.
`=read`	If this right is denied, users cannot use **`p4 print`**, **`p4 diff`**, or **`p4 sync`** on files.
`open`	This gives the user permission to do everything she can do with `read` access, and gives her permission to **`p4 add`**, **`p4 edit`**, **`p4 delete`**, and **`p4 integrate`** files. However, the user is not allowed to lock files or submit files to the depot. The `open` access level gives the user permission to change files but not submit them to the depot. Assign this level when you're temporarily freezing a codeline, but don't want to stop your developers from working, or when you employ testers who are allowed to change code for their own use but are not allowed to make permanent changes to the codeline.
`=open`	If this right is denied, users cannot open files with **`p4 add`**, **`p4 edit`**, **`p4 delete`**, or **`p4 integrate`**.
`write`	The user can do all of the above, and can also write files with **`p4 submit`** and lock them with **`p4 lock`**.
`=write`	If this right is denied, users cannot submit open files.
`=branch`	If this right is denied, users cannot use files as a source for **`p4 integrate`**.
`review`	This permission is meant for external programs that access Perforce. It gives the external programs permission to do anything that `list` and `read` can do, and grants permission to run **`p4 review`** and **`p4 counter`**. It does not include `open` or `write` access.
`admin`	Includes all of the above, including administrative commands that override changes to metadata, but do not affect service operation.

Permission Level / Right	What the User Can Do
	These include **p4 branch** **-f**, **p4 change** **-f**, **p4 client** **-f**, **p4 job** **-f**, **p4 jobspec**, **p4 label** **-f**, **p4 obliterate**, **p4 shelve** **-f** **-d**, **p4 typemap**, **p4 unlock** **-f**, and **p4 verify**.
`super`	Includes all of the above, plus access to the superuser commands such as **p4 admin**, **p4 counter**, **p4 triggers**, **p4 protect**, the ability to create users with **p4 user** **-f**, and so on.
`owner`	Use this permission to assign permission to a specific user or group to run **p4 protect** for a particular path. For details, see the section Authorizing access in *Helix Versioning Engine Administrator Guide: Fundamentals*.

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 uses the following five values to define protections:

Column	Description
Access Level or Mode	One of the access levels `list`, `read`, `open`, `write`, `admin`, `super`, `owner`, `review`; or one of the rights `=read`, `=open`, `=write`, and `=branch`, as defined above.
User or Group	Specifies whether this protection applies to a `user` or a `group`.
Group Name or User Name	The name of the user or the name of the group, as defined by **p4 group**. To grant this permission to all users, use the * wildcard.
Host	The IP address of the client host. IPv6 addresses and IPv4 addresses are also supported. You can use the * wildcard to refer to all IP addresses, but only when you are not using CIDR notation. If you use the * wildcard with an IPv6 address, you must enclose the entire IPv6 address in square brackets. For example, `[2001:db8:1:2:*]` is equivalent to `[2001:db8:1:2::]/64`. Best practice is to use CIDR notation, surround IPv6 addresses with brackets, and to avoid the * wildcard. How the system forms host addresses depends on the setting of the `dm.proxy.protects` variable. By default, this variable is set to 1. This means that if the client host uses some intermediary (proxy, broker, replica) to access the server, the `proxy-` prefix is prepended to the client host address to indicate that the connection is not direct. If you specify `proxy-*` for the `Host` field,

Column	Description
	that will affect all connections made via proxies, brokers, and replicas. A value like **`proxy-10.0.0.5`** identifies a client machine with an IP address of **`10.0.0.5`** that is connected to the server through an intermediary. Setting the **`dm.proxy.protects`** variable to 0, removes the **`proxy-`** prefix and allows you to write a single set of protection entries that apply both to directly-connected clients as well as to those that connect via an intermediary. This is more convenient but less secure if it matters that a connection is made using an intermediary. If you use this setting, all intermediaries must be at release 2012.1 or higher.
Depot File Path	The depot file path this permission is granted on, in Perforce depot syntax. The file specification can contain Perforce wildcards. To exclude this mapping from the permission set, use a dash (`-`) as the first character of this value. If a depot is excluded, the user denied access will no longer see the depot in the output of **`p4 depots`**. Nor will the depot show up, for this user, in the default branch, client, and label views.

Exclusionary mappings

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.

Comments

Protection tables can be difficult to interpret and debug. Including comments can make this work much easier.

- You can append comments at the end of a line using the ## symbols:

```
write user *   10.1.1.1   //depot/test/...  ## robinson crusoe
```

- Or you can write a comment line by prefixing the line with the ## symbols:

```
## robinson crusoe
write user *   10.1.1.1   //depot/test/...
```

Warning

Comments you have created using the P4Admin tool are not compatible with comments created using the 2016.1 version of **p4 protect**. You can use the following command to convert a file containing comments created with P4Admin into a file containing **p4 protect** type comments:

```
$ p4 protect --convert-p4admin-comments -o
```

Then save the resulting file.

Once you have converted the comments, you must continue to define and manage protections using **p4 protect** and can no longer use P4Admin to do so because this tool is unable to parse **p4 protect** comments.

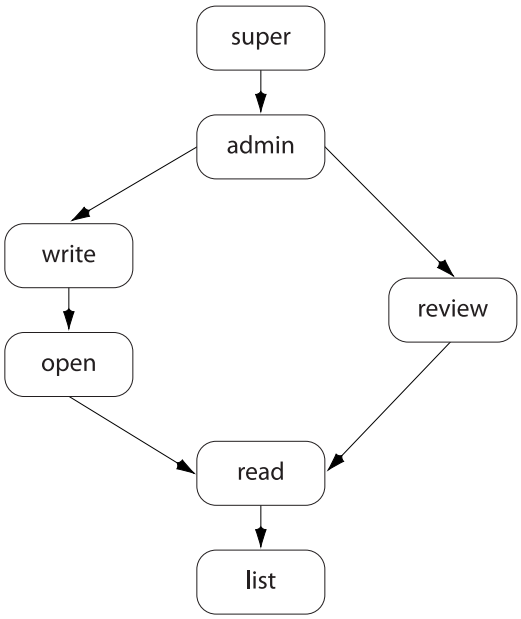
Options

`-i`	Read the form from standard input without invoking an editor.
`-o`	Write the form to standard output without invoking an editor.
`--convert-p4admin-comments`	Converts an existing protections form (and comments) created using P4Admin tool, to a form that can be used by **p4 protect**.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Each permission level includes all the access levels below it, as illustrated in this chart:



The specific rights of `=read`, `=open`, `=write`, and `=branch` can be used to override the automatic inclusion of lower access levels. This makes it possible to deny individual rights without having to then re-grant lesser rights.

For example, if you want administrators to have the ability to run administrative commands, but to deny them the ability to make changes in certain parts of the depot, you could set up a permissions table as follows:

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

In this example, user `joe` can perform administrative functions, and this permission applies to all depots in the system. Because the `admin` permission level also implies the granting of all lower access levels, `joe` can also write, open, read and list files anywhere in the system, including `//depot/build/`. To protect the build area, the `=write` and `=open` exclusionary lines are added to the table. User `joe` is prevented from opening any files for edit in the build area. He is also prevented from submitting any changes in this area he may already have open. He can continue to create and modify files, but only if those files are outside of the protected `//depot/build/...` area.

To limit or eliminate the use of the files on a particular server as a remote depot from another server (as defined by **`p4 depot`**), create protections for user `remote` (or for the service user by which the other server authenticates itself). Remote depots are accessed either by the service user associated with the user's Perforce service, or by a virtual user named `remote`.

Access levels determine which commands you can use.

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.

Some commands (for instance, p4 change, when editing a previously submitted changelist) take a **-f** option that requires admin or super access.

Command	Access Level	Notes
add	open	
admin	super	
annotate	read	
archive	admin	
attribute	write	The **-f** option to set the attributes of submitted files requires **admin** access.
branch	open	The **-f** option to override existing metadata or other users' data requires **admin** access.
branches	list	
change	open	The **-o** option (display a change on standard output) requires only **list** access. The **-f** option to override existing metadata or other users' data requires **admin** access.
changes	list	This command does not operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
client	list	The **-f** option to override existing metadata or other users' data requires **admin** access.
clients	list	
configure	super	
copy	list	**list** access to the source files; **open** access to the destination files.
counter	review	**list** access to at least one file in any depot is required to view an existing counter's value; **review** access is required to change a counter's value or create a new counter.
counters	list	
cstat	list	

Command	Access Level	Notes
dbschema	super	
dbstat	super	
dbverify	super	
delete	open	
depot	super	The **-o** option to this command, which allows the form to be read but not edited, requires only **list** access.
depots	list	This command does not operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
describe	read	The **-s** option to this command, which does not display file content, requires only **list** access.
diff	read	
diff2	read	
dirs	list	
diskspace	super	
edit	open	
export	super	
filelog	list	
files	list	
fix	open	
fixes	list	This command does not 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.
flush	list	
fstat	list	
grep	read	

Command	Access Level	Notes
group	super	The **-o** option to this command, which allows the form to be read but not edited, requires only **list** access. The **-a** option to this command requires only **list** access, provided that the user is also listed as a group owner. The **-A** option requires **admin** access.
groups	list	This command does not operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
have	list	
help	none	
info	none	
integrate	open	The user must have **open** access on the target files and **read** access on the source files.
integrated	list	
interchanges	list	
istat	list	
job	open	The **-o** option to this command, which allows the form to be read but not edited, requires only **list** access. The **-f** option to override existing metadata or other users' data requires **admin** access.
jobs	list	This command does not 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.
journaldbchecksums	super	
key	review	**list** access to at least one file in any depot is required to view an existing key's value; **review** access is required to change a key's value or create a new key.

Command	Access Level	Notes
key	list	admin access is required if the dm.keys.hide configurable is set to 2.
keys	list	admin access is required if the dm.keys.hide configurable is set to 1 or 2.
label	open	This command does not 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 option to override existing metadata or other users' data requires admin access.
labels	list	This command does not 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	open	
license	super	The -u option, which displays license usage, requires only admin access.
list	open	
lock	write	
lockstat	super	
logappend	list	
logger	review	
login	list	
logout	list	
logparse	super	
logrotate	super	
logschema	super	
logstat	super	
logtail	super	
merge	open	

Command	Access Level	Notes
monitor	list	**super** access is required to terminate or clear processes, or to view arguments.
move	open	
obliterate	admin	
opened	list	
passwd	list	
ping	admin	
populate	open	
print	read	
protect	super	
protects	list	**super** access is required to use the **-a**, **-g**, and **-u** options.
property	list	**list** to read, **admin** to add/delete new properties, or show a property setting for all users and groups.
proxy	none	Must be connected to a Perforce Proxy
pull	super	
reconcile	open	
reload	open	**admin** access is required to use **p4 reload** **-f** to reload other users' workspaces and labels.
reopen	open	
replicate	super	
resolve	open	
resolved	open	
restore	admin	
revert	list	
review	review	This command does not operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

Command	Access Level	Notes
reviews	list	This command does not 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.
server	super	
serverid	list	**super** access is required to set the server ID.
set	none	
shelve	open	**admin** access is required to forcibly delete shelved files with **p4 shelve -f -d**
sizes	list	
status	open	
stream	open	
streams	list	
submit	write	
sync	read	
tag	list	
tickets	none	
triggers	super	
typemap	admin	The **-o** option to this command, which allows the form to be read but not edited, requires only **list** access.
unload	open	**admin** access is required to use **p4 unload -f** to unload other users' workspaces and labels.
unlock	open	The **-f** option to override existing metadata or other users' data requires **admin** access.
unshelve	open	
update	list	
user	list	This command does not operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

Command	Access Level	Notes
		The `-f` option (which is used to create or edit users) requires `super` access.
users	`list`	This command does not 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. If the `run.users.authorize` configurable is set to `1`, you must also authenticate yourself to the server before you can run **p4 users**.
verify	`admin`	
where	`list`	This command does not 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.

Examples

Suppose that user `joe` is a member of groups `devgroup` and `buggroup`, as set by **p4 group**, the organization is using only IPv4 connections, and the protections table reads as follows:

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

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

From IP address...	Joe tries...	Results
`10.14.10.1`	**`p4 print //depot/misc/...`**	Succeeds. The second line grants Joe `write` access on these files; `write` access includes `read` access, and this protection is not excluded by any subsequent lines.
`10.14.10.1`	**`p4 print //depot/proj/README`**	Fails. The third line removes all of Joe's permissions on any files in this directory. (If the second protection and the third protection had been switched, then the subsequent protection would have overridden this one, and Joe would have succeeded).

From IP address...	Joe tries...	Results
`192.168.100.123`	**`p4 print //depot/proj/README`**	Succeeds. Joe's workstation is at an IP address from which he is granted this permission in the fourth line.
`192.168.100.123`	**`p4 verify //depot/misc/...`**	Fails. **`p4 verify`** requires **`super`** access; Joe does not have this access level no matter what 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 [-s spec][-a | -u user | -g group] [-h host] [-m] [file ...]
p4 [g-opts] protects -M [-u user | -g group] [file ...]
```

Description

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

- With no options, **p4 protects** displays the lines in the protections table that apply to the current user. If a *file* argument is provided, only those lines in the protection table that apply to the named files are displayed.
- Use the `-a` option to display lines for all users, or `-u` *user*, `-g` *group*, or `-h` *host* options to display lines for a specific user, group, or host IP address.
- Use the `-m` option to display a one-word summary of the maximum applicable access level.
- Use the `-M` option to display a one-word summary of the maximum access level. Unlike the `-m` option, it takes into account exclusions against the specified file path.
- Use the `-s` option to display protection information from a protect table referenced by the file revision specified with the *spec* argument. For example, the following command returns information about the user sam in the third revision of the protections table:

  ```
  $ p4 -u super protects -s //spec/protect.p4s#3 -u sam
  write user* * //...
  ```

  This is useful when users lose access privileges at a given point in time and you want to check what changes were made to the protection table just before that date.

  To use this option, you must define a spec depot for protect forms; this automatically saves revisions to the protect specification every time you edit the protection table. See **p4 depot** for information on how to create a spec depot.

Options

`-a`	Displays protection lines for all users. This option requires `super` access.
`-g` *group*	Displays protection lines that apply to the named *group*. This option requires `super` access.

`-h` *`host`*	Displays protection lines that apply to the specified *`host`* IP address. This option requires `super` access.
`-m`	Display a one-word summary of the maximum applicable access level. (Note: this does not take into account exclusionary mappings or the specified file path into account.)
`-s` *`spec`*	Display information from the specified earlier version of the protect file. The *`spec`* parameter specifies the path of the file version you're interested in. You must define a spec depot for protect forms in order to automatically save protect revisions every time you edit a form.
`-u` *`user`*	Displays protection lines that apply to the named *`user`*. This option requires `super` access.
`g-opts`	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
No	No	`list`, `super` for `-a`, `-h`, `-g`, `-u`

- If the `dm.protects.allow.admin` configurable is set to `1`, Perforce administrators, in addition to Perforce superusers, can also use **`p4 protects -a`**, `-g`, and `-u`.

Related Commands

To edit the protections table	**`p4 protect`**

p4 proxy

Synopsis

Display Proxy connection information.

Syntax

p4 [*g-opts*] proxy

Description

If connected through a Perforce Proxy, the **p4 proxy** command displays information about the proxy connection.

For complete information on the use and configuration of proxies, see the chapter "Perforce Proxy" in *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

Options

g-opts	See "Global Options" on page 571.

Usage Notes

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

- This command only works when the user is connected to a Perforce Proxy.

Related Commands

To display information about a connection to Perforce	**p4 info**

p4 prune

Synopsis

Removes unmodified files from a stream.

The **p4 prune** command is equivalent to the **p4 obliterate** command, except that it can be done by the stream owner rather than an administrator.

Syntax

```
p4 [g-opts] prune [-y] -S stream
```

Description

The **p4 prune** command permanently removes unmodified files (files with one revision) from a stream that is no longer being actively used. Only the owner of a stream may prune it.

By default, **p4 prune** displays a preview of the results. To execute the operation, issue the command again, this time using the **-y** option.

After a stream has been pruned, files with more than one revision remain in the stream so that their edit history is preserved. Unmodified files are gone as if obliterated by an administrator with the **p4 obliterate** command.

Pruned files remain in client workspaces until the next **p4 sync** command runs, which removes them. If pruned files have been branched to a child stream, new integration records are generated to directly link the branched files in the child stream to the files in the parent stream that they were previously related to indirectly.

Mainline, task, and virtual streams may not be pruned. To remove unmodified files from a task stream, delete or unload the stream using the **p4 stream** or **p4 unload** command. The unmodified files automatically go away when the stream spec does.

The stream owner who executes this command must have write access, as granted by the **p4 protect** command.

Options

`-y`	Execute the command. Without this option, the command previews the results but takes no action.
`-S stream`	The name of the stream you want to prune.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`write` for stream owner

Related Commands

An equivalent for **`p4 obliterate -ahbi`**	**`p4 obliterate`**

p4 pull

Synopsis

Retrieve metadata or versioned files from a Perforce master server to a replicate, or display status information about pending transfers.

In most situations, server replication with **p4 pull** is preferable to **p4 replicate**.

Syntax

```
p4 [g-opts] pull [-J prefix] [-i interval] [-b interval] [-T excluded_tables] [-P serverid]
p4 [g-opts] pull -u [-i interval -b interval --batch number]
p4 [g-opts] pull -l [-s | -j [-J prefix]]
p4 [g-opts] pull -d -f file -r revision
p4 [g-opts] pull -L [-i interval]
```

Description

The **p4 pull** command provides five syntax variants:

- The first variant retrieves journal records from a target server specified by P4TARGET.
- The second variant retrieves file contents from a target server specified by P4TARGET.
- The third variant displays information about scheduled file transfers.
- The fourth variant cancels a scheduled file transfer.
- The fifth variant specifies that journal records be retrieved from a local journal file (produced by the **p4 journalcopy** command) rather than from the journal file of the target server. These records are then written to the replica's database. You need to use this variant if you are using a standby replica for failover.

Except for testing purposes, **p4 pull** is rarely run from the command line. Instead, set the **startup.*n*** configurable to start the **p4 pull** processes every time the replica server starts.

When you stop either the master server or a replica server, the replica server tracks the most recent journal position in a small text file called the state file. By default, the state file is named **state** and resides in the replica server's root directory. You can specify a different file name by setting the **statefile** configurable with **p4 configure**.

Retrieving journal and file content

The **p4 pull** command instructs the current replica server to retrieve either journal records or file contents from a target server specified by P4TARGET. Some replica servers do not need both journal records and file contents: for example, if you are creating a replica to help with offline checkpointing, you do not need to transfer file contents.

To replicate both metadata and file contents, you must run at least two **p4 pull** commands: one **p4 pull** (without the **-u** option) to replicate the master server's metadata, and at least one **p4 pull** (with the **-u** option) to replicate the server's versioned files.

- The **-i** option specifies a polling interval (in seconds) between updates. If **-i** is not specified, **p4 pull** runs for one polling interval and then exits.
- The **-b** option specifies a wait time after a failed pull attempt. If **-b** is not specified, **p4 pull** retries after 60 seconds.
- The **-u** option specifies that file content should be retrieved. If this option is not specified only journal records are fetched.
- The **--batch** option specifies the number of files a pull thread should process in a single request. The default value of 1 is usually adequate. For high-latency configurations, a larger value might improve archive transfer speed for large numbers of small files. (Use of this option requires that both master and replica be at version 15.2 or higher.)

Use the **-T** option to exclude tables you do not want to replicate. For example a build farm server does not need to replicate the **db.have**, **db.working**, or **db.resolve** tables.

To delete a pending file transfer operation, use **p4 pull -d -f *file* -r *rev***. This can be useful if a pending file transfer is failing repeatedly due to unrecoverable errors on the master.

Note	Setting the **rpl.compress** configurable allows you to compress journal record data that is transmitted using **p4 pull**.

Getting status information

Use the **-l** option to display a list of files that are scheduled for transfer. If **-s** is specified along with **-l**, a summary of scheduled file transfers is displayed. An additional line specifies the oldest changelist number that has at least one pending transfer. This provides a clue about how far the replica is lagging in its transfer of archive content.

An operator may run the **p4 journalcopy -l**, **p4 pull -l -j**, and **p4 pull -l -s** commands. This makes it possible for an operator to confirm the state of a replica.

```
File transfers: n active/m total, bytes: nnn active/mmmmm total.
Oldest change with at least one pending file transfer: n
```

For example, the following output:

```
File transfers: 1 active/63 total, bytes: 745 active/23684 total.
```

Tells us that there are 63 pending archive file transfers, one of which is currently active; and there are 23,684 bytes needed to be transferred of which 745 are currently actively being transferred.

If **-j** is specified with **-l**, report the current journal state at the current replica and its master, the last time the state file was modified, and the server's local time and time zone. For example:

```
Current replica journal state is: Journal jjj, Sequence: sssss.
Current master journal state is: Journal jjj, Sequence: sssss.
The statefile was last modified at: 2012/01/10 14:23:23.
The Server time is currently: 2012/01/10 14:23:23 -0800 PST
```

The value of *jjj* specifies a journal number; *sssss* specifies an offset in that journal.

Options

`-b` *interval*	Specify a polling interval in seconds for retries after failed retrieval attempts. If you do not specify this option, the pull is retried after 60 seconds.
`--batch` *number*	Use this option to specify the number of files a pull thread should process in a single request. For high-latency configurations, providing a larger value than the default might improve archive transfer speed for large numbers of small files. Default: `1`
`-d -f` *file* `-r` *rev*	Cancel a pending file content transfer, where *file* and *rev* refer to a depot file and a specific revision. **Note** This is not the normal Perforce file and revision data, but rather the archive file and revision. Use the **`p4 pull -l`** command to get the correct file name and revision.
`-i` *interval*	Specify a polling interval in seconds for content retrieval. The smallest interval is one second. If you omit this option, the command runs once and exits. If you set the interval to be `0`, the master server advises the replicate as soon as new data becomes available. This way the replicated server can pull new data with no delay.
`-J` *prefix*	Specify a prefix for the rotated journal file; overrides `journalPrefix` configurable. If your master server uses a non-default rotated journal location, this allows you to specify the rotated journal file location on the master server.
`-l`	List files that are scheduled for transfer. If you use this option on an edge server or build server that has `lbr.replication=cache` set, you might see several entries because of parallel file transfers.
`-l -j`	Display the current journal state on the replica and the master.

	During the process of journal rotation on the master, the output of **p4 pull -l -j** can have three lines of output: one for the replica journal's current state, one for the state of the corresponding journal on the master, and a third line for the new journal on the master, data from which has not yet arrived at the replica.
`-l -s`	Display a summary of scheduled file content transfers. If this list is unexpectedly long or is growing, you might consider running additional **p4 pull -u** commands.
`-L`	Retrieve journal records from a local journal file, normally produced by the **p4 journalcopy** command.
`-P` *serverid*	Filter data from *serverid* according to the `ArchiveDataFilter:` and `ClientDataFilter:` and `RevisionDataFilter:` fields in the specified server's **p4 server** form. In older releases, this option confirmed filters defined in the filter spec. This confirmation is no longer required. The option is retained for continued support of earlier releases. It can also be useful if you want to share filter configuration among multiple servers. In this case, the *serverid* refers to the server whose filter definitions you want shared. **Note** For compatibility with earlier releases of Perforce, you can also supply filter patterns directly within this field by using the same syntax used by the **p4 export**, but specifying a server and using fields in the **p4 server** form is strongly encouraged, because the behavior of a replica that makes use of multiple **p4 pull** commands with inconsistent or conflicting `-P` *filterpattern* arguments is undefined.
`-T` *excluded_tables*	Supply a list of database tables (for example, `db.have` and `db.working`) to exclude from the replica's journal records. The table names must begin with `db.`, following the naming convention used for database files in the server root directory. To specify multiple tables, double-quote the list and separate the table names with spaces. Table names can also be separated by commas. For example, `-T db.have,db.working` or `-T "db.have db.working"`.
`-u`	Transfer archive files instead of journal records. If you omit this option, the command retrieves journal records.
g-opts	See "Global Options" on page 571.

Usage Notes

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

For more about configuring Perforce to run in a replicated environment, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

Related Commands

To configure a Perforce Server to run a set of **`p4 pull`** commands upon startup.	**`p4 configure`**
To replicate metadata from one server to another	**`p4 replicate`**
To display journal or checkpoint records in raw form	**`p4 export`**
To copy journal data to a replica's local file system.	**`p4 journalcopy`**

p4 push

Synopsis

Copy submitted files in your local server to a remote server.

Syntax

```
p4 [g-opts] push [-n -v] [-r remotespec] [-Ox] [-S stream | filespec]

p4 [g-opts] push [-n -v] [-r remotespec] [-Ox] -s shelf
```

Description

The **p4 push** copies the following items from the specified local server to the remote server:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes associated with the changelists, but only if the job that is associated with the fix is already present in the remote server. If it is not, then the fix is not copied.
- all integration records that describe integrations to the files being pushed

A push is only allowed if the files being pushed fit cleanly into the remote server, building precisely on a shared common history. If there are any conflicts or gaps, the push is rejected. Otherwise, the changelists become new submitted changelists in the remote server.

The second form of the command pushes a shelved changelist, rather than one or more submitted changelists, which avoids conflicts; the result is a new shelved change in the remote server.

When the changelists are added to the remote server, they are given newly assigned change numbers but they retain the same description, user, date, type, workspace, and set of files.

When the files are added to the remote server, they are kept in their same changelists, as new revisions starting after the current head. The new revisions retain the same revision number, file type, action, date, timestamp, digest, and file size.

Although the changelists are new submitted changelists in the remote server, none of the submit triggers are run in the remote server.

Note that, once a particular revision of a file has been copied to another server, using **p4 attribute -f** to change the attributes on that revision will only affect the revision on that server, not on any other server to which it may have been copied.

Typically, the **p4 push** command specifies a remote spec, and the `DepotMap` field in the remote spec specifies which files are to be pushed. The **p4 push** command may also specify a filespec argument to further restrict the files to be pushed. If the remote spec uses differing patterns for the local and remote sides of the `DepotMap`, the filespec argument, if provided, must specify the files using the local filename syntax.

If a particular changelist includes some files that match the filespec, and other files that do not, then only the matching files are included in the push. In order to ensure that a partial changelist is not pushed, an appropriate filespec should be specified (e.g., `//...@change,#head`).

The integration records describing integrations to the files being pushed are adjusted in the remote server to reflect the resulting changelist numbers and revision numbers of the remote server.

In order to push a set of files, you must have read access to those files in the local server, and you must have write access to those same files in the remote server (according to the remote server's protections table); your local userid is also used as the userid at the remote server and you must already be logged in to both servers prior to running the **`p4 push`** command.

The **`p4 push`** command is atomic: either all the specified files are pushed, or none of them are pushed.

The value of the `rpl.checksum.change` configurable determines the level of verification performed for the **`p4 push`** command. See "Configurables" on page 595.

Triggering on pushes

The following push trigger types may be invoked during the execution of the **`p4 push`** command:

- The `push-submit` trigger can customize processing during the phase of the **`p4 push`** command when metadata has been transferred but files have not yet been transferred.
- The `push-content` trigger can customize processing during that phase of the **`p4 push`** command when files have been transferred but their contents have not yet been committed.
- The `push-commit` trigger can do any clean up work or other post processing after changes have been committed by the **`p4 push`** command.

For more information, see the section "Triggering on pushes and fetches" in the scripting chapter of *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

If the `-r` option is not specified, **`p4 push`** pushes files to the remote server named origin.

By default, changes cannot be pushed from server to server; in order to push changes between servers, an administrator of each server must enable pushing. Set `server.allowpush` to `1` on the server which initiates the push; set `server.allowpush` to `2` on the destination server. Files with the filetype modifiers `+k`, `+l`, or `+S` have some special considerations. Files of type `+k` have their digests cleared when pushed. This means certain cross-server merge conflicts are not detected. To re-generate the digests in the remote server after the push, use **`p4 verify`**.

When pushing files of type `+l`, the new files are added to the remote server even if the files are currently open by a pending changelist in that server. When pushing files of type `+S`, old archives which exceed the specified limit are not purged by the **`p4 push`** command.

`-n`	Performs all the correctness checks, but does not push any files or changelists to the remote server.

`-Oc`	When set, the **`p4 push`** command outputs information about every changelist. The `-v` option must be set for this to take effect.
`-Of`	When set, the **`p4 push`** command outputs information about every file in every changelist. The `-v` option must be set for this to take effect.
`-Oi`	When set, the **`p4 push`** command outputs information about every integration in every file in every changelist. The `-v` option must be set for this to take effect.
`-r` *`remotespec`*	Specifies a remote spec which contains the address of the remote server, and a file mapping which is to be used to re-map the files when they are pushed to the remote server.
`-s`	Specifies a shelved changelist to be pushed, instead of one or more submitted changelists. For more information, see the section "Fetch and push a shelved changelist" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.
`-S` *`stream`*	Specifies a particular stream to push. If you specify a stream you cannot also specify a file or files.
`-v`	Specifies verbose mode, which provides diagnostics for debugging. You must opt in to verbose mode. With verbose mode turned on, you can refine and control the precise level of verbosity. Specifically, you can indicate whether you want information about: • every changelist pushed (with the `-Oc` option) • every file in every changelist pushed (with the `-Of` option) • every integration of every file in every changelist pushed (with the `-Oi` option) You can specify any combination of these options. The default is to display information about every changelist.
`filespec`	Specifies which file or files to push. If you specify a file or files you cannot specify a stream with the `-S` option.

Usage Notes

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

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
		`write` on the remote server.

Examples

`p4 push -r bruno-remote`	Push a file or files that are specified in the remote spec.

Related Commands

Copy files from a remote server into your local server	**`p4 fetch`**

p4 reconcile

Synopsis

Open files for add, delete, and/or edit in order to reconcile a workspace with changes made outside of Perforce. You might need to use this command after working offline from Perforce.

p4 rec is a synonym for **p4 reconcile**.

Syntax

```
p4 [g-opts] reconcile [-c changelist] [-a -d -e -f -I -k -l -m -n -w] [file ...]
```

Description

If the **p4 reconcile** command finds unopened files in a user's workspace and detects the following three types of inconsistencies between the workspace and the depot, it takes the following actions:

1. Files present in the depot, present in your have list, but missing from your workspace. By default, these files are then opened for `delete`.
2. Files present in your workspace, but missing on the depot. By default, these files are opened for `add`.
3. Files modified in your workspace that are not open for edit. By default, these files are opened for `edit`.
4. Files opened for delete and present in your workspace that don't have pending resolve records are reopened for `edit`.

If the **p4 reconcile** command finds files that are opened for edit but missing from the client, it re-opens them for delete.

If the list of files to be opened includes both adds and deletes, the missing and added files are compared and converted to pairs of `move/delete` and `move/add` operations (as long as the files' sizes and contents are similar.)

To limit the scope of **p4 reconcile** to add, edit, or delete, use the `-a`, `-e`, or `-d` options.

Use the `-k` option to update the have list if files are mapped in a client's workspace to files in the depot that are not on the have list.

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

To improve performance when reconciling changes to large files, the `-m` option can be used under certain conditions.

By default, **p4 reconcile** does not check files and/or paths mentioned in the P4IGNORE file. Use the `-I` option to override this behavior and ignore the P4IGNORE file.

Options

`-a`	Add files: Find files in the workspace that are not under Perforce control and open them for add.
`-c` *`changelist`*	Open the files to be reconciled in the specified pending changelist. If you omit this argument, the files are opened in the default changelist.
`-d`	Delete files: Find files missing from the workspace, but present in the depot; open these files for delete, but only if these files are in the user's have list.
`-e`	Edit files: Find files in the workspace that have been modified outside of Perforce, and open them for edit.
`-f`	Add filenames that contain special (wildcard) characters. Files containing the special characters `@`, `#`, `%`, or `*` are reformatted to encode the characters using hex notation. After these files are added, you must refer to them using their reformatted filenames.
`-I`	Do not perform any ignore checking; ignore any settings specified by `P4IGNORE`.
`-k`	Update the have list if files are mapped in a client's workspace to files in the depot that are not on the have list.
`-l`	Display output in local file syntax with relative paths, similar to the workspace-centric view of **`p4 status`**.
`-m`	Compare the file sync or submit time (in the depot) with the file modification time (in the workspace) to help determine whether the file has changed. Normally Perforce uses file digests to determine whether files in the workspace differ from the head revisions of these files in the depot. This can be time consuming for large files. But when the timestamps are the same when this option is used, the costly digest comparisons can be skipped. This option is only relevant if you are using **`reconcile`** to find changed files rather than files that were deleted or added.
`-n`	Preview the results of the operation without performing any action.
`-w`	Forces the workspace files to be updated to match their corresponding latest synced versions from the depot. Workspace files that are not in the depot are deleted; files that are modified or deleted in the workspace will be replaced with their corresponding versions in the depot. This operation will result in the loss of any changes made to unopened files. The use of **`p4 reconcile`** with this option is the same as using the **`p4 clean`** command. For information on the use of other options when using **`p4 reconcile`** with the `-w` option, see the **`p4 clean`** command. The same options can be used with both.

	This option requires read permission.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- The **p4 reconcile** command produces output in depot syntax. To see file names and paths in local syntax, you must either use the **-l** option with **p4 reconcile**, or use **p4 status**.
- When called without arguments, **p4 reconcile** opens the files in a changelist. To preview an operation, you must either use the **-n** option with **p4 reconcile**, or use the **p4 status** command.

Related Commands

A shortcut for **p4 reconcile -n**	**p4 status**
A shortcut for **p4 reconcile -ead**	**p4 status -A**

p4 reload

Synopsis

Reloads the specified workspace, label, or task stream from the unload depot.

Syntax

```
p4 [g-opts] reload [-f] [-c client | -l label | -s stream] [-p address]
```

Description

The **p4 reload** command reloads the state of an unloaded workspace (or the files tagged by an unloaded label, or stored in an unloaded task stream) from the unload depot into the versioning service's **db.have** (or **db.label**) tables.

Use **-c** *workspace* to reload an unloaded workspace, **-l** *label* to reload an unloaded label, or **-s** *stream* to reload an unloaded task stream. Perforce administrators can use the **-f** option to reload workspaces and/or labels owned by other users.

You can use the **-c** and **-p** options to migrate your unlocked workspace from one edge server to another without unloading the client first. The **p4 reload** command automatically issues the **p4 unload** command and waits for it to complete before reloading your workspace in the new edge server.

Options

-c *client*	Reload the specified client workspace from the unload depot.
-f	Administrator force option; allows reloading of labels and workspaces owned by other users. Requires **admin** access.
-l *label*	Reload the specified label from the unload depot.
-p *address*	In distributed environments, the **-p** option can be used to reload an unloaded client workspace from the remote Edge Server specified by ***address***, thus migrating that workspace from the remote Edge Server to this one. Each Edge Server's service user must be properly authenticated to the other Edge Server in order to perform this operation. The ***address*** parameter can be specified either as the P4PORT or as the server id of the remote server. If you specify a server id, the server spec must contain the correct **P4PORT** value in its **Address** field.
-s *stream*	Reload the specified task stream from the unload depot.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- To reload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough (in the **`p4 group`** form) that users do not need to ask for assistance with **`p4 unload`** or **`p4 reload`** operations.

Related Commands

To unload a client workspace or label name	**`p4 unload`**

p4 remote

Synopsis

Create, modify or delete a remote specification.

Syntax

```
p4 [g-opts] remote [-f] remoteID
p4 [g-opts] remote -d [-f] remoteID
p4 [g-opts] remote -o remoteID
p4 [g-opts] remote -i [-f]
```

Description

A *remote* describes the shared server that your server cooperates with. The **p4 remote** command lets you configure your system such that you can use the fetch and push commands to copy work between your server and the shared server. A remote specification describes the high level configuration and usage of a remote. The **p4 remote** command allows you to create, modify or delete a remote specification.

Note	These remotes have nothing to do with the Perforce construct of remote depots.

The **p4 remote** command puts the remote specification (*spec*) into a temporary file and invokes the editor configured by the `P4EDITOR` environment variable. Saving the file creates or modifies the remote spec.

A remote spec contains the following fields:

- `RemoteID`: The identifier of the remote.
- `Address`: The `P4PORT` that is used by the server.
- `Owner`: The user who created this remote spec. Can be changed.

  The specified owner does not have to be a Perforce user. You might want to use an arbitrary name if the user does not yet exist, or if you have deleted the user and need a placeholder until you can assign the spec to a new user.
- `RemoteUser`: Specifies the identity (user) the Helix Versioning Engine will use to authenticate against this remote server when pushing and fetching.
- `Update`: The date this remote spec was last modified.
- `Access`: The last time this remote was used to fetch or push.
- `Description`: A description of the remote spec (optional).
- `Options`: Flags to change the remote spec behavior. The defaults are marked with *.

  - locked/*unlocked Permits only the owner to change the remote, and prevents the remote spec from being deleted.
  - compress/*nocompress Compresses data sent between the local and remote server to speed up slow connections.
  - *copyrcs/nocopyrcs During a **p4 fetch** or **p4 push**, transfers entire Revision Control System (RCS) archive files when possible, or never transfers entire RCS archive files. By default, **p4 fetch** and **p4 push** copy RCS archive files from and to the shared server as a unit, retrieving multiple revisions with a single file transfer. Sometimes, this optimization is undesirable, because when entire RCS archive files are copies, the archive change numbers are copied as well. This can cause your personal server to experience a "jump" in changelist numbers when changes are fetched from or pushed to the shared server.
- `LastFetch`: The last changelist that was fetched.
- `LastPush`: The last changelist that was pushed.
- `DepotMap`: Mapping between the local and remote files.
- `ArchiveLimits`: One or more entries specifying how many revisions of file archives to store locally when the files are fetched. For more information, see the section "Configure server to limit storage of archive revisions" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.

Options

With no options specified, **p4 remote** invokes your editor for the specified remote spec.

`-d` *`remote`*	Deletes the named remote.
`-f`	Enables a user with *admin* privileges to delete the spec or set the last modified date. By default, specs can be deleted only by their owner.
`-i`	Causes a remote spec to be read from the standard input. The user's editor is not invoked.
`-o` *`remote`*	Writes the remote spec for the named remote to standard output. The user's editor is not invoked.

Usage Notes

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

Examples

p4 remote -i	Read in a remote spec from standard input.

Related Commands

To display a list of remote specifications	**p4 remotes**

p4 remotes

Synopsis

Display a list of remote specifications.

Syntax

```
p4 [g-opts] remotes [[-e|-E] namefilter] [-m count]
```

Description

Use this command to display a list of remote specifications.

Note	These remotes have nothing to do with the Perforce construct of remote depots.

Options

With no options specified **p4 remotes** lists all remote specifications defined on this server.

`-e` *namefilter*	Lists remote specs with a name that matches the *namefilter* pattern. For example: `$ p4 remotes -e svr-dev-rel*` The `-e` option uses the server's normal case-sensitivity rules.
`-E` *namefilter*	Performs the same operation as the `-e` option, but makes the matching case-insensitive even on a case-sensitive server.
`-m` *count*	Limits output to the specified number of remote specs.

Usage Notes

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

Examples

`p4 remotes -m 5`	List up to 5 remote specs.

Related Commands

To create, modify or delete a remote specification	**p4 remote**

p4 rename

Synopsis

Renaming files under Perforce.

Syntax

p4 [*g-opts*] rename [-c *change*] [-f -n -k] [-t *filetype*] *fromFile toFile*

Description

The command **p4 rename** is an alias for **p4 move**.

p4 renameuser

Synopsis

Rename a user and modify all database records that mention the user.

Syntax

```
p4 [g-opts] renameuser [-f] --from=old --to=new
```

Description

The **p4 renameuser** command renames a user and modifies the following elements to reflect this change:

- the user record
- groups that include the user
- properties that apply to the user
- objects owned by the user: workspaces, labels, branches, streams, and so forth
- objects created by the user: all pending, shelved, and committed changes
- files the user has opened or shelved
- fixes the user made to jobs

The user name is not changed in descriptive text fields (such as job descriptions or change descriptions). It is only changed where the name appears as the owner or user field of the database record.

Protection table entries that apply to the user are updated only if the `Name` field exactly matches the user name. If the `Name` field contains wildcards, it is not modified.

The only job field that is processed is attribute code 103. If you have included the user name in other job fields, they will need to be changed manually.

The **p4 renameuser** command does not modify anything in the spec depot.

Important	If you are renaming a user who is being authorized by means of a `P4AUTH` configuration, you must issue the **p4 renameuser** command for every server that the user is authorized to use.

Usage and Limitations

For best results, follow these guidelines:

- Before you use this command, check to see that the user name you want to specify for `new` does not already exist. Using an existing name might result in the merging of data for the existing and the renamed user despite the best efforts of the system to prevent such merges.
- The user issuing this command should not be the user being renamed.
- The user being renamed should not be using the server when this command executes. After the command completes, the user should log out and then log back in.
- The **`p4 renameuser`** command does not process unloaded workspaces: all the user's workspaces should be reloaded (or deleted) first.

  A distributed installation might contain local workspaces or local labels owned by the user; these workspaces and labels, which are bound to Edge Servers, should be deleted or moved to the Commit Server first.

  If a central authentication server has been configured using `P4AUTH`, the user must be renamed in both servers, using separate invocations of **`p4 renameuser`**. The commands may be run in either order.
- Files of type `+k` which contain the `$Author$` tag that were submitted by the user will have incorrect digests following this command. Use **`p4 verify -v`** to recompute the digest value after the rename.

Options

`--from=`*`old`*	The name of the old user.
`--to=`*`new`*	The name of the new user.
`-f`	Forces the command to execute without checking for accidental merge checks that might have happened if the new user had already been used in this server. If the new user name has never been used before, you can improve performance using this option.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To recompute digest values after the rename for certain files.	**`p4 verify`**

To create a user or manage user preferences.	**p4 user**
To list existing users.	**p4 users**

p4 reopen

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 options can be combined to move a file and change its type in the same operation.

Options

`-c changelist`	Move all open files matching file pattern ***file*** to pending changelist ***changelist***. To move a file to the default changelist, use **default** as the changelist number.
`-t filetype`	When submitted, store file as type ***filetype***. All subsequent revisions will be of that file type until the type is changed again. See "File Types" on page 585 for a list of file types.
g-opts	See "Global Options" on page 571.

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 -s pending**
To open a file for edit under a particular pending changelist and as a particular type	**p4 edit -c *changelist* -t *type***
To open a file for add under a particular pending changelist and as a particular type	**p4 add -c *changelist* -t *type***
To implement pessimistic locking (exclusive-open) for all files in a depot. After this changelist is submitted, only one user at a time will be able to edit files in the depot named *depotname*.	**p4 edit -t +l //*depotname*/...**

p4 replicate

Synopsis

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

Syntax

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

Description

This command polls for new journal entries from a Perforce server, and either outputs them to standard output, or, if a *command* is specified, pipe the journal records to the *command*, which is spawned as a subprocess.

Options

`-i interval`	Specify a polling interval, in seconds. The default is two seconds. To disable polling (that is, to check once for updated journal entries and then exit), specify an *interval* of `0`.
`-j token`	Specify a journal number or position token of the form *journalnum/byteoffset* from which to start replicating metadata. If this option is specified, it overrides any state file specification.
`-J prefix`	Specifies a filename prefix for the journal, such as that used with **p4d -jc *prefix***
`-k`	Keep the pipe to the *command* subprocess open between polling intervals.
`-o savefile`	Specify a file for output. If a *command* subprocess is specified, both the subprocess and the specified savefile are provided with the output.
`-R`	The `-R` option causes **p4 replicate** to attempt reconnection to the server in the event of connection loss or serious error. A polling interval must be specified with `-i interval`.
`-s statefile`	Specify a state file which tracks the most recent journal position. You can also use the `statefile` configurable to specify the state file.
`-T tables`	Supply a list of database tables (for example, `db.have`) to exclude from export.
`-x`	Exit the **p4 replicate** command when journal rotation is detected.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- Use **`p4 replicate`** in situations where you need to replicate metadata (but not archived files), or when you need to perform filtering operations on metadata. In most situations, replication with **`p4 pull`** is preferable to **`p4 replicate`**.
- For more information, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment* and the following Perforce Knowledge Base article:

  http://answers.perforce.com/articles/KB_Article/Perforce-Metadata-Replication

Related Commands

To update file content as well as journal records	**`p4 pull`**
To display journal or checkpoint records in raw form	**`p4 export`**

p4 reshelve

Synopsis

Copies shelved files from an existing shelf into either a new shelf or one that has already been created. This command does not require a client workspace

Syntax

```
p4 [g-opts] reshelve -s changelist [file ...]
p4 [g-opts] reshelve [-f] [-p] -s changelist -c changelist [file ...]
```

Description

The **p4 reshelve** command copies shelved files from an existing shelf into either a new shelf or one that has already been created. This command leaves the source shelf intact.

If a file pattern is specified, **p4 reshelve** shelves the files that match the pattern.

Options

`-s` *changelist*	Specify the shelved changelist that contains the shelved files to be copied.
`-c` *changelist*	Specify the pending changelist that will be the target for the shelved files rather than creating a new one. To update a target shelf you must be the owner of the changelist.
`-f`	When the same file already exists in the target changelist, force the overwriting of it.
`-p`	Promote the new or target changelist where it can be accessed by other edge servers participating in the distributed configuration. Once a shelved change has been promoted, all subsequent local modifications to the shelf are also pushed to the commit server and remain until the shelf is deleted.
g-opts	See "Global Options" on page 571.

Usage Notes

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

For more information on promoted shelves see "p4 shelve" on page 399.

Related Commands

To restore shelved files into a workspace	**p4 unshelve**
To shelve files	**p4 shelve**

p4 resolve

Synopsis

Resolve conflicts between file revisions.

Syntax

```
p4 [g-opts] resolve [-aoptions] [-Aoptions] [-doptions] [-f -n -N -o -t -v]
   [-c change] [file ...]
```

Description

Use **p4 resolve** to combine the contents of two files or file revisions into a single file revision in your workspace. Two situations require the use of **p4 resolve** before a file can be submitted:

- When a simple conflict exists: the revision of a file last synced to the client workspace is not the head revision at the time of the submit.

  For example, Alice does a **p4 sync** followed by a **p4 edit** of file `file.c`, and Bob does the same thing. Alice **p4 submit**s `file.c`, and then Bob tries to submit `file.c`. Bob's submit fails because if his version of `file.c` were to be accepted into the depot, Alice's changes to `file.c` would no longer be visible. Bob must resolve the conflict before he can submit the file.

- When **p4 integrate** has been used to schedule the integration of changes from one file (or branch) to another.

The primary difference between these two cases is that resolving a simple file conflict involves multiple revisions of a single file, but resolving for integration involves combining two separate files. In either case:

- If the file is of type `text`, **p4 resolve** allows the user to use the file in the client workspace instead of the file in the depot, overwrite the file in the client workspace with the file in the depot, or merge changes from both the depot revision and the client workspace revision into a single file.

- If the file is of type `binary`, only the first two options (use the file in the workspace, or overwrite the file in the workspace with the file in the depot) are normally available, because merges generally do not work with binary files.

The output of **p4 resolve** is primarily diagnostic in nature; files are either resolved against ("vs") another file, copied, merged, edited, branched, added, deleted, moved, or ignored with respect to other files. The actual work performed by **p4 resolve** is reflected by the changes it makes to files in the client workspace.

Revisions Used to Detect Conflicts

The **p4 resolve** dialog refers to four file revisions whose meaning depends on whether or not the resolution fixes a simple file conflict or is resolving for integration:

Term	Meaning when Resolving Conflicts	Meaning when Resolving for Integration
yours	The revision of the file in the client workspace	The file to which changes are being propagated (in integration terminology, this is the *target* file). Changes are made to the version of this file in the client workspace, and this file is later submitted to the depot.
theirs	The head revision of the file in the depot.	The file revision in the depot from which changes are being propagated (in integration terminology, this is the *source* file). This file is not changed in the depot or the client workspace.
base	The file revision synced to the client workspace before it was opened for edit.	The previously-integrated revision of *theirs*. The latest common ancestor of both *yours* and *theirs*.
merge	A file version generated by Perforce from *yours*, *theirs*, and *base*. The user can edit this revision during the resolve process if the file is a text file.	Same as the meaning at left.

Resolve Options and Details

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

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
`e`	edit merged	Edit the preliminary merge file generated by Perforce.	no
`ey`	edit yours	Edit the revision of the file currently in the workspace.	yes
`et`	edit theirs	Edit the revision in the depot with which the workspace revision conflicts (usually the head revision). This edit is read-only.	yes
`dy`	diff yours	Show diffs between *yours* and *base*.	no
`dt`	diff theirs	Show diffs between *theirs* and *base*.	no
`dm`	diff merge	Show diffs between *merge* and *base*.	no
`d`	diff	Show diffs between *merge* and *yours*.	yes
`m`	merge	Invoke the command:	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
		P4MERGE *base theirs yours merge* To use this option, you must set the environment variable P4MERGE to the name of a third-party program that merges the first three files and writes the fourth as a result. This command has no effect if P4MERGE is not set.	
?	help	Display help for **p4 resolve**.	yes
s	skip	Don't perform the resolve right now.	yes
ay	accept yours	Accept *yours*, ignoring changes that may have been made in *theirs*.	yes
at	accept theirs	Accept *theirs* into the client workspace as the resolved revision. The revision (*yours*) that was in the client workspace is overwritten. When resolving simple conflicts, this option is identical to performing **p4 revert** on the client workspace file. When resolving for integrate, this copies the source file to the target file.	yes
am	accept merge	Accept the *merged* file into the client workspace as the resolved revision without any modification. The revision (*yours*) originally in the client workspace is overwritten.	no
ae	accept edit	If you edited the file (that is, by selecting "e" from the **p4 resolve** dialog), accept the edited version into the client workspace. The revision (*yours*) originally in the client workspace is overwritten.	no
a	accept	Keep Perforce's recommended result: • if *theirs* is identical to *base*, accept *yours*; • if *yours* is identical to *base*, accept *theirs*; • if *yours* and *theirs* are different from *base*, and there are no conflicts between *yours* and *theirs*; accept *merge*;	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
		• otherwise, there are conflicts between *yours* and *theirs*, so skip this file	

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

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

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

The meanings of these values are:

Count	Meaning
n yours	*n* non-conflicting segments of *yours* are different than *base*.
n theirs	*n* non-conflicting segments of *theirs* are different than *base*.
n both	*n* non-conflicting segments appear identically in both *theirs* and *yours*, but are different from *base*.
n conflicting	*n* segments of *theirs* and *yours* are different from *base* and different from each other.

If there are no conflicting chunks, it is often safe to accept Perforce's generated merge file, because 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.

Non-Content-Related Resolves

Beyond differences in content, you can also resolve other types of difference between related files: filetype, deletion, branching, and moves and renames. For details, refer to the *Helix Versioning Engine User Guide*. To constrain the process to one type of resolve, use the **-A** option.

Option	What is Resolved
`-Aa`	Resolve attributes set by **`p4 attribute`**.
`-Ab`	Integrations where the source is edited and the target is deleted.
`-Ac`	Resolve file content changes as well as actions.
`-Ad`	Integrations where the source is deleted and target is edited.
`-Am`	Renames and moves.
`-At`	Filetype changes.
`-AQ`	Charset changes.

Each type of resolve is handled separately. For example, if a file has both a filetype conflict and a content conflict, you are prompted separately to specify how each is handled. To avoid file-by-file prompting when the desired outcome is the same for all resolves, include the `-at` or `-ay` option following the `-A` option. The following example illustrates how prompting is handled for different resolves.

```
Merging //depot/rel/fileb#1
Diff chunks: 1 yours + 0 theirs + 0 both + 0 conflicting
Accept(a) Edit(e) Diff(d) Merge (m) Skip(s) Help(?) ay: m
//depot/main/filez - resolve skipped.
Resolving move to //depot/main/fileb
Filename resolve:
at: //depot/main/fileb
ay: //depot/main/filez
```

Options

`-a`*`options`*
`-am`
`-af`
`-as`
`-at`
`-ay`

Skip the resolution dialog, and resolve the files automatically as follows:

- `-am`: Automatic Mode. Automatically accept the Perforce-recommended file revision: if *theirs* is identical to *base*, accept *yours*; if *yours* is identical to *base*, accept *theirs*; if *yours* and *theirs* are different from *base*, and there are no conflicts between *yours* and *theirs*; accept *merge*; otherwise, there are conflicts between *yours* and *theirs*, so skip this file.
- `-ay`: Accept *Yours*, ignore *theirs*.
- `-at`: Accept *Theirs*. Use this option with caution, as the file in the client workspace will be overwritten!
- `-as`: Safe Accept. If either *yours* or *theirs* is different from *base*, (and the changes are in common) accept that revision. If both are different from *base*, skip this file.

	• **-af**: Force Accept. Accept the *merge* file no matter what. If the *merge* file has conflict markers, they will be left in, and you'll need to remove them by editing the file.
-A*options* **-Aa** **-Ab** **-Ac** **-Ad** **-At** **-Am**	Action (non-content) resolves: Constrain the type of resolve to branching, deletion, file type change, or move/rename. • **-Aa**: Resolve attributes set by **p4 attribute** • **-Ab**: Resolve file branching; that is, integrations where the source is edited and the target is deleted • **-Ac**: Resolve file content changes • **-Ad**: Integrations where the source is deleted and target is deleted • **-At**: Filetype changes • **-Am**: Move and renames For details, see the *Helix Versioning Engine User Guide* and "Non-Content-Related Resolves" on page 362.
-d*option*	When merging files, ignore specified differences in whitespace or line-ending convention. (If you use these options, and the files differ by whitespace only, **p4 resolve** uses the text in the workspace file.) • **-db**: Ignore whitespace-only changes (for instance, a tab replaced by eight spaces) • **-dw**: Ignore whitespace altogether (for instance, deletion of tabs or other whitespace) • **-dl**: Ignore differences in line-ending convention
-f	Allow already resolved, but not yet submitted, files to be resolved again.
-n	List the files that need resolving without actually performing the resolve.
-N	Preview the operation with additional information about any non-content resolve actions that are scheduled.
-o	Output the base file name and revision to be used during the resolve.
-t	Force a three-way merge, even on binary (non-text) files. This allows you to inspect diffs between files of any type, and lets you merge non-text files if **P4MERGE** is set to a utility that can do such a thing.
-v	Include conflict markers in the file for all changes between yours and base, and between theirs and base. Normally, conflict markers are included only when yours and theirs conflict.
-c *change*	Limit the scope of the resolve operation to the files opened in the specified changelist number.

g-opts See "Global Options" on page 571.

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`** or **`p4 merge`**.

    When scheduling files for resolve, **`p4 integrate`** selects the closest common ancestor as the base. The **`p4 merge`** command selects the revision with the most edits in common with the source and target.
- Submitting an open file that was synced from a revision other then the current head revision; the submit fails, and the file is scheduled for resolve.
- Running **`p4 sync`** instead of running **`p4 submit`** on the open file. Nothing is copied into the client workspace; instead, the file is scheduled for resolve. (The only benefit of scheduling files for resolve with **`p4 sync`** instead of a failed submit is that the submit will not fail).

  When **`p4 resolve`** is run with no file arguments, it operates on all files in the client workspace that have been scheduled for resolve.
- If translation errors occur during integrations between `text` and `unicode` files, the most likely cause is the presence of non-ASCII characters in the `text` file. Either remove the non-ASCII characters from the file before integration, or set `P4CHARSET` to `utf8` and attempt the merge again.

Related Commands

To view a list of resolved but unsubmitted files	**`p4 resolved`**
To schedule the propagation of changes between two separate files	**`p4 integrate`**
To submit a set of changed files to the depot	**`p4 submit`**
To copy a file to the client workspace, or schedule an open file for resolve	**`p4 sync`**

p4 resolved

Synopsis

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

Syntax

p4 [*g-opts*] resolved [-o] [*file ...*]

Description

p4 resolved lists files that have been resolved, but have not yet been submitted. The files are displayed one per line in the following format:

```
localFilePath - action from depotFilePath#revisionRange
```

where ***localFilePath*** is the full path name of the resolved file on the local host, ***depotFilePath*** is the path of the depot file relative to the top of the depot, ***revisionRange*** is the revision range that was integrated, and ***action*** is one of `merge`, `branch`, or `delete`.

If file pattern arguments are provided, only resolved, unsubmitted files that match the file patterns are included.

Although the name **p4 resolved** seems to imply that only files that have gone through the **p4 resolve** process are listed, this is not the case. A file is also considered to be resolved if it has been opened by **p4 integrate** for `branch`, opened by **p4 integrate** for `delete`, or has been resolved with **p4 resolve**.

Options

`-o`	Output the base file name and revision that was used during the resolve.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To see a list of integrations that have been submitted	**p4 integrated**

To view a list of integrations that have not yet been resolved	**p4 resolve -n**
To schedule the propagation of changes from one file to another	**p4 integrate**
To resolve file conflicts, or to propagate changes as scheduled by **p4 integrate**	**p4 resolve**

p4 restore

Synopsis

Restore old archived revisions from an archive depot.

Syntax

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

Description

The **p4 restore** command transfers archives from a named *depot* of type **archive** back to their original locations in a local depot. After being restored, the revisions' action is restored to whatever it was before it was archived.

Set the **server.locks.archive** configurable to disable server locks when running the **p4 restore** command.

Options

-D *depot*	Specify an archive depot from which files are to be restored.
-n	Do not restore files; report on revisions that would be restored.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- Storage for the archive depot must be mounted unless you are using the -n option.

Related Commands

To create a depot	**p4 depot**
To archive files into an archive depot	**p4 archive**
To obliterate files without archiving them	**p4 obliterate**

p4 resubmit

Synopsis

Resolve and resubmit some or all unsubmitted changes.

You may not issue this command directly to an edge server; you can issue it directly to a commit server.

Syntax

```
p4 [g-opts] resubmit -l
p4 [g-opts] [-R] resubmit -m
p4 [g-opts] [-R] resubmit -e
p4 [g-opts] [-R] resubmit -i [[-r remote] filespec ...]
```

Description

The **p4 resubmit** command resubmits changes that have been unsubmitted. It is typically used in one of two scenarios:

- to resolve conflicting changes which were detected and unsubmitted by running **p4 fetch -u**.
- to revise a set of changelists that you have submitted locally — but have not pushed to any other server — and have unsubmitted.

Options

`-e`	Runs **p4 resubmit** in partially-interactive mode, allowing you to inspect each change prior to submitting it.
`-i`	Runs **p4 resubmit** as a fully interactive resubmission tool.
`-l`	Lists all the unsubmitted changes but takes no action. This is useful as a way to preview the work that must be resubmitted.

`-m`	Runs in automatic mode. The first thing **p4 resubmit -m** does is to sync your workspace to `#head`. Then, **p4 resubmit -m** processes each unsubmitted change, doing the following for each change:
	
	`$ p4 unshelve -s change -c change`
	`$ p4 sync`
	`$ p4 resolve -am`
	`$ p4 shelve -d -c change`
	`$ p4 submit -c change`
	
	If, for any change in the list, the **p4 resolve -am** processing detects merge conflicts in any file in that change, the **p4 resubmit** command terminates at that point. All the

	files in that change which had merge conflicts are left unresolved. You must then run the **p4 resolve** command to resolve the conflicts. Then re-run **p4 resubmit -Rm** to resume the resubmit process (the first thing it does is submit the resolved files from this change, then it proceeds to the next change).
`-r` *remote*	When **p4 resubmit** is run with the `-i` option, the `-r` option specifies the remote spec whose mapping is used to limit the files affected by the unsubmit operation. For example: `$ p4 resubmit -r rmt @>=17` This example affects only the files specific by the remote spec, not all files in the depot.
`-R`	Resume the resubmit process once conflicts have been resolved. With this flag, resubmit begins by submitting the fully-resolved change and then proceeds to the next unsubmitted change.
filespec	When a filespec is provided with the `-i` option, the interactive resubmit first unsubmits each change that modified a file in that path.

Usage Notes

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

Examples

p4 resubmit -m	Merges and resubmits your unsubmitted changes.

Related Commands

To unsubmit submitted changelists	**p4 unsubmit**

p4 revert

Synopsis

Discard changes made to open files.

Syntax

```
p4 [g-opts] revert [-a -n -k -w] [-c change] [-C client] [--remote=remote] file ...
```

Description

Use **p4 revert** to discard changes made to open files, reverting them to the revisions last synced from the depot (with **p4 sync**). This command also removes the reverted files from the pending changelists with which they're associated. An administrator can use the **-C** option to revert another user's open files.

- When you revert files you opened with **p4 delete**, the files are reinstated in the client workspace.
- When you revert files that have been opened by **p4 add**, 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.
- When you revert files you've opened with **p4 move**, only the file open for **move/add** can be reverted.

The host name is implied and the username is not needed.

Options

-a	Revert only those files that haven't changed (in terms of content or filetype) since they were opened. The only files reverted are those whose client revisions are the following: • Open for edit but have unchanged content and unchanged filetype. • Open for integrate via **p4 integrate** and have not yet been resolved with **p4 resolve**. • Open for add, but are missing from the workspace. Files that are open for add that are missing but which also have pending integrations will not be reverted.
-c *change*	Reverts only those files in the specified changelist. Revert all files in a specific change in a client workspace

```
$ p4 revert -c 345627 "//..."
```

`-C client`

Revert another user's open files. This option implies the `-k`, `-H`, and `-u` options. `-H` and `-u` are global options.

Revert a single file in a client workspace:

```
$ p4 revert -C bruno_ws //depot/www/dev/Jam.html
```

Revert all files in a client workspace:

```
$ p4 revert -C bruno_ws //...
```

This option is useful in freeing up exclusive locks held on an edge server. In such a case, the command needs to be directed to the edge server where the client resides. For example:

```
$ p4 revert -C SusanClient //depot/src/x.cc
```

The option is also very useful in cleaning up after old users' workspaces: you need to revert any open files in a workspace before you delete it.

`-k`

Keep workspace files; the file(s) are removed from any changelists and Perforce records that the files as being no longer open, but the file(s) are unchanged in the client workspace.

`-n`

List the files that would be reverted without actually performing the revert.

This lets you make sure the revert does what you think it does before actually reverting the files.

`--remote=remote`

Reverts the file in your personal server, and additionally — if the file is of type `+l` — releases the global exclusive lock on the file in the shared server from which you cloned the file.

For more information, see the section Support for exclusive locking in the Fetching and Pushing chapter of *Using Helix for Distributed Versioning*.

For more information, see the section "Support for exclusive locking in personal servers" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.

`-w`

Files that are open for `add` are to be deleted (wiped) from the workspace when reverted.

g-opts

See "Global Options" on page 571.

Usage Notes

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

- **`p4 revert`** differs from most Perforce commands in that it usually *requires* a file argument. The files that are reverted are those that lie in the intersection of the command line file arguments and the client view.

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

- Reverting a file that has been opened for **`edit`** will overwrite any changes you have made to the file since the file was opened. It may be prudent to use **`p4 revert -n`** to preview the results before running **`p4 revert`**.

Examples

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

Related Commands

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

p4 review

Synopsis

List all submitted changelists above a provided changelist number.

Syntax

p4 [*g-opts*] review [-c *changelist*] [-t *countername*]

Description

p4 review -c *changelist* provides a list of all submitted changelists between ***changelist*** and the highest-numbered submitted changelist. Each line in the list has this format:

```
Change changelist username <email-addr> (realname)
```

The ***username***, ***email-addr***, and ***realname*** are taken from the **p4 user** form for ***username*** whenever **p4 review** is executed.

When used as **p4 review -t *countername***, all submitted changelists above the value of the Perforce counter variable *countername* are listed. (Counters are set by **p4 counter**). When used with no arguments, **p4 review** lists all submitted changelists.

The **p4 review** command is meant for use in external programs that call Perforce, such as the Perforce change review daemon. The Perforce change review daemon is available from the Perforce Public Depot:

http://wiki.workshop.perforce.com/wiki/P4Review

and is documented in the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

-c *changelist*	List all submitted changelists above and including ***changelist***.
-t *countername*	List all submitted changelists above the value of the Perforce counter ***countername***.
-c *changelist* -t *countername*	Set the value of counter ***countername*** to ***changelist***. This command has been replaced by **p4 counter**, but has been maintained for backwards compatibility.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- The commands **`p4 review`**, **`p4 reviews`**, and **`p4 counter`** are all intended for use by external programs that call Perforce.
- The warnings applicable to **`p4 counter`** apply here as well.

Related Commands

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

p4 reviews

Synopsis

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

Syntax

p4 [*g-opts*] reviews [-C *client*] [-c *change*] [*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 *change* lists each user who has subscribed to review any files included in the submitted changelist ***change***. The alternate form, (**p4 reviews *file* ...**), lists the users who have subscribed to review any files that match the file patterns provided as arguments. If you provide no arguments to **p4 reviews**, all users who have subscribed to review any files are listed.

Options

-C *client*	List all users who have subscribed to review any files opened in the specified workspace ***client***.
-c *change*	List all users who have subscribed to review any files included in submitted changelist ***changelist***.
g-opts	See "Global Options" on page 571.

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 user** 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. Because there's nothing built into Perforce that allows users to subscribe to review jobs, we co-opt a single line of the **`Reviews:`** field: Perforce sends job email to any users who have subscribed to review the non-existent path **`//depot/jobs/`**.

Related Commands

To subscribe to review files	**`p4 user`**
List all submitted changelists above a provided changelist number	**`p4 review`**
To set or read the value of a Perforce counter	**`p4 counter`**
To read full information about a particular changelist	**`p4 describe`**

p4 server

Synopsis

Create, modify, or delete a Perforce server specification.

Syntax

```
p4 [g-opts] server serverID
p4 [g-opts] server -g
p4 [g-opts] server -d serverID
p4 [g-opts] server -o [-l] serverID
p4 [g-opts] server -i [-c edge-server|commit-server]
p4 [g-opts] server -c edge-server|commit-server serverID
```

Description

A server specification describes the high-level configuration and intended usage of a Perforce server. For installations with only one Perforce server, the server specification is optional.

The **p4 server** command puts the server spec into a temporary file and invokes the editor configured by the P4EDITOR variable. Saving the file creates or saves changes to the server specification.

The `ClientDataFilter:`, `RevisionDataFilter:`, and `ArchiveDataFilter:` fields are intended for use in replicated environments where you wish to filter out unnecessary data. (For instance, a build farm replica has little need to replicate the state, including the have list, for every open client workspace on the master server.) For further information about filtering in replicated environments, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

Warning	It is best if `ArchiveDataFilter:` is kept static. You must reseed the server if you change this filter.

An operator type user may not execute this command.

Form Fields

Field Name	Type	Description
`ServerID:`	Read-only	A unique identifier for this server. This must match the contents of the server's `server.id` file as defined by the **p4 serverid** command. If the server type is `identifier`, the server id specifies the name of the cluster
`Type:`	Writable	Server executable type. One of the following: • `server` • `proxy`

Field Name	Type	Description
		• `broker` • `identifier` • `admin` Each type may offer one or more services. See next.
`Services:`	Writable	The `server` type server provides the following services: • `standard` - a standard Perforce server • `replica` - a read-only replica server • `commit-server` - central server in distributed installation • `edge-server` - node in distributed installation • `forwarding-replica` - a replica configured to forward commands that involve database writes to a master server • `build-server` - a replica that supports build automation and build farm integration • `P4AUTH` - a server that provides authentication • `P4CHANGE` - a server that provides change numbering • `depot-master` - commit-server with automated failover • `depot-standby` - standby replica of the depot-master • `workspace-server` - node in a cluster installation • `standby` - read-only replica server that uses p4 journalcopy • `forwarding-standby` - forwarding replica server that uses p4 journalcopy The `proxy` type server provides a **`p4p`** caching proxy. The `broker` type server provides the following services: • `broker` - a **`p4broker`** process • `workspace-router` - routing broker for a cluster

Field Name	Type	Description
		The `services` field for the `identifier` type server specifies the existence of the cluster, and has the value `cluster`. The name of the cluster is then drawn from the `ServerID` field. The `admin` type server provides the following services: • `hxca-server` - the admin server for a Helix cluster. • `zookeeper-server` - ZooKeeper server for a cluster
`Name:`	Writable	The `P4NAME` associated with this server. You can leave this blank or you can set it to the same value as the serverid.
`Address:`	Writable	The `P4PORT` used by this server.
`ExternalAddress:`	Writable	For an edge server, this optional field specifies the external address used for connections to a commit server. This field must be set for the edge server to enable parallel submits in a federated environment.
`Description:`	Writable	An optional description for this server.
`User:`	Writable	The service user name used by the server. For additional information about the use of this field, see the section "Service users" in the chapter "Perforce Replication" in *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

`ClientDataFilter:`	Writable	For a replica server, this optional field can contain one or more patterns describing how active client workspace metadata is to be filtered. Active client workspace data includes have lists, working records, and pending resolves.
		
		To include client data, use the syntax:
		
		`//client-pattern/...`
		
		To exclude client data, use the syntax:
		
		`-//client-pattern/...`
		
		All patterns are specified in client syntax.

`RevisionDataFilter:`	Writable	For a replica server, this optional field can contain one or more patterns describing how submitted revision metadata is to be filtered. Submitted revision data includes revision records, integration records, label contents, and the files listed in submitted changelists.

Field Name	Type	Description

		To include depot data, use the syntax:
		
		`//depot/pattern/...`
		
		To exclude depot data, use the syntax:
		
		`-//depot/pattern/...`
		
		All patterns are specified in depot syntax.

`ArchiveDataFilter:`	Writable	For a replica server, this optional field can contain one or more patterns describing the policy for automatically scheduling the replication of file content. If this field is present, only those files described by the pattern are automatically transferred to the replica; other files are not transferred until they are referenced by a replica command that needs the file content.
		
		Files specified in the `ArchiveDataFilter:` field are transferred to the replica regardless of whether any users of the replica have made requests for their content.
		
		To automatically transfer files on submit, use the syntax:
		
		`//depot/pattern/...`
		
		To exclude files from automatic transfer, use the syntax:
		
		`-//depot/pattern/...`
		
		All patterns are specified in depot syntax.
		
		Warning It is best if this filter is kept static. You must reseed the server if you change this filter.

`DistributedConfig:`	Writable	For an edge or commit server, this optional field, which is displayed only when you use the `-l` or `-c` option, shows configuration settings for this server. • `-l` flag shows the current configuration. • `-c` flag shows current configuration values, recommended default values for fields that are not set, or `unset` for fields that are not set and do not have default values. If this field is present when invoked with `-c`, the configuration commands in this field are run on the

Field Name	Type	Description
		current server using the scope of the server specified in the `serverID` field.

Options

`-c` *`edge-server`* | *`commit-server`*	Set or change configuration values used to set up the distributed environment on an edge or commit server. The specified service dictates which configuration values can be set. Configuration fields are initially populated with the configured values if set, default values if unset, or `unset` for unset values with no default. After exiting from the form, any configuration commands in this field will be run on the current server for the scope of the `serverID`. Because the commands apply only to the ServerID server, the `server#` prefix is not allowed. For more information, see the section "Shortcuts to configuring the server" in *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.
`-d` *`serverID`*	Delete the named server specification.
`-g`	Generate a new serverID as part of the form.
`-i`	Read a server specification from standard input. You can combine this option with the `-c` option to generate and run configuration variables used to set up an edge or commit server. When used with `-c`, only the fields explicitly set in standard input from the `DistributedConfig` field will be configured.
`-l`	Use with `-o` flag to display the values of the configuration variables used to set up the current edge or commit server in a distributed environment. This option shows the configuration commands in the `DistributedConfig` field.
`-o`	Write the named server specification to standard output.
`g-opts`	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	see discussion below

Only super can run **`p4 server`** in update mode (using `-i`, `-g`, and `-d` options). Non-operators can run **`p4 server`** in non-update mode (using `-o` or `-o -g` options). Operators cannot run **`p4 server`** at all.

Related Commands

To change a server's ID after creation	**p4 serverid**
To list all known servers	**p4 servers**

p4 serverid

Synopsis

Get or set the unique ID associated with a Perforce server.

Syntax

p4 [*g-opts*] serverid [*serverID*]

Description

p4 serverid retrieves or sets the unique ID of a Perforce server by reading or writing the **server.id** file in the server's root directory.

Unless a P4NAME value has been specified for the server, the server uses the serverid to determine the appropriate configuration settings. See **p4 configure**.

The recommended technique for configuring servers in a multi-server installation is to give each server its own serverid, and specify the server configuration for that serverid; specifying a separates P4NAME for the server is generally not necessary,

Use this command to create or update the **server.id** file after first generating a unique ID for the server with the **p4 server** command.

The **server.id** file exists in the server's root directory, and must be backed up. If you are using the **p4 server** command to configure your servers, and one of your servers suffers a catastrophic data loss, the restored server will require that this file be present (or be re-created) in order to correctly configure itself upon restart.

Options

serverID	If supplied, update **server.id** with the unique ID of the server.
g-opts	See "Global Options" on page 571.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	**list**, or **super** to set the server ID

Related Commands

To edit or view a server specification	**p4 server**

To list all known servers	**p4 servers**

p4 servers

Synopsis

Display list of all server specifications or evaluate replication status.

A user with operator privileges may execute **p4 servers** and **p4 servers -J**.

Syntax

p4 [*g-opts*] servers [-J | --replication-status]

Description

Syntax variants are described in the following subsections.

Listing server specifications

p4 servers lists all server specifications stored at a master Perforce server.

```
depot-master server depot-master myHost:1111 depot-master 'depot-master '
depot-standby_1 server depot-standby_1 10.0.101.55:37046 depot-standby 'depot-standby '
workspace-server_1 server workspace-server_1 10.0.101.55:41261 workspace-server 'workspace-server
 '
workspace-server_2 server workspace-server_2 10.0.101.55:47050 workspace-server 'workspace-server
 '
```

Output lists the server ID, the type, the services provided, and the description supplied when the server was created.

The output of **p4 servers** may be easier to parse if you retrieve it in tagged form:

p4 -ztag servers

```
... ServerID depot-master
... Name depot-master
... Address myHost:1111
... Type server
... Services depot-master
... Description depot-master


... ServerID depot-standby_1
... Name depot-standby_1
... Address myHost2:37046
... Type server
... Services depot-standby
... Description depot-standby


... ServerID workspace-server_1
... Name workspace-server_1
... Address myHost3:41261
... Type server
... Services workspace-server
... Description workspace-server


... ServerID workspace-server_2
... Name workspace-server_2
... Address myHost4:47050
... Type server
... Services workspace-server
... Description workspace-server
```

Evaluating replication status

Using the `-J` or `--replication-status` option allows you to check how efficiently one or more replicas are replicating the master server's records. Given a server `A` and a replica `B`, output for this command gives you two basic pieces of information:

- The size and update time of `A`'s journal.
- For every server, `B`, that has sent a **`p4 pull`** or **`p4 journalcopy`** request, information is given as to when that request was sent and what is the persisted and applied state of `B`'s journal. (In the case of a simple master and replica, the persisted and applied numbers are always the same: `B`'s journal is updated by the **`p4 pull`** command.

This assumes that the command is executed with the master server as the target. A standby server can replicate master server records using two operations:

- It uses the **`p4 journalcopy`** command to copy (*persist*) the master server's journal to the standby's journal.
- It uses the **`p4 pull -L`** command to *apply* the copied journal records to the standby's database and to update its state file.

You can look at the output to evaluate the load on various parts of your distributed system and to see how well your replicas are keeping up with the master. Growing lag times might be a reason for concern.

The untagged output of **`p4 servers -J`** looks like this:

```
depot-master '2014/09/08 13:13:58' depot-master 5/258 5/258 wadL/1 1
depot-standby_1 '2014/09/08 13:14:58' depot-standby 5/258 5/258 WAdl/12 1
workspace-server_1 '2014/09/08 13:14:58' workspace-server 5/258 5/258 WaDl/10 1
workspace-server_2'2014/09/08 13:14:57' workspace-server 5/258 5/258 WaDl/10 1
```

It is easier to interpret this output in tagged form:

```
... ServerID depot-master
... Updated 2014/09/08 13:13:58
... ServerType depot-master
... PersistedJournal 5
... PersistedSequence 258
... AppliedJournal 5
... AppliedSequence 258
... JAFlags wadL/1
... IsAlive 1

... ServerID depot-standby_1
... Updated 2014/09/08 13:14:58
... ServerType depot-standby
... PersistedJournal 5
... PersistedSequence 258
... AppliedJournal 5
... AppliedSequence 258
... JAFlags WAdl/12 1
... IsAlive 1

... ServerID workspace-server_1
... Updated 2014/09/08 13:14:58
... ServerType workspace-server
... PersistedJournal 5
... PersistedSequence 258
... AppliedJournal 5
... AppliedSequence 258
... JAFlags WaDl/10 1
... IsAlive 1

... ServerID workspace-server_2
... Updated 2014/09/08 13:14:57
... ServerType workspace-server
... PersistedJournal 5
... PersistedSequence 258
... AppliedJournal 5
... AppliedSequence 258
... JAFlags WaDl/10 1
... IsAlive 1
```

The meaning of the fields are described in the following table.

`ServerID`	The server ID of the server. The server ID should always match `P4NAME` if both are set. We recommend setting the server ID, but support `P4NAME` for backward compatibility.
`Updated`	The date and time the requesting server last requested journal records from this server (normally the master).
`ServerType`	The server type. One of the following: `standard`, `replica`, `forwarding-replica`, `build-server`, `edge-server`, `commit-server`, `depot-master`, `depot-standby`, `workspace-server`, `standby`, `forwarding-standby`.
`PersistedJournal`	The rotation number of the journal to which records are being persisted.
`PersistedSequence`	The persisted journal position. For master servers, replicas, and workspace servers, the persisted and applied positions are always the same. They differ only for all types of standby servers.
`AppliedJournal`	The rotation number of the applied journal.
`AppliedSequence`	The applied journal position. For master servers, replicas, and workspace servers, the persisted and applied positions are always the same.
`JAFlags`	Set of fields printed in upper-case if set or lower-case if not. The numeric value of the flags is displayed after the alphabetic display. Common field displays with their associated pull or journalcopy commands are as follows: • `WAdl/12`: **`p4 journalcopy -i 0`** • `WaDl/10`: **`p4 pull -i 0`** • `wAdl/4`: **`p4 journalcopy -i 1`** • `waDl/2`: **`p4 pull -i 1`** • `wadL/1`: synthesized record for master status. You can compare the journal positions of each replica with that of this server to see if any replica is falling behind. Symbols are to be interpreted as follows: • `W/8`: wait, long-poll request • `w`: no wait

	• `A/4`: Acknowledging • `a`: non-acknowledging • `D/2`: durable • `d`: non-durable • `L/1`: data about the local journal; that is, the journal of the server that is the target of the **`p4 servers`** command. • `l`: request from a replica (shows progress in copying master's journal).
`IsAlive`	1 if the server is up; 0 if it's down.

Pull or journal-copy requests are recorded in the **`db.jnlack`** table only when made from a replica that has either a server ID or a `P4NAME`. Any replica that makes such a request but does not have a server ID or `P4NAME` is not recorded in the table.

Options

`-J` | `--replication-status`	Provides information about the server's journal and about the replication status of all replicas that replicate from this server.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To edit or view a server specification	**`p4 server`**
To set a server's unique ID	**`p4 serverid`**

p4 set

Synopsis

Set Perforce system variables.

Syntax

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

Description

Both Perforce client applications and the shared versioning service make use of certain system variables. Depending on the operating system and other factors, variable definitions may be stored in the file defined by **P4CONFIG**, in the file defined by **P4ENVIRO**, or in the Windows registry.

- On Linux, values defined with **p4 set** are stored in the **P4ENVIRO** file.
- On Windows, values defined with **p4 set** are stored in the **P4ENVIRO** file if this is set. If it is not set, they are stored in the Windows registry.

  Windows administrators running Perforce as a service can set environment variables used by the service with **p4 set -S *svcname var=value***. These variables are always stored in the Windows registry.

  To change a variable setting that applies to the current user, use **p4 set *var=value***. Administrators can use **p4 set -s *var=value*** to set the variable's default values for all users on the machine.

For more information, see "P4ENVIRO" on page 519.

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. If a **P4CONFIG** file was used to set the variable, its location is displayed.

- On UNIX, this displays the values of the associated environment variables.
- On Windows, this displays either the environment variable (if set), or the value in the registry and whether it was defined with **p4 set** (for the current user) or **p4 set -s** (for the local machine).

Note how the source of the variable definition is shown in this sample output to the **p4 set** command. If no source is given, the value is stored in the variable itself rather than in a file containing the definition.

```
P4CLIENT=symlinks-nix (config)
P4CONFIG=p4config.txt (config
'/home/perforce/p4clients/symlinks-nix/p4config.txt')
P4EDITOR=/usr/bin/vi
P4IGNORE=p4ignore.txt (enviro)
P4PORT=win-jbrown:20151 (config)
P4USER=jbrown (config)
P4_20151_CHARSET=none (enviro)
```

Please refer to your shell manual for information on setting variables.

Options

`-q`	Reduce the output. When listing files, don't display the origin of the setting. The output is suitable for parsing with scripts.
`-s`	Set the value of the registry variable for the local machine. On Windows, without this option, **`p4 set`** sets the variables in the `HKEY_CURRENT_USER` hive; when you use the `-s` option (and have Windows administrative privileges), the variables are set in the `HKEY_LOCAL_MACHINE` hive. The location is reflected in the output of **`p4 set`** on Windows.
`-S` *`svcname`*	Set the value of the registry variables as used by service *`svcname`*. You must have Windows administrator privileges to do this.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- You'll find a listing and discussion of the Perforce variables in the "Environment and Registry Variables" on page 485 section of this manual.
- Changes to registry settings 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, variables have the following precedence:
  - Environment variables with the same names have precedence;
  - Values within `P4CONFIG` files have precedence over both of these;

  - For the Perforce service, configurables set with **p4 configure** override all environment variables, including registry entries set with **p4 set -S**;
  - The "Global Options" on page 571, specified on the command line, have the highest precedence.
- If you're working in a UNIX-like environment on a Windows machine (for example, Cygwin), use environment variables instead of **p4 set**. (In these cases, the 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 and their origins without changing their values.
p4 set -q	On all platforms, display a list of Perforce variables in a format suitable for scripts to parse without changing their values.
p4 set P4MERGE=	On Windows or OS X, unset the value of P4MERGE.
p4 set P4PORT=ssl:tea:1666	On Windows, set a variable telling Perforce applications to connect to a Perforce service at host **tea**, port **1666**, via SSL. The variable is set only for the current local user.
p4 set -s P4PORT=ssl:tea:1666	Set P4PORT as above, but for all users on the system. You must have administrative privileges to do this.
p4 set -S p4svc P4PORT=1666	For the Windows service **p4svc**, instruct **p4s.exe** to listen on port 1666 for incoming connections from Perforce applications. You must have administrative privileges to do this.
p4 set P4EDITOR="C:\File Editor\editor.exe"	On Windows, for the current local user, set the path for the default text editor. The presence of spaces in the path to the editor's executable requires that the path be enclosed in quotation marks.

p4 shelve

Synopsis

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

Syntax

```
p4 [g-opts] shelve [-p] [file ...]
p4 [g-opts] shelve [-a option] [-p] -i [-f | -r]
p4 [g-opts] shelve [-a option] [-p] -r -c change
p4 [g-opts] shelve [-a option] [-p] -c change [-f] [file ...]
p4 [g-opts] shelve -d -c change [-f] [file ...]
```

Description

Shelving is the process of temporarily storing work in progress in the Perforce versioning service without submitting a changelist. Shelving is useful when you need to perform multiple development tasks (such as interruptions from higher-priority work, testing across multiple platforms) on the same set of files, or when you need to share files for code review before committing your work to the depot.

The **p4 shelve** command creates, modifies, or discards shelved files in a pending changelist. Shelved files persist in the depot until they are discarded (by means of **p4 shelve -d**) or replaced by subsequent **p4 shelve** commands. **p4 shelve** displays the working revision for the files being shelved.

In addition to the files being shelved, **p4 shelve** also shelves any open stream specification. For more information on open stream specifications, see **p4 stream**.

After shelving files, you can revert or modify them in your client workspace. You can also restore the shelved versions of those files to your workspace with the **p4 unshelve** command.

While files are shelved, other users can unshelve the shelved files into their own workspaces, or into other client workspaces.

Files that have been shelved can also be accessed with the **p4 diff**, **p4 diff2**, **p4 files**, and **p4 print** commands, using the revision specifier **@=*change***, where ***change*** is the pending changelist number.

If you are working in a distributed environment, use the **-p** option to promote a shelved change from an edge server to a commit server where it can be accessed by other edge servers in the distributed configuration. When an existing shelved change is promoted, it is promoted without modification unless the **-f** or **-r** options are also used to change the shelved file content. For more information about the use of shelves in distributed environments, see "Usage Notes" on page 401 and also *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

If no arguments are specified, **p4 shelve** creates a new changelist, adds files from the user's default changelist, and (after the user completes a form similar to that used by **p4 submit**), shelves the specified files into the depot. If a file pattern is given, **p4 shelve** shelves only the files that match the pattern.

In order to add a file to a pre-existing shelve, the file must first be opened in the shelve's changelist; use **p4 reopen** to move an opened file from one changelist to another.

Note	**p4 obliterate myfile** does not obliterate a shelve of the file (archive or metadata). When you attempt to unshelve a file that has been obliterated, you will get an error. To recover the content of that file, print the file. To get rid of the shelve, delete the shelf.

Options

`-a` *option*	The `submitunchanged` (default) option shelves all files. The `leaveunchanged` option shelves only the changed files; it leaves the unchanged files opened at the numbered pending changelist.
`-c` *change*	Specify the pending changelist in which shelved files are to be created, discarded, or modified. Only the user and client workspace that owns the pending changelist can add or modify its shelved files. (Administrators can use `-f` to discard files.) Any files specified by a file pattern must already be open in the specified changelist; use **p4 reopen** to move an opened file from one changelist to another.
`-d`	Discard the shelved files in the specified changelist so that they are no longer available for **p4 unshelve** operations. These files are moved from the shelved state to pending state in the changelist specified with the `-d` option. Only the user and workspace that owns the pending changelist can discard its shelved files. (Administrators can use `-f` to discard files.)
`-f`	Force the overwriting of any existing shelved files in a pending changelist with the contents of their client workspace copies. Perforce administrators can use this option with `-d` to force the discarding of shelved files in a specified changelist. Using this option will delete shelved files that are the source of pending resolves. If this happens, the resolving user will not be able to merge content from the shelf; the user must either ignore (`-ay`) the missing shelf or revert.
`-i`	Read a changelist description from standard input. Input must be in the same format used by the **p4 shelve** form. (When modifying an existing changelist with shelved files, this option also implies `-c`).
`-p`	Promote a shelved change from an Edge server to a Commit server where it can be accessed by other edge servers participating in the distributed configuration. Once a shelved change has been promoted, all subsequent local modifications to the shelf are also pushed to the commit server and remain until the shelf is deleted. See "Usage Notes" on page 401 for more information.
`-r`	Replace all shelved files in the changelist with the files that are opened in your workspace.

	The `-r` option (used with `-c` or `-i`) enables you to replace all shelved files in that changelist with the files opened in your own workspace at that changelist number. Previously shelved files will be deleted. Only the user and client workspace of the pending changelist can replace its shelved files.
g-opts	See "Global Options" on page 571.

Usage Notes

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

A promoted shelf is a shelf that exists on the Commit server of a distributed configuration. It is there either because it was directly created on the Commit server or because it was promoted with the `-p` option of the **`p4 shelve`** command. Commands that access shelves know how to handle promoted shelves.

To unpromote a shelf, delete the shelf and create a new one.

A shelf can be promoted when it's first created. A normal shelf can be promoted after it is created by running one of the following commands:

```
$ p4 shelve -p -f -c myChange
$ p4 shelve -p -r -c myChange
```

Promoting a shelf gives you a way to move a shelf from one server to another. To do this, you must complete the following steps:

1. Promote the shelf you want to copy on the server from where you want to copy it, say server X.
2. Unshelve the shelf in the server to which you want to copy it, say server Y.
3. Shelve the change on server Y; this opens the files in a change that is owned by server Y. The new shelf is created as a non-promoted shelf; but you can promote it if you like.

To determine whether a shelved change is promoted, you can try to access the shelf on a server other than the server that owns the change, or you can look at the output of the **`p4 -ztag changes`** command.

Observe the following limitations when working with promoted shelves:

- You can't unload an Edge server workspace if you have promoted shelves.
- Use promoted shelves sparingly; shelf promotion and shelf access are time-consuming operations.

Related Commands

To restore shelved files into a workspace	**`p4 unshelve`**

p4 sizes

Synopsis

Display size information for files in the depot.

Syntax

```
p4 [g-opts] sizes [-a -S] [-s|-z] [-b blocksize] [-h | -H] [-m max] file[revRange] ...
p4 [g-opts] sizes -A [-a -s] [-b blocksize] [-m max] archivefile ...
p4 [g-opts] sizes -U unloadfile ...
```

Description

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

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

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

The **-h** or **-H** option displays size in human-readable form, using a scaling factor of 1,024 for **-h** or 1,000 for **-H**. The size displayed will be automatically scaled to bytes, kilobytes, megabytes, gigabytes, or terabytes, as needed. For example, if you specify **-h**, the output of 75,883,921 bytes, would be represented as 72.36 M.

The **-z** option works the same way as **-s**, but excludes space occupied by lazy copies (files that exist by virtue of integration operations). Use **-z** to estimate the space occupied by files on a Perforce installation, and use **-s** to estimate the local diskspace requirement if files were synced to a client workspace.)

Options

-a	Include all revisions within the range, rather than just the highest revision in the range.
-A	Display files in archive depots. See **p4 archive** for details.
-b *blocksize*	Display results in blocks of *blocksize* bytes. Each accumulated file size is rounded up to the nearest *blocksize* bytes.
-m *max*	Limit output to *max* lines of output.

`-h` or `-H`	Display size in human-readable form, using a scaling factor of 1,024 for `-h` or 1,000 for `-H`. The size displayed will be automatically scaled to bytes, kilobytes, megabytes, gigabytes, or terabytes, as needed.
`-s`	Calculate the sum of the file sizes for the specified file argument.
`-S`	Display size information for shelved files only. If you use this option, revision specifications are not permitted.
`-U` *`unloadfile`*	List only file sizes in the unload depot. See **`p4 unload`** for details.
`-z`	When calculating size information, exclude lazy copies.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- The **`p4 sizes`** command is functionally similar to the UNIX **`du`** command.
- If no revision range is specified, the implicit revision range of `#1` through `#head` is assumed.
- File sizes are based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.

Examples

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

p4 status

Synopsis

Previews output of open files for add, delete, and/or edit in order to reconcile a workspace with changes made outside of Perforce.

The **p4 status** command produces output in local syntax. To see file names and paths in depot syntax, use the **-n** option to **p4 reconcile**.

Syntax

```
p4 [g-opts] status [-c change] [-A | [-e -a -d] | [-s]] [-f -I -m] [file ...]
```

Description

When called without arguments, **p4 status** only previews the results of the workspace reconciliation. To limit the scope of **p4 status** to add, edit, or delete, use the **-a**, **-e**, or **-d** options. You must use either **p4 status -A** (or **p4 reconcile**) to actually open the files in a changelist.

The **p4 status** command finds unopened files in a client's workspace and detects the following three types of inconsistencies between your workspace and the depot:

1. Files present in the depot, present in your have list, but missing from your workspace. By default, these files are then opened for **delete**.
2. Files present in your workspace, but missing on the depot. By default, these files are opened for **add**.
3. Files modified in your workspace that are not open for edit. By default, these files are opened for **edit**.

If the list of files to be opened includes both adds and deletes, the missing and added files are compared and converted to pairs of **move/delete** and **move/add** operations (as long as the files' sizes and contents are similar.)

By default, **p4 status** displays opened files as well as files that need to be reconciled. If you use the **-A**, **-e**, **-a**, or **-d** options or client applications earlier than 2015.1, opened files are not displayed.

By default, **p4 status** does not check files and/or paths mentioned in the P4IGNORE file. Use the **-I** option to override this behavior and ignore the P4IGNORE file.

Options

-a	Display files to be opened for add.
-A	Add, edit, *and* delete files. Files in the client workspace not under Perforce control are opened for add. Changed files are opened for edit. Files in the user's have list that have been removed from the workspace are opened for delete.

	`p4 status -A` is equivalent to **`p4 reconcile -ead`**.
`-c` *`change`*	The changelist containing the files whose status is sought.
`-d`	Display files to be opened for delete.
`-e`	Display files to be opened for edit.
`-f`	Display files to be added whose names contain special (wildcard) characters. Files containing the special characters `@`, `#`, `%`, or `*` are reformatted to encode the characters using hex notation. After these files are added, you must refer to them using their reformatted filenames.
`-I`	Do not perform any ignore checking; ignore any settings specified by `P4IGNORE`.
`-m`	Use in conjunction with the `-e` option to minimize costly digest computation on the client by checking file modification times before checking digests to determine if files have been modified outside of Perforce.
`-s`	Generate summarized output for files to open for add. Using this option causes the command to preview files needing to be reconciled, but provides shorter output for files to be opened for add. Files in the current working directory are listed, but subdirectories containing files to be opened for add are listed rather than the individual files. This provides the shorter output. **Note** This requires version 2015.1 of both server and client.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To reconcile a workspace that has been modified outside Perforce — **`p4 reconcile`**

p4 stream

Synopsis

Create, edit or delete a stream specification.

Syntax

```
p4 [g-opts] stream [-P parent] -t type name
p4 [g-opts] stream [-f -d] [-o [-v]] [-P parent] -t type name
p4 [g-opts] stream -i [-f]
p4 [g-opts] stream edit
p4 [g-opts] stream resolve [-a flag] [-n] [-o]
p4 [g-opts] stream revert
```

Description

The **p4 stream** command enables you to maintain Perforce streams, which are hierarchical branches with policies that control the structure and the flow of change. Stream hierarchies are based on the stability of the streams, specified by the type you assign to the stream. *Development* streams are least stable (most subject to change), *mainline* streams are somewhat stable, and *release* streams are highly stable. *Virtual* streams can be used to copy and merge between parent and child streams without storing local data. *Task* streams are lightweight short-lived branches that are useful for bug fixing or new features that only modify a small subset of the branch data.

Stream contents are defined by the paths that you map. By default, a stream has the same structure as its parent (the stream from which it was branched), but you can override the structure, for example to ensure that specified files cannot be submitted or integrated to other streams.

By default, **p4 stream** edits the stream associated with your current workspace. It throws an error if you're not using a stream workspace. For more information, see the section "Stream workspaces" in the "Streams" chapter of the *Helix Versioning Engine User Guide*.

p4 stream edit puts your client's stream spec into the "opened" state, isolating any edits made to fields that affect view generation. While the spec is open, these fields are marked with the comment "#open" to indicate that they are open and isolated to your client. Changes made to these fields affect your client's view only; other clients' views are not affected.

p4 stream resolve resolves changes that have been submitted to the stream spec since you opened it. You may not submit changes to the stream spec until newer changes have been resolved. See the options table for an explanation of the flags you can pass to this command.

p4 stream revert reverts any pending changes made to the open stream spec, returning your client to the latest submitted version of the stream.

Note	If you integrate from a classic branch or other stream depot to a task stream, the files are not copied up to the parent unless they are edited and submitted first.

For a detailed discussion of streams, refer to the *Helix Versioning Engine User Guide*.

Form Fields

Field Name	Type	Description
`Stream:`	Writable, mandatory	Specifies the stream's name (permanent identifier) and its path in the stream depot, in the form `//depotname/streamname`.
`Update:`	Read-only	The date the stream specification was last modified.
`Access:`	Read-only	The date and time that the stream specification was last accessed by any Perforce command.
`Owner:`	Writable, mandatory	The Perforce user or group who owns the stream. The default is the user who created the stream.
`Name:`	Writable	Display name of the stream. Unlike the `Stream:` field, this field can be modified. Defaults to the *`streamname`* portion of the stream path.
`Parent:`	Writable	The parent of this stream. Must be `none` if the stream's `Type:` is `mainline`, otherwise must be set to an existing stream identifier of the form `//depotname/streamname`.
`Type:`	Writable, mandatory	The stream's type determines the expected flow of change. Valid stream types are `mainline`, `virtual`, `development`, and `release`. • `mainline` The mainline stream is the parent of all streams in the stream depot. Every stream depot must have at least one mainline stream. • `virtual` Virtual streams allow merging and copying between parent and child streams without storing local data. Data is passed through to the destination (a non-virtual stream) after applying restrictions on the scope of files defined in the virtual stream's view. Because virtual streams do not have files in their depot namespace, it is impossible to import a virtual stream. • `release` More stable than the mainline. Release streams copy from the parent and merge to the parent. • `development` Less stable than the mainline. Development streams expect to merge from parent streams and copy to the parent. • `task`

Field Name	Type	Description
		Task streams are lightweight short-lived branches that are useful for bug fixing or new features that only modify a small subset of the branch data. Because branched (copied) files are tracked in a set of shadow tables which are later removed, repository metadata is kept to a minimum when using this type of stream. Workspaces associated with task streams see all branched data, but only modified and promoted data is visible to users with access to the stream's namespace. The default is stream type is `development`.
`Description:`	Writable, optional	Description of the stream.
`Options:`	Writable	Settings that configure stream behavior as follows: • `[un]locked` Enable/disable other users' ability to edit or delete the stream. If locked, the stream specification cannot be deleted, and only its owner can modify it. The default is `unlocked`. • `[all|owner]submit` Specifies whether all users or only the owner of the stream can submit changes to the stream. The default is `allsubmit`. If the `Owner:` of a stream marked `ownersubmit` is a group, all users who are members of that group can submit changes to the stream. • `[no]toparent` Specifies whether integrations from the stream to its parent are expected. The default is `toparent`. • `[no]fromparent` Specifies whether integrations to the stream from its parent are expected. The default is `fromparent` for mainline and development streams, and `nofromparent` for release streams. • `mergeany | mergedown` Specifies whether the merge flow is restricted or whether merge is permitted from any other stream. For example, the `mergeany` option would allow a merge from a child to a parent with no warnings. A `virtual` stream must have its flow options set to `notoparent` and `nofromparent`.

Field Name	Type	Description
		Flow options are ignored for `mainline` streams.
`Paths:`	Writable	Paths define how files are incorporated into the stream structure. Specify paths using the following format: ***`path_type view_path`*** `[`***`depot_path`***`]` where ***`path_type`*** is a single keyword, ***`view_path`*** is a file path with no leading slashes, and the optional ***`depot_path`*** is a file path beginning with `//`. The default path is `share ...` Valid path types are: • `share` ***`view_path`*** Specified files can be synced, submitted, and integrated to and from the parent stream. • `isolate` ***`view_path`*** Specified files can be synced and submitted, but cannot be integrated to and from the parent stream. • `import` ***`view_path`*** `[`***`depot_path`***`]` Specified files can be synced, but cannot be submitted or integrated to and from the parent stream. The ***`view_path`*** is mapped as in the parent stream's view, or to an (optional) ***`depot_path`***. The ***`depot_path`*** may include a changelist specifier. That stream's client workspaces will be limited to seeing revisions at that change or lower within that depot path. For example, you can specify a depot path like this: `//depot/import/...@1000`. Revisions from changelists greater than 1000 will be automatically hidden from most commands. The changelist limits in effect for a given stream workspace are displayed in a read-only client workspace specification field called `ChangeView`. • `import+` ***`view_path`*** `[`***`depot_path`***`]` Functions like a standard `import` path, enabling you to map a path from outside the stream depot to your stream, but unlike a standard import path, you can submit changes to the files in an `import+` path. • `exclude` ***`view_path`***

Field Name	Type	Description
		Specified files cannot be synced, submitted or integrated to and from the parent stream. By default, streams inherit their structure from the parent stream (except mainlines, which have no parent). Paths are inherited by child stream views; a child stream's path can downgrade the inherited view, but not upgrade it. (For example, a child stream can downgrade a **shared** path to an **isolated** path, but if the parent stream defines a path as **isolated**, its child cannot restore full access by specifying the path as **shared**.) Note that the ***depot_path*** is relevant only when the ***path_type*** is **import** or **import+**.
Remapped:	Writable, optional	Reassigns the location of workspace files. To specify the source path and its location in the workspace, use the following syntax: ***view_path_1 view_path_2*** where ***view_path_1*** and ***view_path_2*** are Perforce view paths (omit leading slashes and leading or embedded wildcards; terminal wildcards are fine). For example, to ensure that files are synced to the local ProjectX folder, remap as follows: `... projectX/...` Line ordering in the **Remapped:** field is significant: if more than one line remaps the same files, the later line takes precedence. Remappings are inherited by child streams and the workspaces associated with them.

Ignored:	Writable, optional	A list of file or directory names to be ignored in client views. For example:
		
		`/tmp # ignores files named "tmp"`
		`/tmp/... # ignores directories named "tmp"`
		`.tmp # ignores file names ending in .tmp`
		
		Lines in the **Ignored:** field can appear in any order. Ignored files and directories are inherited by child stream client views.

Options

`-d` *`streamname`*	Delete the stream specification. A stream specification cannot be deleted if it is referenced by child streams or stream client workspaces. Deleting a stream does not remove its files; however, changes can no longer be submitted to the stream.
`-f`	Administrators can use the `-f` option to delete or modify locked streams owned by other users.
`-i`	Read the stream specification from standard input.
`-o`	Write the stream specification to standard output. By default, it outputs the stream associated with the current workspace. If you pass the **`stream@change`** argument, the Helix Versioning Engine uses the version of the stream as of the specified changelist.
`-o -v`	Verbose option; includes the automatically-generated client view for this stream.
`-P` *`parent`*	When creating a new stream specification, specify the stream's parent. (This option has no effect on an existing stream specification.)
`-t` *`type`*	When creating a new stream specification, you must specify the stream's type: either `mainline`, `development`, `release`, `task`, or `virtual`.
`-as`	For **`p4 stream resolve`**, performs a "safe" resolve; it skips fields that need merging.
`-am`	For **`p4 stream resolve`**, resolves by merging; skips fields with conflicts.
`-af`	For **`p4 stream resolve`**, forces a concatenation of text fields with conflicts.
`-at`	For **`p4 stream resolve`**, forces acceptance of theirs, overwrites yours.
`-ay`	For **`p4 stream resolve`**, forces acceptance of yours, overwrites theirs.
`-o -v`	For **`p4 stream resolve`**, previews which fields require resolve.
`-n`	For **`p4 stream resolve`**, outputs the base used for the merge.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- As the name implies, task streams are intended to be short-lived; after you have finished using a task stream by promoting your changes to its parent, delete the task stream.

Examples

`p4 stream -t development -P main //projectX/bruno-dev`	Create a development stream for project X by branching the mainline.

Related Commands

List streams	`p4 streams`
Create stream depot	`p4 depot`

p4 streams

Synopsis

Display a list of streams.

Syntax

```
p4 [g-opts] streams [-U] [-F filter] [-T fields] [-m max] [streamPath ...]
```

Description

Lists the streams defined in the currently connected service. To filter the list, for example, to list streams for a particular depot, specify the *streamPath*.

Options

`-F` *filter*	Filter the output according to the contents of specified fields.
`-m` *max*	Maximum number of streams to list.
`-T` *fields*	Limit field output to fields specified in a list of *fields*. Field names may be separated by a space or comma. Intended for scripting. This option forces tagged output.
`-U`	Display task streams unloaded with **p4 unload**.
g-opts	See "Global Options" on page 571.

Usage Notes

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

Examples

List the streams in the jam depot	`p4 streams //jam/...`
List the release streams owned by Bruno	`p4 streams -F "Owner=bruno Type=release"`

Related Commands

Create, edit or delete a stream	**p4 stream**

p4 submit

Synopsis

Commit a pending changelist and the files it contains to the depot.

Syntax

```
p4 [g-opts] submit [-r -s] [-f submitoption] [--noretransfer 0|1]
p4 [g-opts] submit [-r -s] [-f submitoption] file
p4 [g-opts] submit [-r] [-f submitoption] -d description
p4 [g-opts] submit [-r] [-f submitoption] -d description file
p4 [g-opts] submit [-r] [-f submitoption] [--noretransfer 0|1] -c change
p4 [g-opts] submit -e shelvedchange
p4 [g-opts] submit -i [-r -s] [-f submitoption] --parallel=threads=N[,batch=N][,min=N]
```

Description

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

In addition to the files being submitted, any open stream specification is also submitted. To submit only files and not an open stream spec, run **p4 submit -aF**. For more information on open stream specifications, see **p4 stream**.

By default, files are opened within the default changelist, but you can also create new numbered changelists with **p4 change**.

- To submit the default changelist, use **p4 submit**.
- To submit a numbered changelist, use **p4 submit -c *changelist***.

  Using the **-c** option also allows you to change the description information for a numbered changelist.

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

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

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

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

A file's location in the depot is determined by its location in the local filesystem and by the client workspace definition, which is specified in the **p4 client** form. See "Refining Workspace Views" in the *Helix Versioning Engine User Guide* for more information.

Submit processing

p4 submit works atomically: either all the files listed in the changelist are saved in the depot, or none of them are. The atomic nature of **p4 submit** allows files to be grouped in a changelists according to their purpose. For example, a single changelist might contain changes to three files that fix a single bug. **p4 submit** fails if it is interrupted, or if any of the files in the changelist are not found in the current client workspace, are locked in another client workspace (with **p4 lock**), or require resolution and remain unresolved.

A progress indicator is available for **p4 submit** if you request it with **p4 -I submit**.

Before committing a changelist, **p4 submit** briefly locks all files being submitted. If any file cannot be locked or submitted, the files are left open in a numbered pending changelist. By default, the files in a failed submit operation are left locked unless the **submit.unlocklocked** configurable is set. Files are unlocked even if they were manually locked prior to submit if submit fails when **submit.unlocklocked** is set.

If **p4 submit** fails while processing the default changelist, the changelist is assigned the next number in the changelist sequence, and the default changelist is emptied. The changelist that failed submission must be resubmitted by number after the problems are fixed.

If **p4 submit** fails, some or all of the files might have been copied to the server. By default, retrying a failed submit transfers all these files again unless the **submit.noretransfer** configurable is set, in which case the server attempts to detect if the files have already been transferred and does not re-transfer all files when retrying a failed submit. You can use the **--noretransfer** option to override the **submit.noretransfer** configurable and allow the user to choose the preferred re-transfer behavior for the current submit operation.

Parallel submits

You can transfer files in parallel during the submit process. If there are sufficient resources, a submit command might execute more rapidly by transferring multiple files in parallel. For this feature to work, you must have both server and client upgraded to version 2015.1 or newer. Please read this section in its entirety to make sure that you are using this feature appropriately.

To enable parallel submits, set the **net.parallel.max** configurable.

- Specify **threads=*N*** to request that files be sent concurrently using the specified number of independent network connections. The threads grab work in batches. You specify **batch=*N*** to control the number of files in a batch.

  A submit that is too small will not initiate parallel file transfers. Use the **min** option to control the minimum number of files in a parallel submit.

If the `net.parallel.max` configuration variable is not set, or if the requested number of parallel threads exceeds the value set for `net.parallel.max`, the command will execute without using parallel threads or will use the maximum number of allowed threads.

- Parallel submits from an edge server to a commit server use standard pull threads to transfer the files. The administrator must ensure that pull threads can be run on the commit server by doing the following:
  - Makes sure that the service user used by the commit server is logged into the edge server.
  - Make sure the `ExternalAddress` field of the edge server's server spec is set to the address that will be used by the commit server's pull threads to connect to the edge server.

    If the commit and edge servers communicate on a network separate from the network used by clients to communicate with the edge server, the `ExternalAddress` field must specify the network that is used for connections from the commit server. Furthermore, the edge server must listen on the two (or more) networks.
- The `--parallel` option is ignored when the archives are shared, for **`p4 submit -e`**, and when progress indicators are used.

  You can set the `net.parallel.max`, `net.parallel.submit.threads`, `net.parallel.submit.min`, and `net.parallel.submit.batch` configurables to automate parallel processing. In this case, you can use any of the syntax variants for the **`p4 submit`** command (without specifying the `--parallel` option) and processing will be automatically done in parallel.
  - If you do use the `--parallel` option explicitly and you have the `net.parallel.*` configurables set, the configurable values you specify on the command line override the value of the configurables.
  - You can turn off automatic parallel submit by unsetting the `net.parallel.submit.threads` configurable.
  - You can disable the parallel submit configurable settings by specifying `p4 submit --parallel=0`.

Using parallel submits improves performance in cases like the following:

- Significant network latency exists somewhere along the path through which the submitted file content travels from the client to the repository where the file content is stored.

  This includes significant network latency between a Proxy and Server, or between an Edge Server and a Commit Server. When using parallel submit in such a configuration, the inherent TCP delays related to network latency occur concurrently, rather than sequentially when not using parallel submit.
- Significant resources are required during the transfer of the submitted file, and those resources are available.

  For example, if significant CPU cycles are required to compress ctext or binary file content as it is transferred from a client to a server, the compression of the file content can occur on one CPU core per parallel submit thread compressing either a ctext or binary file, so long as there are enough available CPU cores.

In other cases, using parallel submit might not result in significant performance benefits:

- In some environments, network bandwidth can be a precious resource.

  If network latency is minimal, it might not take many parallel submit threads to use the available network bandwidth. Once the available network bandwidth is used, adding parallel submit threads might not improve performance. This is especially true when transferring file content for which only network bandwidth resources are needed, such as when transferring ubinary files.

- Using a small value for the **batch** and **min** arguments specified with the **--parallel** option is only practical in some cases.

  For example, if a small number of large ctext or binary files are submitted using parallel submit, transferring a small number of files per parallel submit thread can result in the best performance, provided that adequate CPU and network bandwidth resources are available. In order for parallel submit to transfer an evenly-distributed number of files over the number of parallel submit threads specified (which defaults to four), the **batch** argument might need to be set to a value lower than its default of eight. (For example, if submitting eight large ctext or binary files using four parallel submit threads, the **batch** argument should be set to two.) And it follows that the value for the **min** argument, which defaults to nine, should be set to less than or equal to the number of large ctext or binary files being submitted.

  On the other hand, using a small value for the **batch** argument can degrade performance when submitting many small files using parallel submit. The overhead of the server frequently querying **db.sendq** for each batch by each parallel submit thread can result in **db.sendq** concurrency issues. This is because as the size of the files submitted using parallel submit decreases, the more frequently the server queries **db.sendq** for the next batch processed by a parallel submit thread.

Form Fields

Field Name	Type	Description
Change:	Read-only	The change number, or **new** if submitting the default changelist.
Client:	Read-only	Name of current client workspace.
User:	Read-only	Name of current Perforce user.
Status:	Read-only, value	One of **pending**, **submitted**, or **new**. Not editable by the user. The status is **new** when the changelist is created; **pending** when it has been created but has not yet been submitted to the depot with **p4 submit**, and **submitted** when its contents have been stored in the depot with **p4 submit**.
Description:	Writable	Textual description of changelist. This value *must* be changed.
Jobs:	List	A list of jobs that are fixed by this changelist. This field does not appear if there are no relevant jobs. Any job that meets the jobview criteria as specified on the **p4 user** form are listed here by default, but can be deleted from this list.

Field Name	Type	Description
`Type:`	Writable, value	Type of change: `restricted` or `public`. A restricted shelved or committed changelist denies access to users who do not own the changelist and who do not have list permission to at least one file in the changelist. A restricted pending (unshelved) changelist denies access to non-owners of the changelist. Public changes are displayed without these restrictions.
`Files:`	List	A list of files being submitted in this changelist. Files can be deleted from this list, but cannot be changed or added.

Options

`-c` *`change`*	Submit changelist number *`change`*. Changelists are assigned numbers either manually by the user with **`p4 change`**, or automatically by Perforce when submission of the default changelist fails.
`-d` *`description`*	Immediately submit the default changelist with the *`description`* supplied on the command line, and bypass the interactive form. This option is useful when scripting, but does not allow for jobs to be added, nor for the default changelist to be modified.
`-e` *`shelvedchange`*	Submit shelved changelist number *`shelvedchange`*. The `-e` option submits a shelved changelist without transferring files or modifying the workspace. The shelved change must be owned by the person submitting the change, but the workspace may be different. Files shelved to a stream target may only be submitted by a stream workspace that is mapped to the target stream. In addition, files shelved to a non-stream target cannot be submitted by a stream workspace. To submit a shelved change, all files in the shelved change must be up to date and resolved. No files may be open in any workspace at the same change number. Your **`p4 client`** form's `SubmitOptions:` settings (`revertunchanged`, etc) are ignored. If the submit is successful, the shelved change and files and are no longer available to be unshelved or submitted. This is the only submit option supported for files with propagating attributes from an edge server in a distributed environment.
`-f` *`submitoption`*	Override the `SubmitOptions:` setting in the **`p4 client`** form. Valid *`submitoption`* values are: • `submitunchanged`

All open files (with or without changes) are submitted to the depot. This is the default behavior of Perforce.

- `submitunchanged+reopen`

  All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist.

- `revertunchanged`

  Only those files with content or type changes are submitted to the depot. Unchanged files are reverted.

- `revertunchanged+reopen`

  Only those files with content or type changes are submitted to the depot and reopened in the default changelist. Unchanged files are reverted and *not* reopened in the default changelist.

  Note

  Since the two `revertunchanged` options do not revert a file back to the state where the have revision was obtained, you must run **`p4 sync -f`** to revert the file back to the state where the have revision was obtained.

- `leaveunchanged`

  Only those files with content or type changes are submitted to the depot. Any unchanged files are moved to the default changelist.

- `leaveunchanged+reopen`

  Only those files with content or type changes are submitted to the depot. Unchanged files are moved to the default changelist, and changed files are reopened in the default changelist. This option is similar to `submitunchanged+reopen`, except that no unchanged files are submitted to the depot.

`-i`	Read a changelist specification from standard input. Input must be in the same format as that used by the **`p4 submit`** form.
`--noretransfer 0|1`	Set to 1 to have the server avoid re-transferring files that have already been archived after a failed submit operation; set to 0 to have the server retransfer all files after a failed submit operation. This setting overrides the setting of the `submit.noretransfer` configurable for the current submit operation.
`--parallel`	Specify options for parallel file transfer. The configuration variable `net.parallel.max` must be set to a value greater than 1 to enable the `--parallel` option. • `threads=`*`n`* sends files concurrently using *n* independent network connections. The specified threads grab work in batches.

	• `batch=n` specifies the number of files in a batch. • `min=n` specifies the minimum number of files in a parallel sync. A sync that is too small will not initiate parallel file transfers. See "Parallel processing" on page 430 for more information.
`-r`	Reopen files for `edit` in the default changelist after submission. Files opened for `add` or `edit` in will remain open after the submit has completed.
`-s`	Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of `closed`. To leave a job unchanged, use the special status of `same`. On new changelists, the fix status is displayed as the special status `ignore`. (If the status is left unchanged, the job is not fixed by the submission of the changelist.) This option works in conjunction with the `-s` option to **`p4 fix`**, and is intended for use in conjunction with defect tracking systems.
`file`	A single parameter that can be a path with '…' as a wildcard. This file pattern parameter can only be used when submitting the default changelist. The files in the default changelist that match the specified pattern are submitted. Files that don't match the file pattern are moved to the next default changelist.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Examples

`p4 submit`	Submit the default changelist. The user's revisions of the files in this changelist are stored in the depot.
`p4 submit -c 41`	Submit changelist 41.
`p4 submit *.txt`	Submit only those files in the default changelist that have a suffix of `.txt`. Move all the other files in the default changelist to the next default changelist.

`p4 submit -d "header files" *.h`	Submit only those files in the default changelist that have a suffix of **`.h`**, with a description of **`header files`**. No changelist form is displayed. Move all the other files in the default changelist to the next default changelist.

Related Commands

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

p4 switch

Synopsis

Create a stream or switch to a different stream, with an option to populate that stream, or to display current streams.

Syntax

```
p4 [g-opts] switch [-c -m -v -P parent] [-Rx] [-r] stream stream
p4 switch [-r -v] [-Rx] stream@change
p4 switch -l -L
p4 switch
```

Description

This command allows you to create, manage, and switch between your streams. Note that **p4 switch** automatically performs a **p4 reconcile** and **p4 sync** as part of its operations and automatically shelves work in progress when switching between streams.

The **stream@change** argument sets the workspace view to match the version of a stream as of the specified changelist, and syncs the files to the versions matching that same changelist.

You cannot switch to a new stream if files are open in a numbered changelist. If files are open in the default changelist, they are shelved and reverted prior to switching to the new stream; they are automatically unshelved when switching back to this stream.

Options

With no options specified, **p4 switch** displays the current stream.

`-c`	Specifies that the new stream be created and populated with a copy of the files that are in the current stream.
`-l`	Lists all known streams.
`-L`	Lists all streams that were switched at the specified change with open files.
`-m`	Specifies that switch -c should create a new stream with no parent. This would be an independent mainline. The new stream is initially empty.
`-P` *parent*	Specifies that **p4 switch -c** should create a new stream with the specified stream as its parent. The new stream is populated with the files from the specified stream, rather than with the files from the current stream.
`-r`	Reopens files in the new mapped location of the specified stream. If you omit this option and you have opened files in the current stream, those files are shelved before switching to the new stream; they are unshelved when you switch back to this stream.

`-Rx`	Controls how files are reconciled when switching between streams: • `Ra` reconcile files not currently under Perforce control. (Files open for add.) • `Re` reconcile files that have been modified. (Files open for edit.) • `Rd` reconcile files that have been removed. (Files marked for delete.) • `Rn` reconcile does not run.
`-v`	Verbose mode.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	`open` to use the `-c` or `-r` options, `list` to use the `-L` option, or `write` for default switching

Examples

`p4 switch -r bugfix17`	Switch to the `bugfix17` stream and open all the files in it.

Related Commands

Merge	**`p4 merge`**
Resolve	**`p4 resolve`**

p4 sync

Synopsis

Update the client workspace to reflect the contents of the depot.

Syntax

```
p4 [g-opts] sync [-f -k -L -n -N -q -r] [-m max] [file[revRange] ...]
p4 [g-opts] sync [-L -n -N -q -s] [-m max] [file[revRange] ...]
p4 [g-opts] sync [-L -n -N -p -q] [-m max] [file[revRange] ...]
                 --parallel=threads=n[,batch=n][,batchsize=n][,min=n][,minsize=n]
```

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:

- It is visible through the client view;
- It is not already opened by **p4 edit**, **p4 delete**, **p4 add**, or **p4 integrate**;
- It does 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 included in the revision range are updated, and the highest revision in the range is used. Files that are no longer in the workspace view are not affected if the file argument includes a revision range. Use **p4 help revisions** to get help about specifying revisions.

The **p4 sync** command automatically resolves files where the previously synced version does not differ from the newer depot version. Any files you have open when you run **p4 sync**, will need to be manually resolved using the p4 resolve command.

Newly synced files are read-only; **p4 edit** and **p4 delete** make the files writable. 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 option does not affect open files, but it *does* override the `noclobber` client option.

`-k`	Keep existing workspace files; update the have list without updating the client workspace. Use **`p4 sync -k`** only when you need to update the have list to match the actual state of the client workspace. **`p4 sync -k`** is an alias for the **`p4 flush`**. Refer to documentation for the **`p4 flush`** for additional details and a description of the relevant use cases. If your administrator has set the **`zerosyncPrefix`** configurable, *all* workspaces with names that begin with the specified prefix assume **`p4 sync -k`**.
`-L`	For scripting purposes, perform the sync on a list of valid file arguments in full depot syntax with a valid revision number. When this flag is used, the arguments are processed together by building an internal table similar to a label. This file list processing is significantly faster than having to call the internal query engine for each individual file argument. However, the file argument syntax is strict and the command will not run if an error is encountered.
`-m` *`max`*	Sync only the first *`max`* files specified.
`-n`	Preview mode: Display the results of the sync without actually performing the sync.
`-N`	Preview mode: Display a summary of the expected network traffic associated with a sync, without performing the sync. This tells you how many files are to be added or updated, which is useful if you're dealing with many large files and/or are bandwidth or diskspace-limited. This option is useful for estimating network impact of a sync before attempting to perform the sync. If you've recently updated your client workspace view, it's useful to know if you have inadvertently included a folder tree that holds several gigabytes of assets *before* attempting to sync your newly-configured workspace.
`-p`	Populate a client workspace, but do not update the have list. Any file that is already synced or opened is bypassed with a warning message. This option is typically used for workspaces used in processes (such as certain build or publication environments) where there is no need to track the state of the workspace after it has first been synced.
`--parallel`	Specify options for parallel file transfer. The configuration variable **`net.parallel.max`** must be set to a value greater than 1 to enable the **`--parallel`** option. • **`threads=`***`n`* sends files concurrently using *`n`* independent network connections. The specified threads grab work in batches. There is no default value; a value must be set. • **`batch=`***`n`* specifies the number of files in a batch. Default value is **`8`**.

	• `batchsize=n` specifies the number of bytes in a batch. Default value is `512K`. • `min=n` specifies the minimum number of files in a parallel sync. A sync that is too small does not initiate parallel file transfers. Default value is `9`. • `minsize=n` specifies the minimum number of bytes in a parallel sync. A sync that is too small will not initiate parallel file transfers. Default value is `576K`. See "Parallel processing" on page 430 for more information.
`-q`	Quiet operation: suppress normal output messages. Messages describing errors or exceptional conditions are not suppressed.
`-r`	Reopen files that are mapped to new locations in the depot, in the new location. By default, open workspace files remain associated with the depot files that they were originally opened as. For example, pending work can be moved to a different stream by running **`p4 client -f -s`** followed by **`p4 sync -r`**.
`-s`	Safe sync: Compare the content in your client workspace against what was last synced. If the file was modified outside of Perforce control, an error message is displayed and the file is not overwritten. If your client workspace specification has both the `allwrite` and `noclobber` options set, this check is performed by default.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

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

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

- If a user has made certain files writable by using OS commands outside of 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.

  You can prevent this behavior (at a minor cost in performance) by using the **-s** "safe sync" option. Even if the **clobber** option is set, **p4 sync -s** will not overwrite files modified outside of Perforce control.

- A progress indicator is available for **p4 sync** if you request it with **p4 -I sync -q**.

- It is not recommended that you issue a **p4 sync** with multiple arguments referencing the same file multiple times, as in this example: **p4 sync depot/project/...@1000 //depot/project/file.txt@1010**. Doing so will result in unpredictable, inconsistent revisions.

Scripting

The **-m** *max* option is useful when combined with the **-n** option for efficient scripting. For example, a command like **p4 sync -n -m 1** does not sync any files, but displays only one line of output if there are any files to be synced, or a message indicating that the workspace is up to date. Without the **-m 1** option, the output could conceivably be thousands of lines long, all of which would be discarded.

The **-L** option is intended for use by scripts or automated reporting processes. File arguments must be in full depot syntax, and have a valid revision number. File specifications that do not meet these requirements are silently ignored. Using this option speeds up file list processing.

Parallel processing

Depending on the number of files being transferred, the **p4 sync** command might take a long time to execute. You can speed up processing by having this command transfer files using multiple threads. You do this by setting the **net.parallel.max** configuration variable to a value greater than one and by using the **--parallel** option to the **p4 sync** command. Parallel processing is most effective with long-haul, high latency networks or with other network configuration that prevents the use of available bandwidth with a single TCP flow. Parallel processing might also be appropriate when working with large compressed binary files, where the client must perform substantial work to decompress the file.

To configure parallel processing, set the **net.parallel.max** configuration variable to a value greater than one and use the **--parallel** option to the **p4 sync** command.

- The **net.parallel.max** configuration variable can be set to any value between 0 and 100. A value of 0 or 1 disables parallel processing. A value greater than 1 enables parallel processing up to the specified level. So if you want to set the **--parallel** option to 12, the **net.parallel.max** variable must be set to at least 12.

- The **--parallel** option allows you to specify how the parallel processing is to be done. You can specify the suboptions in any order.

  Use the **min** and/or **minsize** suboptions to indicate that you don't want parallel processing unless the sync involves sending at least **min** number of files or at least **minsize** number of bytes.

  Use the **batch** and/or **batchsize** suboptions to specify how many files or bytes should be taken at a time. Setting the batch size small should result in the best use of the network at the risk of overloading database resources.

If the `net.parallel.max` configuration variable is not set or if the requested number of parallel threads exceeds the value set for `net.parallel.max`, the command will execute without using parallel threads or will use the maximum number of allowed threads.

You can also control parallel processing behavior using the following configurables: `net.parallel.batch`, `net.parallel.batchsize`, `net.parallel.min`, `net.parallel.minsize`, and `net.parallel.threads`. This sets behavior globally and can be overridden for a given **`p4 sync`** command if that command specifies different parallel processing option values.

Working with streams

If your client workspace is dynamically-generated because your client workspace's `Stream:` field is set to a valid stream, and you have also set the `StreamAtChange:` field to point to a specified changelist number, **`p4 sync`**, when called with no arguments, will sync your workspace to the revisions of files available as of that changelist, using the client workspace that corresponds to the corresponding stream specification at that point in time.

Retrying the command

Over unreliable networks, you can specify the number of retries to attempt and the length of time beyond which the Perforce application assumes that the network has timed out. Set `net.maxwait` in your workspace's P4CONFIG file or on a one-command basis from the command line, and specify the number of retries with `-r` *n*, where *n* is the number of times to attempt reconnection. For example, the command:

```
p4 -r3 -vnet.maxwait=60 sync
```

attempts to sync the user's workspace, making up to three attempts to resume the sync if interrupted. The command fails after the third 60-second timeout.

Because the format of the output of a command that times out and is restarted cannot be guaranteed (for example, if network connectivity is broken in the middle of a line of output), avoid the use of `-r` on any command that reads from standard input.

Examples

`p4 sync`	Copy the latest revision of all files from the depot to the client workspace, as mapped through the client view. If the file is already open in the client workspace, or if the latest revision of the file exists in the client workspace, it is not copied.
`p4 sync file.c#4`	Copy the fourth revision of `file.c` to the client workspace, with the same exceptions as in the example above.
`p4 sync //depot/proj1/...@21`	Copy all the files under the `//depot/proj1` directory from the depot to the client workspace, as mapped through the client view.

	Don't copy the latest revision; use the revision of the file in the depot after changelist 21 was submitted.
p4 sync @*labelname*	If ***labelname*** is a label created with **p4 label**, and populated with **p4 labelsync**, bring the workspace into sync with the files and revision levels specified in ***labelname***. Files listed in ***labelname***, but not in the workspace view, are not copied into the workspace. Files *not* listed in ***labelname*** are deleted from the workspace. (That is, **@*labelname*** is assumed to apply to all revisions up to, and including, the revisions specified in ***labelname***. This includes the nonexistent revision of the unlisted files.)
p4 sync @*labelname*,@*labelname*	Bring the workspace into sync with a label as with **p4 sync @labelname**, but preserve non-labeled files in the workspace. (The revision range **@*labelname*,@*labelname*** applies only to the revisions specified in the label name itself, and excludes the nonexistent revision of the unlisted files.)
p4 sync @2011/06/24	Bring the workspace into sync with the depot as of midnight, June 24, 2011. (That is, include all changes made during June 23.)
p4 sync status%40june1st.txt	Sync a filename containing a Perforce wildcard by using the ASCII expression of the character's hexadecimal value. In this case, the file in the client workspace is **status@june1st.txt**. For details, see "Limitations on characters in filenames and entities" on page 578.
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 -g -n -U] -l *labelname* *file*[*revRange*] ...

Description

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

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

Options

-d	Delete the label tag from the named files.
-g	In distributed environments, use the **-g** option to specify whether the label being applied is local to an edge server, or is globally available from the commit server. To update a global label, the client workspace must also be an unbound (global) workspace.
-l *labelname*	Specify the label to be applied to file revisions
-n	Display what **p4 tag** would do without actually performing the operation.
-U	If tagging files with a new label, set the **unload** option of the newly-created label. This option has no effect when used with an existing label.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- By default, **p4 tag** operates on the head revision of files in the depot. To preserve the state of a client workspace, use **p4 labelsync**, which operates on the revision of files last synced to your workspace.
- With a distributed Perforce service, **p4 tag** works with a label local to the edge server (to which you are sending a request). The **-g** option can be used to apply a global label, but only with an unbound (global) client workspace.

  By default, labels are local to your edge server, and you use the **-g** option to access global labels on the commit server. If your administrator has set **rpl.labels.global** to **1**, labels are global by default, and the meaning of the **-g** option is inverted to allow updating of local labels.

Examples

p4 tag -l rel1 //depot/1.0/...	Tag the head revisions of files in **//depot/1.0/...** with label **rel1**. If the label **rel1** does not exist, create it.
p4 tag -l build //depot/1.0/...@1234	Tag the most recent revisions as of the submission of changelist **1234** of files in **//depot/1.0/...** with label **build**. If the label **build** does not exist, create it.
p4 files @labelname	List the file revisions tagged by ***labelname***.

Related Commands

To create or edit a label	**p4 label**
To list all labels known to the system	**p4 labels**
To tag revisions in your client workspace with a label	**p4 labelsync**

p4 tickets

Synopsis

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

Syntax

```
p4 [g-opts] tickets
```

Description

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

Options

g-opts	See "Global Options" on page 571.

Usage Notes

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

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

Examples

p4 tickets	Display all tickets stored in a user's local ticket file.

Related Commands

To start a login session (to obtain a ticket)	**p4 login**
To end a login session (to delete a ticket)	**p4 logout**

p4 triggers

Synopsis

Create or display a list of scripts to be run conditionally whenever changelists are submitted, forms are updated, when integrating Perforce with external authentication or archive mechanisms, when rotating journals, or when pushing or fetching content to and from a remote depot.

Syntax

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

Description

Perforce *triggers* are user-written scripts or programs 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.

The **`p4 triggers`** command includes three variants:

- With no options specified, the command invokes the default editor to allow the user to specify one or more trigger definitions.
- The `-i` option specifies that the user use standard input to specify one or more trigger definitions.
- The `-o` option displays the trigger definitions currently stored in the trigger table.

A trigger definition contains four fields that specify the name of the trigger, the type of event that should trigger the execution of the script, the location of the script, and other trigger type-dependent information. When the condition specified in a trigger definition is satisfied, the associated script or program is executed.

For detailed information about writing triggers and trigger definitions, see "Scripting Perforce: Triggers and Daemons" in the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

`-i`	Read the trigger table from standard input without invoking the editor.
`-o`	Write the trigger table to standard output without invoking the editor.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

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%"
```

Note the indentation; the **Triggers:** field name is not indented; each row is indented with at least one space or tab character.

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 "Using triggers to customize behavior" in *Helix Versioning Engine Administrator Guide: Fundamentals*.

Related Commands

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

p4 trust

Synopsis

Establish trust of an SSL connection to a Perforce service.

Use the command **`p4 trust -h`** to get help for the server (if you have not yet trusted your server). You must do this because the command is implemented on the client rather than the server. When a command begins with **`p4 help`**, the client forwards it to the server to construct the text of what is displayed on the client machine. In the case of the trust command, the client might not trust the server to send any commands to it, so the help must begin with **`p4 trust`** to get it serviced locally by the client rather than have it forwarded to the server.

Syntax

`p4 [`*`g-opts`*`] trust [-l -y -n -d -f -r] [-i `*`fingerprint`*`]`

Description

Use **`p4 trust`** to manage the P4TRUST file (by default, **`.p4trust`** in your home directory) to establish (or manage) the trust of an SSL connection.

The trust file contains the fingerprints of the keys received for SSL connections. When you first connect to a Perforce service, you are prompted with its fingerprint; if the fingerprint is correct, you can use **`p4 trust`** to add the service's fingerprint to your trust file. If the fingerprint changes (or expires), subsequent attempts to connect to that service will result in warning or error messages.

Your system administrator can help you confirm the accuracy of any fingerprint (or change to a fingerprint) provided to you by a Perforce service.

Only after you have added an SSL-enabled Perforce service to your P4TRUST file can you connect to it by setting P4PORT to **`ssl:`***`hostname:port`*.

Options

`-d`	Delete an existing trusted fingerprint.
`-f`	Force the replacement of a mismatched fingerprint.
`-i` *`fingerprint`*	Install the specified *`fingerprint`*.
`-l`	List all known fingerprints on this client workstation.
`-n`	Automatically refuse any prompts.
`-r`	List, install, or delete a replacement fingerprint. If a replacement fingerprint exists for the connection, and the primary fingerprint does not match (but the replacement fingerprint does), the replacement fingerprint replaces the primary. This option may be combined with the `-l`, `-i`, or `-d` options.

`-y`	Automatically accept any prompts.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

p4 typemap

Synopsis

Modify the file name-to-type mapping table.

Syntax

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

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 Perforce.

By default, Perforce automatically determines if a file is of type **text** or **binary** based on an analysis of the first 65,536 bytes of a file. If the high bit is clear in each of the first 65,536 bytes, Perforce assumes it to be **text**; otherwise, it's **binary**. Files compressed in the **.zip** format (including **.jar** files) are also automatically detected and assigned the type **ubinary**.

Although this default behavior can be overridden by the use of the **-t *filetype*** option, it's easy to overlook this, particularly in cases where files' types were usually (but not always) detected correctly. This situation occasionally appears with PDF files (which sometimes begin with over 65,536 bytes of ASCII comments) and RTF files, which usually contain embedded formatting codes.

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

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

Form Fields

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

Column	Description
filetype	Any valid Perforce file type. For a list of valid file types, see "File Types" on page 585.
pattern	A file pattern in depot syntax. When a user adds a file matching this pattern, its default file type is the file type specified in the table. To exclude files from the typemap, use exclusionary (**-*pattern***) mappings.

Options

`-i`	Reads the typemap table from standard input without invoking the editor.
`-o`	Writes the typemap table to standard output without invoking the editor.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- To specify all files with a given extension at or below a desired subdirectory, use four periods after the directory name, followed by the extension. (for instance, *`//path/....ext`*) The first three periods specify "all files below this level". The fourth period and accompanying file extension are parsed as "ending in these characters".
- File type modifiers can be used in the typemap table. Useful applications include forcing keyword expansion on or off across directory trees, enforcing the preservation of original file modification times (the `+m` file type modifier) in directories of third-party DLLs, or implementing pessimistic locking policies.
- Specify multiple file type modifiers consecutively. For example, `binary+lFS10` refers to a `binary` file with exclusive-open (`l`), stored in full (`F`) rather than compressed, and for which only the most recent ten revisions are stored (`S10`). For more information on syntax, see "File Types" on page 585.
- If you use the `-t` option and file type modifiers to specify a file type on the command line, and the file to which you are referring falls under a **`p4 typemap`** mapping, the file type specified on the command line overrides the file type specified by the typemap table.

Examples

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

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

The first three periods ("`...`") in the specification are a 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 "File Types" on page 585.

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

```
Typemap:
        +l //depot/...
```

Related Commands

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

p4 undo

Synopsis

Undo a range of revisions.

Syntax

p4 [*g-opts*] undo [-n] [-c *change*] file[*revRange*]

Description

The **p4 undo** command opens files in order to undo a set of previously submitted changes. The "undone" changes remain a part of the file history, but the new revisions submitted after **p4 undo** reverse their effect.

If a single revision is specified, the specified revision is undone. If a revision range is specified, the entire range is undone.

The workspace files opened by **p4 undo** are synced to the revision prior to those of the range being undone and opened at the most recent undone revision. Files that are opened at a revision prior to the head must be resolved prior to submission; to accomplish this, run **p4 sync** followed by **p4 resolve**.

Options

`-n`	Preview the operation without changing any files.
`-c change`	Open files in the specified pending changelist rather than the default changelist.
g-opts	See "Global Options" on page 571.

Usage Notes

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

To undo the effects of a particular change, issue this command:

```
$ p4 undo @change
```

To undo all changes made on April 1, 2016, issue this command:

```
$ p4 undo @2016/04/01,@2016/04/02
```

The above examples open files in the default changelist. To use a numbered changelist, specify it with the `-c` option.

p4 unload

Synopsis

Unloads a workspace, label, or task stream to the unload depot or to a flat file.

Syntax

```
p4 [g-opts] unload [-f -L -p -z] [-c client | -l label | -s stream] [-o localFile]
p4 [g-opts] unload [-f -L -z] [-a | -al | -ac] [-d date | -u user]
```

Description

There are two main uses for the **p4 unload** command:

- You can use the command to transfer infrequently-used metadata from the versioning engine's `db.*` files to a set of flat files in the unload depot. Unloading metadata reduces the size of the working set required by the versioning engine, and on large sites with many years of historical metadata, can offer significant performance improvements.

  Perforce commands such as **p4 clients**, **p4 labels**, **p4 files**, **p4 sizes**, and **p4 fstat** ignore unloaded metadata. (To view metadata that has been unloaded, use the `-U` option with these commands). Most users who use Perforce reporting commands do so with the intent of retrieving a superset of the desired data, and then use automated or manual post-processing to discard the irrelevant lines of output. For example, the **p4 clients** command (when called without arguments) returns the name of every client workspace ever created by every current and former employee of your organization, even those who left ten years ago. Unloading obsolete metadata is a good way to offer your users a higher signal-to-noise ratio not only in the output of their command-line queries, but also in the amount of information displayed in applications such as P4V.

- You can use the command with the `-o` option to unload a client, label, or task stream to a flat file on the client rather than to a file in the unload depot. This can be useful for seeding a client into another database or for creating a private backup of the client. The flat file uses standard journal format. The client, label, or task stream remains fully loaded after the command is run.

Use the `-c` and `-l` options to unload a specific client workspace or label. By default, users can only unload their own workspaces or labels; administrators can use the `-f` option to unload workspaces and labels owned by other users.

You do not need to unload a workspace in preparation for moving it from one edge server to another; running the **p4 reload** command automatically unloads the specified workspace before reloading it into a new edge server.

Use the `-a`, `-al`, or `-ac` options to indicate that all specified labels and/or client workspaces are to be unloaded. You cannot use these options if you are also using the `-o` option.

Use the `-d` *date* and/or `-u` *user* to restrict the unloading operation to labels and/or workspaces older than a specific *date*, owned by a specific *user*, or both.

Use the `-L` option to unload locked workspaces and/or labels. By default, only unlocked labels or workspaces are unloaded.

The access date for a workspace is updated whenever the workspace is used by a command that directly references the workspace. Similarly, the access date for a label is updated when the label is used by a command which directly references that label. The access date for a workspace is also updated when the workspace is used in a revision specifier of the form `@workspace`, and the access date for a label is updated when the label is used in a revision specifier of the form `@labelname`.

By default, data in the unload depot is uncompressed. Use `-z` to store it in compressed form; unloaded metadata is often highly compressible, particularly in continuous build environments characterized by millions of build-associated workspaces labels that are used to perform a single build and then rarely, if ever, accessed again.

Options

`-a`	Unload all applicable client workspaces and labels; requires `-d`, `-u`, or both `-d` and `-u` options. This option does not affect task streams.
`-ac`	Unload client workspaces; requires `-d`, `-u`, or both `-d` and `-u` options.
`-al`	Unload labels; requires `-d`, `-u`, or both `-d` and `-u` options.
`-c client`	Unload the specified client workspace's metadata from `db.have` (and related tables) and store it in the unload depot.
`-d date`	Unload metadata older than the specified date.
`-f`	Force option; administrators can unload workspaces, labels, and task streams owned by other users.
`-l label`	Unload the specified label from `db.label` (and related tables) and store it in the unload depot.
`-L`	Unload a `locked` workspace, label, or task stream.
`-o outputFile`	Unload metadata to a file rather than to the unload depot. Note however that an unload depot must exist for this option to work. This is because the data is first placed in the unload depot and then moved to the output file you specify. Normal users can only unload objects in their own clients. An administrator can use this option to unload an object owned by other users.
`-p`	Promote any non-promoted shelves belonging to the specified client that is being unloaded. The shelf is promoted to the commit server where it can be accessed by other edge servers.
`-s stream`	Unload the specified task stream. (the `stream` must be of type `task`)
`-u user`	Unload metadata owned by the specified user.
`-z`	Store the unloaded workspace, label, or task stream in compressed format.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- To unload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough (in the **`p4 group`** form) that users do not need to ask for assistance with **`p4 unload`** or **`p4 reload`** operations.

Related Commands

To reload data from the unload depot.	**`p4 reload`**

p4 unlock

Synopsis

Release the lock on a file.

Syntax

```
p4 [g-opts] unlock [-c change | -s shelvedchange | -x] [-f] [file ...]
p4 [g-opts] -c client unlock [-f ] -r
p4 [g-opts] -g -c change [-f ]
```

Description

The **p4 unlock** command includes three syntax variants:

- The first syntax variant releases locks that were created explicitly using the **p4 lock** command or implicitly during the course of a submit operation or other operations that require file locking.

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

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

- The second syntax variant allows you to unlock files that were left locked due to a failed **p4 push** command.

  If a **p4 push** command from a remote server to this server fails, files can be left locked on this server, preventing other users from submitting changes that affect these files. In this case, the user who issued the **p4 push** command can use the **-r** option of the **p4 push** command (and specify the name of the client that was used on that remote server) to unlock the files on this server. An administrator can run **p4 unlock -f -r** as well. For example:

  ```
  $ p4 -p central -c myworkspace unlock -r
  ```

- In an edge/commit architecture, this syntax variant unlocks the files in the specified changelist locally and globally. This option may only be used from an edge server containing the local change, and it must be used with the **-c** *changelist* option.

By default, files can be unlocked only by the changelist owner who must also be the user who has the files locked. However, administrators may use the **-f** option to forcibly unlock a file opened by another user. For example, you can run a command like the following on the commit server.

```
$ p4 -c myclient unlock -xf myfile
```

An admin may also use this syntax to forcibly unlock files of type +1, which are not marked as orphaned, but which can no longer be unlocked by the user who opened them.

Options

`-c changelist`	Unlock files in pending changelist ***changelist***. This option applies to opened files in a pending changelist that were locked by **p4 lock** or a failed submit operation of an unshelved changelist.
`-f`	Superuser or administrator force option; allows unlocking of files opened by other users. You may now use a command like to following to unlock files of type =1
`-r`	Unlock the files associated with the specified client that were locked due to a failed **p4 push** command.
`-s shelvedchange`	If a file is locked in a pending shelved changelist, unlock it and keep it within the ***shelvedchange***. This can typically only happen if a **p4 submit -e** command is aborted.
`-x`	In distributed environments, unlock files that have the **+l** filetype (exclusive open) but have become orphaned (this is typically only necessary in the event of an extended network outage between an edge server and the commit server).
g-opts	See "Global Options" on page 571.

Usage Notes

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

Related Commands

To lock files so other users can't submit them	**p4 lock**
To display all your open, locked files (UNIX)	**p4 opened | grep "*locked*"**

p4 unshelve

Synopsis

Restore shelved files from a pending change into a workspace

Syntax

```
p4 [g-opts] unshelve -s shelvedchange [-f -n] [-c change]
                     [-b branch | -S stream [-P stream]] [file ...]
```

Description

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

In addition to the files being unshelved, **p4 unshelve** also unshelves any open stream specification. For more information on open stream specifications, see **p4 stream**.

You can limit the files to be unshelved by specifying a file pattern.

Unshelving copies the shelved files into the user's workspace as they existed when they were shelved. (For example, a file open for edit when shelved will also be open for edit in the unshelving user's workspace.)

You may unshelve a promoted shelf into open files and branches on a server from where the shelf did not originate.

Note	As a best practice, use the `-f` option to unshelve a shelf that has added files. **p4 obliterate myfile** does not obliterate a shelve of the file (archive or metadata). When you attempt to unshelve a file that has been obliterated, you will get an error. To recover the content of that file, print the file. To get rid of the shelve, delete the shelf.

Options

`-b branch`	Specifies a branch spec through which the shelved files will be mapped prior to unshelving. This option enables you to shelve files in one branch and unshelve them in another.
`-c change`	Specify a changelist number in the user's workspace into which the files are to be unshelved. By default, **p4 unshelve** retrieves files into the default changelist.
`-f`	Force the overwriting of writable (but unopened) files during the unshelve operation.

`-n`	Preview the results of the unshelve operation without actually restoring the files to your workspace.
`-P` *`stream`*	Unshelve to the specified parent stream. Overrides the parent defined in the source stream specification.
`-s` *`shelvedchange`*	Specify the pending changelist number that contains the originally-shelved files.
`-S` *`stream`*	Specifies the use of a stream-derived branch view to map the shelved files between the specified stream and its parent stream. See also the `-P` option.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- Unshelving a file over an already opened file is only permitted if both the shelved file and the opened file are opened for `edit`. (After unshelving, the workspace file is flagged as unresolved, and you must run **`p4 resolve`** to resolve the differences between the shelved file and the workspace file before submitting or reshelving the file.)

Related Commands

To store files from a pending changelist into the depot without submitting them.	**`p4 shelve`**

p4 unsubmit

Synopsis

Unsubmit a changelist, making it a shelved set of changes.

You may not issue this command directly to an edge server; you can issue it directly to a commit server.

Syntax

p4 [*g-opts*] unsubmit [-n] [-r *remote*] *filespec*[*revRange*] ...

Description

The **p4 unsubmit** command takes one or more submitted changelists and undoes the submission(s), leaving the changelist as a shelved change with the same content. The changelist can then be unshelved and further updated prior to resubmitting it.

The changelist must have been submitted by the same user and workspace which are used in the **p4 unsubmit** command. The files in the changelist must be the head revisions of those files in the server. The files must not have been integrated into any other files in the server. The files must not be open by any pending or shelved changelists. The files must not have been archived or purged. The files must not have associated attributes.

If the command specifies multiple files and/or multiple revisions, all the changelists which affected the specified revisions of the specified files are unsubmitted; each such change becomes its own separate shelf. Fix records linked to the changelist are not modified.

After unsubmitting a change which has associated jobs, you should review the job and fix status for accuracy. The shelved changelists that are created do not fire any triggers of type `shelve-submit` or `shelve-commit`.

After all the specified changelists have been unsubmitted, the **p4 unsubmit** command syncs the workspace to the head revision.

Options

`-n`	Performs all the correctness checks, but does not unsubmit any files.
`-r` *remotespec*	Specifies a remote spec. The map in the remote spec is used to limit the files affected by the **p4 unsubmit** command. Thus a command such as **p4 unsubmit -r rmt @>=17** will affect only the files specified by the remote spec, not all files in the depot.
filespec	The files whose changes will be unsubmitted.
revRange	If the file argument has a revision, the specified revision is unsubmitted. If the file argument has a revision range, the revisions in that range are unsubmitted.

Usage Notes

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

Examples

`p4 unsubmit //depot/foo#head`	Unsubmits the last change to `//depot/foo`.

Related Commands

To resubmit unsubmitted changelists	**`p4 resubmit`**

p4 unzip

Synopsis

Import files from a **p4 zip** package file.

Syntax

```
p4 [g-opts] unzip -i zipfile [-Ox] [-f -n -A -I -v --enable-dvcs-triggers]
```

Description

Imports the following into the specified zip file:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes association with the changelists
- all integration records that describe integrations to the files being unzipped

The value of the **rpl.checksum.change** configurable determines the level of verification performed for the **p4 unzip** command. See "Configurables" on page 595.

Triggering when unzipping

The following push trigger types may be invoked during the execution of the **p4 unzip** command:

- The **push-submit** trigger can customize processing during the phase of the **p4 unzip** command when metadata has been transferred but files have not yet been transferred.
- The **push-content** trigger can customize processing during that phase of the **p4 unzip** command when files have been transferred but their contents have not yet been committed.
- The **push-commit** trigger can do any clean up work or other post processing after changes have been committed by the **p4 unzip** command.

You must use the **--enable-dvcs-triggers** option to enable push triggers for the **p4 unzip** command.

For more information, see the section "Triggering on pushes and fetches" in the scripting chapter of *Helix Versioning Engine Administrator Guide: Fundamentals*.

Options

-A	Include the archive content of the new revisions.
--enable-dvcs-triggers	Enable any push triggers when the specified file is processed. Push triggers are disabled by default for the **p4 unzip** command.

	For more information about these kinds of triggers, see the chapter "Scripting Perforce: Scripts and Daemons" in *Helix Versioning Engine Administrator Guide: Fundamentals*.
`-f`	Bypasses the correctness checks.
`-i zipfile`	Specifies the zip file name.
`-I`	Excludes integration records for the new revisions.
`-n`	Performs all the correctness checks, but does not push any files or changelists to the remote server.
`-Oc`	When set, the **`p4 unzip`** command outputs information about every changelist. The `-v` option must be set for this to take effect.
`-Of`	When set, the **`p4 unzip`** command outputs information about every file in every changelist. The `-v` option must be set for this to take effect.
`-Oi`	When set, the **`p4 unzip`** command outputs information about every integration in every file in every changelist. The `-v` option must be set for this to take effect.
`-v`	Specifies verbose mode, which provides diagnostics for debugging. You must opt in to verbose mode. With verbose mode turned on, you can refine and control the precise level of verbosity. Specifically, you can indicate whether you want information about: • every changelist unzipped • every file in every changelist unzipped • every integration of every file in every changelist unzipped You can specify any combinations of these options, but must always include the `-O`. The default is to display information about every changelist.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Examples

`p4 unzip -i foo`	Imports the contents of the zip package named `foo`.

Related Commands

Package a set of files and their history for use by **`p4 unzip`**	**`p4 zip`**

p4 update

Synopsis

Update a client workspace without overwriting files that have changed since last sync.

Syntax

p4 [*g-opts*] update [-L -n -q] [*file*[*revRange*] ...]

Description

p4 update is an alias for a **p4 sync -s**.

Options

`-L`	For scripting purposes, perform the update on a list of valid file arguments in full depot syntax with a valid revision number.
`-n`	Display the results of the update without actually performing the update. This lets you make sure that the update does what you think it will do before you do it.
`-q`	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- **p4 update** is an alias for **p4 sync -s**.

Related Commands

p4 update is an alias for **p4 sync -s**	**p4 sync** -s
To copy files from the depot to the client workspace	**p4 sync**

p4 user

Synopsis

Create or edit Perforce user specifications and preferences.

Syntax

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

Description

Use the **p4 user** command to edit these settings or to create new user records. (By default, new users are created automatically. After installing Perforce, a Perforce superuser can control this behavior with the **p4 configure** command.)

There are three types of Perforce users: `standard` users, `operator` users, and `service` users. Standard users are the default, and each standard user consumes one Perforce license. The operator user type is intended for system administrators; they are subject to the same restrictions on permissions as any other user, but are further restricted in that they can run only a limited subset of Perforce commands. Service users are intended for inter-server communication in replicated and multi-server environments, and are restricted to an even smaller subset of Perforce commands. Neither `operator`s nor `service` users consume Perforce licenses.

When called without a *username*, **p4 user** edits the specification of the current user. When called with a *username*, the user specification is displayed, but cannot be changed. The form appears in the editor defined by the P4EDITOR environment variable.

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

Note that both the `-f` and `-F` options can be used to delete users, but the `-F` option has additional effects on protections and groups. See the description of command options below for more information.

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* option 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 has not been done, but the file pointed to by the P4CONFIG environment 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 variable has been set, then the command runs as that user.
- If none of the above apply, then the username is taken from the OS level USER or USERNAME environment variable.

Form Fields

Field Name	Type	Description
User:	Read-only	The Perforce username under which **p4 user** was invoked. By default, this is the user's system username.
Type:	Read-only	Type of user: **standard**, **operator**, or **service**. **Important** Once you set the type, you cannot change it.
AuthMethod:	Writable	One of the following: **perforce** or **ldap**. This field can only be changed when the **-f** option is specified for the **p4 user** command. • Specifying **perforce** enables authentication using Perforce's internal **db.user** table or by way of an authentication trigger. This is the default unless it is overridden with the **auth.default.method** configurable. • Specifying **ldap** enables authentication against AD/LDAP servers specified by the currently active LDAP configurations.
Email:	Writable	The user's email address. By default, this is ***user@client***.
Update:	Read-only	The date and time this specification was last updated.
Access:	Read-only	The date and time this user last ran a 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 "Usage Notes" on page 467).
Password:	Writable	The user's password (described in "Usage Notes" on page 467).
PasswordChange:	Read-only	The date and time of the user's last password change. If the user has no password, this field is blank.

Field Name	Type	Description
`Reviews:`	Writable List	A list of files the user would like to review (see "Usage Notes" on page 467). This field can include exclusionary mappings.

Options

`-d` *`username`*	Deletes the specified user. Only user *`username`*, or a Perforce superuser, can run this command. If you have set `P4AUTH`, no warning will be given if you delete a user who has an open file or client.
`-f`	Superuser force option; allows the superuser to modify or delete the specified user, or to change the last modified date.
`-F`	Superuser option; used only with `-d`: forces the deletion of the specified user, and also removes the user from the protections table and from all groups. The command fails if removing the user from a group causes the group to be deleted. In such a case, delete the group before deleting the user.
`-i`	Read the user specification from standard input. The input must conform to the **`p4 user`** form's format.
`-o`	Write the user specification to standard output.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

- The `-d` option can be used by non-superusers only to delete the user specification that invoked the **`p4 user`** command. 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` option. To prevent another user from impersonating you, set a password with the **`p4 passwd`** command.

  Passwords can be created, edited, or changed in the **`p4 user`** form or by using the **`p4 passwd`** command. Setting your password in the **`p4 user`** form is only supported at security levels 0 or 1.

You can **p4 passwd** to set passwords at any server security level, and you *must* use **p4 passwd** to set passwords at higher security levels. For more about how the various security levels work, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

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.

If the **dm.user.resetpassword** configurable has been set, all users created with passwords are required to reset their passwords before they can issue commands.

- Passwords are displayed as six asterisks in the **p4 user** form regardless of their length.
- If you are using ticket-based authentication (see **p4 login** for details), changing your password automatically invalidates all of your outstanding tickets.
- The collected values of the **Email:** fields can be listed for each user with the **p4 users** command, and can used for any purpose.
- The **p4 reviews** command, which is used by the 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.

- Operators are intended for system administrators who, even though they have super or admin privileges, are responsible for the maintenance of the Perforce service, rather than the development of software or other assets versioned by the service. Operators can run only the following commands:

p4 admin stop	p4 admin restart	p4 admin checkpoint
p4 admin journal	p4 counter	p4 counters
p4 dbstat	p4 dbverify	p4 diskspace
p4 configure	p4 counter (including -f)	p4 counters
p4 journaldbchecksums	p4 jobs (including -R)	p4 login
p4 logout	p4 logappend	p4 logparse
p4 logrotate	p4 logschema	p4 logstat
p4 logtail	p4 lockstat	p4 monitor
p4 passwd	p4 ping	p4 verify
p4 user		

- Service users are used in replication environments, and can run only the following commands:

p4 dbschema	p4 export	p4 login
p4 logout	p4 passwd	p4 info
p4 user		

Examples

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

Related Commands

To view a list of all Perforce users	p4 users

To change a user's password	**p4 passwd**
To view a list of users who have subscribed to review particular files	**p4 reviews**
To control how new users are created by changing the **dm.user.noautocreate** configurable	**p4 configure**

p4 users

Synopsis

Print a list of all known users of the current Perforce service.

Syntax

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

Description

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

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

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

Options

`-a`	Include service users in list.
`-c`	On replica servers, only user information from the master server are reported.
`-l`	Login information: includes time of last password change and login ticket expiry, if applicable. You must be a Perforce superuser to use this option.
`-m max`	List only the first *max* users.
`-r`	On replica servers, only users who have used this replica server are reported.
g-opts	See "Global Options" on page 571.

Usage Notes

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

- You must be connected to a replica to use `-c` or `-r`, and the `-c` and `-r` options are mutually exclusive.

- If the **run.users.authorize** configurable has been set to 1, users must authenticate themselves to the Perforce service before running the **p4 users** command.

Related Commands

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

p4 verify

Synopsis

Verify that the Perforce versioning service's archives are intact.

If **p4 verify** returns errors, contact Perforce technical support.

Syntax

```
p4 [g-opts] verify [-t | -u | -v | -z] [-m max -q -s -X -b batch N] file[revRange] ...
p4 [g-opts] verify -S [-t -m max -q -X -b N] file...
p4 [g-opts] verify -U unloadfile ...
p4 [g-opts] verify -A archivefiles ...
```

Description

For each revision of the specified files, **p4 verify** reports the revision specific information and an MD5 digest (fingerprint) of the revision's contents. See **p4 revisions** for information about specifying revisions.

The verification process involves opening the file's archive, reading it, and verifying that its digest matches the expected value. If the file can't be opened, verify reports `MISSING`. If the file can be opened, but does not have the expected contents, **p4 verify** reports `BAD`.

If invoked without arguments, **p4 verify** computes and displays the MD5 digest of each revision.

Syntax variants offer the following choices:

- Verify a given set of file revisions.
- Verify a given set of shelved file revisions.
- Verify file revisions in the unload depot.
- Verify file revisions in the archive depot.

It is good administrative practice to regularly verify the integrity of your depot files with **p4 verify -qz //...**

For details, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

Verifying shelved files

The verification of shelved files lets you know whether your shelved archives have been lost or damaged.

If a shelf is local to a specific edge server, you must issue the **p4 verify -S** command on the edge server where the shelf was created. If the shelf was promoted, run the **p4 verify -S** on the commit server.

You may also run the **p4 verify -S t** command on a replica to request re-transfer of a shelved archive that is missing or bad. Re-transferring a shelved archive from the master only works for shelved archives that are present on the master; that is, for a shelf that was originally created on the master or that was promoted if it was created on an edge server.

Verifying archived files

The verification of archived files lets you know whether your archived files have been damaged; it is a good practice to run a command like the following before you restore files with the **p4 restore** command.

```
$ p4 verify -A //depot/mysource/...
```

Options

`-A` *`archivefiles`*	It is possible for files in the archive depot to become corrupted over time; this option allows you to verify these files before you restore them with the **p4 restore** command. *`archivefiles`* specifies a file, a list of files, or set of files to verify; for example: `//depot/..../source`
`-b` *`batch`*	By default, **p4 verify** processes files in batches of 10,000 files at a time. You can change this batch size with the `-b` *`batch`* option. To disable batching, specify `-b 0`. If the `-z` option is specified, the `-b` option is ignored and all options are processed in a single batch.
`-m` *`max`*	Limit **p4 verify** to *`max`* number of revisions. Use this option with the `-u` option to compute and save digests for a limited number of revisions in each invocation of the **p4 verify** command.
`-q`	Run quietly; report only errors from mismatched digests or unreproducible revisions.
`-s`	Verify file size as well as digest. The `-v` implies the `-s` option.
`-S`	Verify shelved files. If you specify this option, the only valid revision specifier is `@=change`, specifying a single shelf. Use file patterns to restrict verification to a specific set of shelved files. In a distributed installation, this command should be run on the edge server where the shelf was created. If the shelf has been promoted, this command may also be run on the commit server.

`-t`	For use on replicas only:**`p4 verify -t`** causes the replica to schedule a transfer of the contents of any damaged or missing revisions. (This option also works on a replica with **`lbr.replication=cache`**.) The **`-t`** option cannot be used with the **`-v`** or **`-u`** options. In replicated environments, **`p4 verify -t`** reports **`BAD!`** or **`MISSING!`** files with **`(transfer scheduled)`** at the end of the line.
`-u`	Store the filesize and MD5 digest of each file in the Perforce database if no filesize and/or digest has been previously stored. Subsequent uses of **`p4 verify`** will compare the computed version against this stored version.
`-U` *`unloadfile`*	Verify files in the unload depot. See **`p4 unload`** for details.
`-v`	Store the MD5 digest of each file in the Perforce database, even if there's already a digest stored for that file, overwriting the existing digest. (The **`-v`** option is used only to update the saved digests of archive files that have been deliberately altered outside of Perforce control by a Perforce system administrator.) The **`-v`** and **`-u`** options are mutually exclusive.
`-X`	Skip files of filetype **`+X`** (for which the service runs an **`archive`** trigger.)
`-z`	Optimizes performance by skipping revisions that have already been computed in the current pass; this option speeds verifications for files that exist via lazy copies. The resulting output might report a lazy copy revision if it is the first revision in the sort order to access a common archive file. This option cannot be used with the **`-v`** or **`-u`** options.
`g-opts`	See "Global Options" on page 571.

Usage Notes

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

Examples

`p4 verify -S //depot/main/p4/...`	Verifies all shelved files matching the specified path.
`p4 verify -qS //...@=1023548`	Verifies the shelved files in shelf 1023548 only.
`p4 verify -q //...#head,#head`	Verifies only the head revision of all files in the depot, reporting only on files with problems.

Related Commands

To get information about how you can specify revisions.	**`p4 help revisions`**

p4 where

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

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

Options

g-opts	See "Global Options" on page 571.

Usage Notes

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

- The mappings are derived from the client view: a simple view, mapping the depot to one directory in the client workspace, produces one line of output.

  More complex client views produce multiple lines of output, possibly including exclusionary mappings. For instance, given the client view:

  ```
  View: //a/... //client/a/...
        //a/b/... //client/b/...
  ```

  Running:

  ```
  $ 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] [workspace]
p4 [g-opts] workspace -d [-f [-Fs]] workspace
p4 [g-opts] workspace -o [-t template] [workspace]
p4 [g-opts] workspace -S stream [[-c change] -o] [workspace]
p4 [g-opts] workspace -s [-f] -S stream [workspace]
p4 [g-opts] workspace -s [-f] -t template [workspace]
p4 [g-opts] workspace -i [-f]
```

Description

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

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

p4 workspaces

Synopsis

List all client workspaces currently known to the system.

Syntax

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

Description

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

p4 zip

Synopsis

Package a set of files and their history for use by **p4 unzip**.

Syntax

```
p4 zip -o file [-r remote -A -I] [filespec | -c change]

p4 zip -o file [-r remote -A -I] -s shelf
```

Description

Writes the following to the specified zip file:

- the specified set of files
- the changelists that submitted those files
- the files' attributes
- any fixes association with the changelists
- all integration records that describe integrations to the files being zipped

The content of the zip file can be specified either by providing a filespec, which selects a set of revisions, or by providing one or more changelist numbers using the **-c** option, which selects all the revisions modified by those changelists.

The second form of the command writes a single shelved changelist to the specified zip file. When you use the **-s** option, you must also use the **-A** option.

Options

-A	Include the archive content of the new revisions.
-c *change*	Zips up just the files covered by the specified changelist.
-I	Excludes integration records for the new revisions.
-o *filename*	Specifies the zip file name.
-r *remotespec*	Specifies the remote spec to be used to re-map the files when they are written to the zip file.
-s	Specifies a shelved changelist to be zipped, instead of one or more submitted changelists. For more information, see the section "Fetch and push a shelved changelist" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.

filespec Specifies the filepath of the files to zip up.

Usage Notes

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

Examples

`p4 zip -o foo //...` Creates a zip file named `foo` with all changes and revisions.

Related Commands

Import files from a **`p4 zip`** package file	**`p4 unzip`**

Environment and Registry Variables

Each operating system and shell has its own syntax for setting environment variables.The following table shows how to set the P4CLIENT environment variable on various systems:

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

Note

You may use `$home` to set environment variables. For example:

```
P4IGNORE=$home/myignorefile
```

`$home` is expanded to the path of the user's home directory. The user's home directory is taken to be the value of the `HOME` environment variable or of `USERPROFILE` on Windows.

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

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

Crucial Variables	Useful Variables	Esoteric Variables	Server Variables
P4CLIENT	P4CONFIG	P4PAGER	P4AUDIT

Crucial Variables	Useful Variables	Esoteric Variables	Server Variables
P4PORT	P4DIFF	PWD	P4JOURNAL
P4PASSWD	P4EDITOR	TMP, TEMP	P4LOG
P4USER	P4MERGE	P4TICKETS	P4PORT
	P4CHARSET	P4LANGUAGE	P4ROOT
	P4TRUST	P4LOGINSSO	P4DEBUG
		P4COMMANDCHARSET	P4NAME
		P4DIFFUNICODE	P4SSLDIR
		P4MERGEUNICODE	
		P4CLIENTPATH	

P4ALIASES

Description

The location of the directory that holds your `.p4aliases` file.

Usage Notes

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

If `P4ALIASES` is not defined, the system looks in the `$HOME` directory on Unix and Mac systems, and in your `$USERPROFILE` directory on Windows.

For more information on custom command aliases, see "Command aliases" on page 2.

Examples

```
/usr/bin
/customize
```

P4AUDIT

Description

Location of the audit log file.

Usage Notes

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

Value if not Explicitly Set

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

Notes

`P4AUDIT` specifies the location of the audit log file.

When auditing is enabled, Perforce adds a line to the audit log file every time file content is transferred from the shared versioning service to any user. On an active installation, the audit log file will grow very quickly.

Lines in the audit log appear in the form:

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

For example:

```
2011/05/09 09:52:45 karl@nail 192.168.0.12 diff //depot/src/x.c#1
2011/05/09 09:54:13 jim@stone 127.0.0.1 sync //depot/inc/file.h#1
```

If a command is run on the same physical machine that hosts the Perforce service, the *`clientIP`* is shown as `127.0.0.1`.

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

In order to ensure that user activity on replica and edge servers (specifically in environments involving build farm replicas, forwarding replicas, and/or edge servers) is tracked, each replica or edge server must have `P4AUDIT` configured.

For more information, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4AUTH

Description

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

Usage Notes

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

Value if not Explicitly Set

Program	Value
Perforce Servers	`null`

Examples

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

Notes

The format of `P4AUTH` is *`host:port`*, or *`port`* by itself if both the Perforce server and the authorization server are running on the same host. All servers must be at the same release level.

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

Warning	If you have set `P4AUTH`, no warning will be given if you delete a user who has an open file or client.

For more information about central authorization servers, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4BROKEROPTIONS

Description

Set Perforce Broker options for a Windows service.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	**p4broker %P4BROKEROPTIONS%**	N/A

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

For example, if you normally run the Broker with the command:

```
p4broker -c c:\p4broker\broker.conf
```

you can set the `P4BROKEROPTIONS` variable for the Windows service to run with:

```
p4 set -S "Broker" P4BROKEROPTIONS= "-c c:\p4broker\broker.conf"
```

When you run P4Broker under the `"Broker"` service, the Broker will configure itself using the specified `broker.conf` file. Use `P4BROKEROPTIONS` when you need to call **`p4broker`** with options for which there are no corresponding environment variables, or when you are doing so within the context of a Windows service.

For more information on the Perforce Broker, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4CHANGE

Description

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

Usage Notes

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

Value if not Explicitly Set

Program	Value
Perforce Servers	`null`

Examples

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

Notes

The format of `P4CHANGE` is ***`host:port`***, or *port* by itself if both the Perforce server and the changelist server are running on the same host. All servers must be at the same release level.

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

For more information about changelist servers, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4CHARSET

Description

Character set used for translation of unicode files.

Usage Notes

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

Value if not Explicitly Set

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

Notes

If the server is set to Unicode-mode, the client sets P4CHARSET to **auto** and examines the client's environment to determine the character set to use in converting files of type **unicode**. Thus, the only time you need to set P4CHARSET to a specific type is if the client's choice of charset results in a faulty conversion or if you have other special needs. For example, the application that uses the checked out files expects a specific character set.

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

For Perforce services operating in the default (non-Unicode mode), P4CHARSET must be left unset (or set to **none**) on user workstations. If P4CHARSET is set, but the service is not operating in internationalized mode, the service returns the following error message:

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

For Perforce services operating in Unicode mode, **P4CHARSET** must either be set to **auto** or be set to some value (other than **none**) on user machines. If **P4CHARSET** is unset, but the service is operating in Unicode mode, Perforce applications return the following error message:

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

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

http://www.perforce.com/perforce/r16.2/user/i18nnotes.txt

In addition to affecting the client, Unicode settings also affect trigger scripts that communicate with the server. You should check your trigger's use of file names, Perforce identifiers, and files containing unicode characters, and make sure that these are consistent with the character set used by the server. See the URL cited above for details about Unicode settings.

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

P4_port_CHARSET

Description

Specifies whether the server is in Unicode mode.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	`-C` option	Yes

Value if not Explicitly Set

Operating System	Value
All	None, which specifies that the server is not in Unicode mode.

Notes

When a client connects to the server, it attempts to discover the server's Unicode mode setting, and it sets the `P4_port_CHARSET` variable to specify that setting: for non-Unicode, the variable is set to `none`; for Unicode, the variable is set to `auto`. If `P4_port_CHARSET` is set to `auto`, the client sets the `P4CHARSET` to `auto`. The client then examines its own environment to determine what character set it needs to use in appropriately rendering unicode files from the server.

The *port* part of this environment variable specifies the *host:port* of the server to which the client is connected.

For more information about using servers in Unicode mode, see "Setting up and managing Unicode installations" in the *Helix Versioning Engine Administrator Guide: Fundamentals*.

P4CLIENT

Description

Name of current client workspace.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	**`p4 -c clientname cmd`**	Yes

Value if not Explicitly Set

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

Examples

```
cinnamon
```

```
computer1
```

```
WORKSTATION
```

P4CLIENTPATH

Description

A list of directories to which Perforce applications are permitted to write.

Any attempt by a Perforce application to access or modify files outside these areas of the filesystem will result in an error message.

To specify more than one directory, separate the directories with semicolons.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
"C:\Users\Joe Coder"

/usr/team/joe/workspace/buildfarm/joe
```

P4COMMANDCHARSET

Description

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

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

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

http://www.perforce.com/perforce/r16.2/user/i18nnotes.txt

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

P4CONFIG

Description

Contains a file name without a path. The specified file is used to store other Perforce environment variables.

The current working directory (returned by PWD) and its parents are searched for the file. If a file is found, the variable settings within the file are used. If additional files are found in parent directories, and they contain variable settings not already found in other files, those variable settings are also used.

Usage Notes

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

Value if not Explicitly Set

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

Examples

A sample P4CONFIG file might contain the following lines:

```
P4CLIENT=joes_client
P4USER=joe
P4PORT=ssl:ida:3548
```

Notes

P4CONFIG makes it trivial to switch 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.

Common settings may be placed into a configuration file in a parent directory. These settings may be overridden by setting them in configuration files in child directories.

The file defined by P4ENVIRO contains the same kind of information as the P4CONFIG file. The difference is that the P4CONFIG variable contains just the file name of a configuration file, for which the system

searches through successive parent directories; the P4ENVIRO variable contains the exact location and name of a configuration file if it is not at its default location.

Each line in the configuration file defines one variable; the definition takes the form ***variable=value***.

You can use both **P4CONFIG** and P4ENVIRO files to define environment variables: use the **P4CONFIG** file for those variables that have different values for different workspaces and the P4ENVIRO file for those variables that remain constant for all projects. Values set in a **P4CONFIG** file override those set in a P4ENVIRO file.

Common variables to set within a **P4CONFIG** file include the following:

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

P4DEBUG

Description

Set Perforce server or proxy trace options.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	As of Release 2011.1, `server=1` is the default setting.

Examples

```
server=0
server=1
server=2
server=3
```

Notes

To disable logging, set `P4DEBUG` to `server=0`.

Higher settings for the Perforce server trace options are useful only to administrators working with Perforce Technical Support to diagnose or investigate a problem. The proxy does not set debugging by default.

The preferred way to set trace options for the 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 options within `P4DEBUG`, allowing the service to run with the options enabled.

Setting server debug levels on a Perforce server (**`p4d`**) has no effect on the debug level of a Perforce Proxy (**`p4p`**) process, nor on downstream replicas or edge servers.

For further information about server trace options, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

P4DESCRIPTION

Description

An optional description for a Perforce server.

`P4DESCRIPTION` is used by **`p4 server`** as a means of identifying servers.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
"Commit server"

"Replica server"

"Build farm"
```

P4DIFF

Description

The name and location of the diff program used by **p4 resolve** and **p4 diff**.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
Windows	If the environment variable `DIFF` has been set, then the value of `DIFF`; otherwise, if the environment variable `SHELL` has been set to *any* value, then the program diff is used; otherwise, **p4diff.exe**.
All Others	If the environment variable `DIFF` has been set, then the value of `DIFF`; otherwise, Perforce's internal diff routine is used.

Examples

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

Notes

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

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

P4DIFFUNICODE

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/r16.2/user/relnotes.txt

P4EDITOR

Description

The editor invoked by those Perforce commands that use forms.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
UNIX and OS X	If `EDITOR` is set to any value, then the value of `EDITOR`; otherwise, **`vi`**.
Windows	If `SHELL` is set to any value, then **`vi`**; otherwise, **`notepad`**.
VMS	If `POSIX$SHELL` is set, then **`vi`**; otherwise, **`edit`**.
Mac	If `EDITOR_SIGNATURE` is set, then the program with that four-character creator; otherwise, **`SimpleText`**.

Examples

```
/usr/bin/vi
```

```
emacs
```

```
/usr/bin/vi
```

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`**.

P4ENVIRO

Description

Contains the non-default path and name of a configuration file that stores Perforce environment variables.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
Windows, Mac OS X	None
POSIX/Unix	`$HOME/.p4enviro`

Notes

The file specified by `P4ENVIRO` contains the same kind of information as the file specified by `P4CONFIG`. The difference is that the `P4CONFIG` variable contains just the file name of a configuration file for which the system searches through successive parent directories; the `P4ENVIRO` variable contains the exact location of a configuration file if it is not at its default location. For Windows and Mac OS X platforms, the `P4ENVIRO` variable must be explicitly set if you have values stored in a configuration file you mean to use across projects.

Each line in the `P4ENVIRO` file is used to define one variable; the definition takes the form *`variable=value`*.

You can use both `P4ENVIRO` and `P4CONFIG` files to define environment variables: use the `P4CONFIG` file for those variables that have different values for different workspaces and the `P4ENVIRO` file for those variables that remain constant for all projects. Values set in a `P4CONFIG` file override those set in a `P4ENVIRO` file.

Note	Setting `P4ENVIRO` on Windows will cause **`p4 set`** to store values in the specified environment file rather than in the Windows registry.

Examples

A sample `P4ENVIRO` file might contain the following line:

```
P4_myServer:1667_CHARSET=auto
```

P4HOST

Description

Name of host computer to impersonate.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	**p4 -H *hostname command***	Yes

Value if not Explicitly Set

Operating System	Value
All	The value of the client hostname as returned by **p4 info**.

Examples

```
workstation123.perforce.com
```

Notes

Perforce users can use the **Host:** field of the **p4 client** form to specify that a particular client workspace can be used only from a particular host machine. When this field has been set, the P4HOST variable can be used to fool the service into thinking that the user is on the specified host machine regardless of the machine being used by the user. As this is a very esoteric need, there's usually no reason to set this variable.

The hostname must be provided exactly as it appears in the output of **p4 info** when run from that host.

P4IGNORE

Description

Specify a list of files that contain lists of rules for ignoring files when adding files to the depot and reconciling workspaces.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	none

Examples

```
.p4ignore
.p4ignore;$home/.myp4ignore
```

Notes

Note

You may use `$home` to set environment variables. For example

```
P4IGNORE=$home/myignorefile
```

The syntax for the contents of a `P4IGNORE` file is *not* the same as Perforce syntax. Instead, it is similar to that used by other versioning systems:

- Rules are specified using local filepath syntax. Unix style paths will work on Windows for cross platform file support.
- A `#` character at the beginning of a line denotes a comment.

  You can escape the `#` comment character with a backslash (`\`). This allows filenames beginning with `#` to be ignored.
- A `!` character at the beginning of a line line excludes the file specification. These exclusions override rules defined above it in the `P4IGNORE` file, but may be overridden by later rules.

- A / (or \ on Windows) character at the beginning of a line causes the file specification to be considered relative to the P4IGNORE file. This is useful when the rule must apply to files at particular depots of the directory tree.
- A / (or \ on Windows) character at the end of a line causes the file specification to only match directories, and not files of the same name.
- The * wildcard matches substrings. Like the Perforce wildcard equivalent, it does not match path separators; however, if it is not used as part of a path, the directory scanning nature of the rule may make it appear to perform like the Perforce "**...**" wildcard.
- The ** wildcard matches substrings including path separators. It is equivalent to the Perforce "**...**" wildcard, which is not permitted.

For example:

```
# Ignore .p4ignore files
.p4ignore

# Ignore object files, shared libraries, executables
*.dll
*.so
*.exe
*.o

# Ignore all HTML files except the readme file
*.html
!readme.html

# Ignore the bin directory
bin/

# Ignore the build.properties file in this directory
/build.properties

# Ignore all text files in test directories
test/**.txt
```

The first match is used when figuring out what to ignore.

Use the **p4 ignores** command to get information about why a file is being ignored during add and reconcile operations. For example, the following command, lets you know which line of the **P4IGNORE** file is being used to ignore a file.

```
$ p4 ignores -v -i mypath
```

P4JOURNAL

Description

A file that holds the Perforce database's journal data.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	P4ROOT/journal

Examples

```
journal
off
/disk2/perforce/journal
```

Notes

If a relative path is provided, it should be specified relative to the Perforce server root.

Setting `P4JOURNAL` to `off` disables journaling. This is not recommended.

For further information, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

P4LANGUAGE

Description

This environment variable is reserved for system integrators.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	**p4 -L** ***language cmd***	Yes

Value if not Explicitly Set

Operating System	Value
All	N/A

P4LOG

Description

Name and path of the file to which Perforce errors are written.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	**`p4d -L`** ***`file`*** **`p4p -L`** ***`file`***	N/A

Value if not Explicitly Set

Operating System	Value
All	Standard error

Examples

```
log
/disk2/perforce/log
```

Notes

If a relative path is provided, it is specified relative to the Perforce server root.

For further information, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

P4LOGINSSO

Description

Client-side single-signon script.

Usage Notes

Triggers of type `auth-check-sso` fire when standard users run the **`p4 login`** command. Two scripts are run: a client-side script is run on the user's workstation, and its output is passed (in plaintext) to the Perforce Server, where the server-side script runs.

- On the user's client workstation, a script (whose location is specified by the `P4LOGINSSO` environment variable) is run to obtain the user's credentials or other information verifiable by the Perforce Server. The `P4LOGINSSO` contains the name of the client-side script and zero or more of the following trigger variables, passed as parameters to the script: `%user%`, `%serverAddress%`, and `%P4PORT%`. For example,

  ```
  $ export P4LOGINSSO="/path/to/sso-client.sh %user% %serverAddress% %P4PORT%"
  ```

  Where `%user%` is the Perforce client user, `%serverAddress%` is the address of the target Perforce server, and `%P4PORT%` is an intermediary between the client and the server.

- On the server, the output of the client-side script is passed to the server-side script as standard input. The server-side script specified in the trigger table runs, and the server returns an exit status of 0 if successful.

  With a distributed configuration in which a proxy or broker acts as an intermediary between the client and the server, the `%serverAddress%` variable will hold the address/port of the server and the `%P4PORT%` variable will hold the port of the intermediary. It is up to the script to decide what to do with this information.

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

Value if not Explicitly Set

Operating System	Value
All	N/A

Examples

`/Users/joe/bin/runsso`

Notes

For further information, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

P4MERGE

Description

A third-party merge program to be used by **p4 resolve**'s merge option.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	If the `MERGE` environment variable is set, then its value; otherwise, nothing.

Examples

`c:\Perforce\p4merge.exe`

`c:\progra~1\Perforce\p4merge.exe`

Notes

The program represented by the program name stored in this variable is used only by **p4 resolve**'s merge option. When **p4 resolve** calls this program, it passes four arguments, representing (in order) *base*, *theirs*, and *yours*, with the fourth argument holding the resulting *merge* file.

If the program you use takes its arguments in a different order, set P4MERGE to a shell script or batch file that reorders the arguments and calls the proper merge program with the arguments in the correct order.

If you are running under Windows, you must call a batch file, even if your third-party merge program already accepts arguments in the order provided by Perforce. This is due to a limitation within Windows. For instance, if you want to use a program called **`MERGE.EXE`** under Windows, your batch file might look something like this:

```
SET base=%1
SET theirs=%2
SET yours=%3
SET merge=%4
C:\FULL\PATH\TO\MERGE.EXE %base% %theirs% %yours% %merge%
```

P4MERGEUNICODE

Description

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

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None

Notes

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

http://www.perforce.com/perforce/r16.2/user/relnotes.txt

P4NAME

Description

A unique identifiable name for a Perforce server.

This configurable cannot be set globally; you must specify a server id.

`P4NAME` is used by **`p4 configure`** as a means of identifying servers.

Unless a `P4NAME` value has been specified for the server, the server uses the serverid to determine the appropriate configuration settings. See **`p4 serverid`**.

Warning	On Windows if there is no `P4NAME` defined in the registry for a service, it is picked up from the name of the service itself.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None

Examples

```
masterserver

failoverserver

buildserver
```

P4PAGER

Description

The program used to page output from **p4 resolve**'s diff option.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	If the variable `PAGER` is set, then the value of `PAGER`; otherwise, none.

Examples

`/bin/more` (UNIX)

Notes

The value of this variable is used *only* to display the output for **p4 resolve**'s diff routine. If the variable is not set, the output is not paged.

P4PASSWD

Description

Supplies the current Perforce user's password for any Perforce command.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	**p4 -P *passwd command***	Yes

Value if not Explicitly Set

Operating System	Value
All	None

Notes

Perforce passwords are set via **p4 passwd**, or in the form invoked by **p4 user**. The setting of P4PASSWD is used to verify the user's identity. If a password has not been set, the value P4PASSWD is not used, even if set.

While it is possible to manually set the P4PASSWD environment variable to your plaintext password, the more secure way is to use the **p4 passwd** command. On UNIX, this will invoke a challenge/response mechanism which securely verifies your password. On Windows, this sets P4PASSWD to the encrypted MD5 hash of your password.

On Windows platforms, if you set a password in P4V, the value of the registry variable P4PASSWD is set for you. Setting the password in P4V is like using **p4 passwd** (or **p4 set** P4PASSWD) from the MS-DOS command line, setting the registry variable to the encrypted MD5 hash of the password. The unencrypted password itself is never stored in the registry.

If you are using ticket-based authentication, but have a script that relies on a P4PASSWD setting, use **p4 login -p** to display the value of a ticket that can be passed to Perforce commands as though it were a password (that is, either from the command line, or by setting P4PASSWD to the value of the valid ticket).

P4PCACHE

Description

For the Perforce Proxy, the directory in which the proxy stores its files and subdirectories.

Usage Notes

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

Value if not Explicitly Set

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

Notes

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

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

For more information on setting up a Perforce Proxy, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

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 *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4POPTIONS

Description

Set Perforce Proxy options for a Windows service.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	**p4p %P4POPTIONS%**	N/A

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

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

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

you can set the P4POPTIONS variable for the Windows **proxysvc** to run with

```
$ p4 set -S "Perforce Proxy" P4POPTIONS="-p 1999 -t mainserver:1666"
```

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

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

For more information on setting up a Perforce Proxy, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4PORT

Description

For the Perforce service (server, broker, or proxy), the port number on which it listens, and the network transport(s) to which it is to bind.

For Perforce applications, the protocol, host and port number of the Perforce service with which to communicate. The most commonly-used communications protocols are `tcp` (plaintext over TCP/IP) or `ssl` (SSL over TCP/IP).

Perforce supports connectivity over IPv6 networks as well as over IPv4 networks. You can specify whether you require (or merely prefer) to use IPv4 or IPv6 addresses when resolving hostnames. The protocol settings of `tcp4` and `ssl4` require IPv4 address support. Similarly, `tcp6` and `ssl6` require IPv6 support. Using `tcp64` and `ssl64` attempts first to resolve the host to an IPv6 address, but will accept an IPv4 address if IPv6 is not available. The opposite behavior is available with `tcp46` and `ssl46`; these default to the use of IPv4 if possible, and use IPv6 if IPv4 is unavailable. A configurable, `net.rfc3484`, may be set on user workstations or in `P4CONFIG` files in order to permit the operating system to automatically determine which transport to use.

Behavior and performance of networked services is contingent not merely upon the networking capabilities of the machine that hosts the service, nor only on the operating systems used by the end users, but also on your specific LAN and WAN infrastructure (and the state of IPv6 support for every router between the end user and the Perforce versioning service).

To illustrate just one possible scenario, a user working from home; even if they have an IPv6-based home network, their ISP or VPN provider may not fully support IPv6. We have consequently provided several variations on `P4PORT` to provide maximum flexibility and backwards compatibility for administrators and users during the transition from IPv4 to IPv6.

P4PORT protocol value	Behavior in IPv4/IPv6 or mixed networks
`<not set>`	Use `tcp4:` behavior, but if the address is numeric and contains two or more colons, assume `tcp6:` If the `net.rfc3484` configurable is set, allow the OS to determine which transport is used.
`tcp:`	Use `tcp4:` behavior, but if the address is numeric and contains two or more colons, assume `tcp6:` If the `net.rfc3484` configurable is set, allow the OS to determine which transport is used.
`tcp4:`	Listen on/connect to an IPv4 address/port only.
`tcp6:`	Listen on/connect to an IPv6 address/port only.
`tcp46:`	Attempt to listen/connect to an IPv4 address. If this fails, try IPv6.
`tcp64:`	Attempt to listen/connect to an IPv6 address. If this fails, try IPv4.
`ssl:`	Use `ssl4:` behavior, but if the address is numeric and contains two or more colons, assume `ssl6:` If the `net.rfc3484` configurable is set, allow the OS to determine which transport is used.

P4PORT protocol value	Behavior in IPv4/IPv6 or mixed networks
`ssl4:`	Listen on/connect to an IPv4 address/port only, using SSL encryption.
`ssl6:`	Listen on/connect to an IPv6 address/port only, using SSL encryption.
`ssl46:`	Listen on/connect to an IPv4 address/port. If that fails, try IPv6. After connecting, require SSL encryption.
`ssl64:`	Listen on/connect to an IPv6 address/port. If that fails, try IPv4. After connecting, require SSL encryption.

In mixed environments it is good practice to set the `net.rfc3484` configurable to `1`:

```
p4 configure set net.rfc3484=1
```

Doing so ensures RFC3484-compliant behavior for users who do not explicitly specify the protocol value; that is, if the client-side configurable `net.rfc3484` is set to `1`, and P4PORT is set to `example.com:1666`, or `tcp:example.com:1666`, or `ssl:example.com:1666`, the user's operating system will automatically determine, for any given connection, whether to use IPv4 or IPv6.

If you use SSL to connect to Perforce, the fingerprint of the Perforce server must match that stored in the P4TRUST file. (When you connect to a new Perforce installation for the first time, the server's fingerprint is displayed. If it matches the one your administrator has assigned it, you may safely connect to the server by using the **`p4 trust`** command to add the server to your P4TRUST file.)

Usage Notes

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

Value if not Explicitly Set

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

Examples

Perforce application	Perforce versioning service
`1818`	`1818`
`ssl:squid:1234`	`ssl:1234`
`example.com:1234`	`1234`
`ssl:192.168.0.123:1818`	`ssl:1818`
`tcp6:[2001:db8::123]:1818`	`tcp6:[::]:1818`
`tcp6:example.com:1818`	`tcp6:[::]:1818`
`ssl64:[2001:db8::123]:1818`	`ssl6:[::]:1818ssl64:[::]:1818`

Notes

The format of P4PORT for Perforce applications is ***protocol*:*host*:*port***, or ***port*** by itself if both the Perforce application and versioning service are running on the same host. Port numbers must be in the range **1024** through **32767**.

If you specify both an IP address *and* a port number in P4PORT, the Perforce versioning service ignores requests from any IP addresses other than the one specified in P4PORT.

If you do not specify a protocol, transmissions between Perforce applications and the Perforce versioning service are performed in plaintext, and IPv4 addresses are assumed.

P4ROOT

Description

Directory in which the Perforce service stores its versioned files and its database files.

Usage Notes

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

Value if not Explicitly Set

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

Notes

Create this directory before starting the Perforce versioning service (**p4d**).

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

For more information on setting up a Perforce installation, see the *Helix Versioning Engine Administrator Guide: Fundamentals* and *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

P4SSLDIR

Description

Directory containing a server's SSL keys and/or certificates.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	If P4SSLDIR is set to any value, the directory named by P4SSLDIR stores the files that contain server ssl credentials. If this variable is unset, or if the permissions of P4SSLDIR or its contents are incorrect, the service will not start in SSL mode.

Examples

```
/path/to/dir
```

Notes

All Perforce server processes (**p4d**, **p4p**, **p4broker**) that accept SSL connections require a certificate and key pair (stored in this directory) on startup. In order for any of these processes to start, the following additional conditions must be met:

- P4SSLDIR must be set to a valid directory.
- The P4SSLDIR directory must be owned by the same userid as the one running the Perforce server, proxy, or broker process. The P4SSLDIR directory must not be readable by any other user. On UNIX, for example, the directory's permissions must be set to 0700 (`drwx------`) or 0500 (`dr-x------`).
- Two files, named `privatekey.txt` and `certificate.txt`, must exist in P4SSLDIR.

  These files correspond to the PEM-encoded unencrypted private key and certificate used for the SSL connection. They must be owned by the userid that runs the Perforce server, proxy, and broker process, and must also have their permissions set such as to make them unreadable by other users. On UNIX, for example, the files' permissions must be set to 0600 (`-rw-------`) or 0400 (`-r--------`).

You can supply your own private key and certificate, or you can use **p4d -Gc** to generate a key and certificate pair. For more information, see the *Helix Versioning Engine Administrator Guide: Fundamentals*.

- To generate a fingerprint from your server's private key and certificate, run **p4d -Gf**. (P4SSLDIR must be configured with the correct file names and permissions, and the current date must be valid for the certificate.)

  After you have communicated this fingerprint to your end users, your end users can then compare the fingerprint the server offers with the fingerprint you have provided. If the two fingerprints match, users can use **p4 trust** to add the fingerprint to their P4TRUST files.

P4TARGET

Description

For the Perforce Proxy and replica servers, the name and port number of the target Perforce server (that is, the Perforce server for which P4P acts as a proxy; for a replica or edge server, the upstream master or commit server from which it retrieves its data, and towards which changelists, if applicable, are forwarded.)

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	**p4p -t *host:port*** **p4d -t *host:port***	N/A

Value if not Explicitly Set

Program	Value
Perforce Proxy	`perforce:1666`
Replicated environments	None

Examples

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

Notes

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

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

For more about replicas, edge servers, and the Perforce Proxy, see *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.

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	Yes	N/A	Yes

Value if not Explicitly Set

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

Examples

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

Notes

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

P4TRUST

Description

Specifies the path to the SSL trust file. The trust file contains the fingerprints of the keys used for SSL connections; it is controlled by the **p4 trust** command.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	Yes	N/A	Yes

The `P4TRUST` configurable is used by any service acting as a client to any SSL-enabled server. This includes client applications, proxies, brokers, replica servers, edge servers, and so on. How each of these uses the configurable is covered in the chapters describing the service.

Value if not Explicitly Set

Program	Value
Windows	`%USERPROFILE%\p4trust.txt`
All others	`$HOME/.p4trust`

Notes

Your system administrator can help you confirm the accuracy of any fingerprint (or change to a fingerprint) provided to you by a Perforce server.

P4USER

Description

Current Perforce username.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	**p4 -u *username command***	Yes

Value if not Explicitly Set

Operating System	Value
Windows	The value of the `USERNAME` environment variable.
All Others	The value of the `USER` environment variable.

Examples

```
edk
lisag
```

Notes

By default, the Perforce username is the same as the OS username.

If a particular Perforce user does not have a password set, then any other Perforce user can impersonate this user by using the `-u` option with their Perforce 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 *Helix Versioning Engine Administrator Guide: Fundamentals*.

PWD

Description

The directory used to resolve relative filename arguments to Perforce commands.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	**`p4 -d`** ***`directory command`***	No

Value if not Explicitly Set

Operating System	Value
UNIX	The value of `PWD` as set by the shell; if not set by the shell, `getcwd()` is used.
All Others	The actual current working directory.

Notes

Sometimes the `PWD` variable is not inherited properly across shells. For instance, if you're running **`ksh`** or **`sh`** on top of **`csh`**, `PWD` will be inherited from your **`csh`** environment but not updated properly, causing possible confusion in subsequent 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 environment variables unset in your startup files; the confusion only occurs when variables are exported to sub-shells.)

TMP, TEMP

Description

The directory to which Perforce applications and services write temporary files.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
UNIX	`/tmp`
All Others	For Perforce applications: 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 are written in the directories described in the table above.

When using triggers on Windows, `%formfile%` and other variables that use a temp directory should use the `TMP` and `TEMP` system variables in Windows, *not* the user's `TEMP` variables.

Additional Information

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

- "Global Options" on page 571 that can be used with any Perforce command,
- How to use Perforce "File Specifications" on page 575 in depot syntax, client syntax, and local syntax,
- How to create and use "Views" on page 581 to describe client workspaces, branches, and labels,
- Perforce "File Types" on page 585, and
- Server "Configurables" on page 595.

For an in-depth treatment of these and other topics from a conceptual level, see *Introducing Helix*, which is available at our web site: http://www.perforce.com/documentation.

Global Options

Synopsis

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

Syntax

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

Options

`-b` *`batchsize`*	Specifies a batch size (number of arguments) to use when processing a command from a file with the `-x` *`argfile`* option. By default, the batch size is 128.
`-c` *`client`*	Overrides any `P4CLIENT` setting with the specified client name.
`-d` *`dir`*	Overrides any `PWD` setting (current working directory) and replaces it with the specified directory.
`-I`	Specify that progress indicators, if available, are desired. This option is not compatible with the `-s` and `-G` options.
`-G`	Causes all output (and batch input for form commands with `-i`) to be formatted as marshaled Python dictionary objects. This is most often used when scripting.
`-H` *`host`*	Overrides any `P4HOST` setting and replaces it with the specified hostname.
`-p` *`port`*	Overrides any `P4PORT` setting with the specified *`protocol:host:port`*.
`-P` *`pass`*	Overrides any `P4PASSWD` setting with the specified password.
`-r` *`retries`*	Specifies the number of times to retry a command (notably, **`p4 sync`**) if the network times out.
`-s`	Prepends a descriptive field (for example, `text:`, `info:`, `error:`, `exit:`) to each line of output produced by a Perforce command. This is most often used when scripting.
`-u` *`user`*	Overrides any `P4USER`, `USER`, or `USERNAME` setting with the specified user name.
`-x` *`argfile`*	Instructs Perforce to read arguments, one per line, from the specified file. If file is a single hyphen (`-`), then standard input is read.
`-C` *`charset`*	Overrides any `P4CHARSET` setting with the specified character set.

`-Q charset`	Overrides any P4COMMANDCHARSET setting with the specified character set.
`-L language`	This feature is reserved for system integrators.
`-z tag`	Causes output of many reporting commands to be in the same tagged format as that generated by **p4 fstat**.
`-q`	Quiet mode; suppress all informational message and report only warnings or errors.
`-V`	Displays the version of the **p4** application and exits.
`-h`	Displays basic usage information and exits.

Usage Notes

- Be aware that the global options must be specified on the command line before the 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 - edit** command.

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

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

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

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

  In some cases, it may not be intuitively obvious what keys are used by the application. If you pipe the output of any **p4 -G** invocation to the following script, you will see every record printed out in key/value pairs:

```
#!/usr/local/bin/python

import marshal, sys

try:
    num=0
    while 1:
        num=num+1
        print '\n' % num
        dict = marshal.load(sys.stdin)
        for key in dict.keys(): print "%s: %s" % (key,dict[key])

except EOFError: pass
```

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

For additional information about this option, see

http://answers.perforce.com/articles/KB/3518

- At present, the progress indicator requested when you use the `-I` option is only available with **`p4 -I submit`** and **`p4 -I sync -q`**.
- Some uses of the global options are absurd.

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

Examples

`p4 -p` *`new_service`*`:1234 sync`	Performs a sync after connecting to *`new_service`* and port `1234`, regardless of the settings of the P4PORT environment variable.
`p4 -c` *`new_client`* `submit -c 100`	The first `-c` is the global option to specify the client workspace name. The second `-c` specifies a changelist number.
`p4 -s -x filelist.txt edit`	If `filelist.txt` contains a list of files, this command opens each file on the list for editing, and produces output suitable for parsing by scripts. Any errors as a result of the automated **`p4 edit`** commands (for example, a file in `filelist.txt` not being found) can then be easily detected by examining the command's output for lines beginning with "`error:`"

File Specifications

Synopsis

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

Syntax forms

Local syntax refers to filenames as specified by the local shell or operating system. Filenames referred to in local syntax can be specified by their absolute paths or relative to the current working directory. (Relative path components can only appear at the beginning of a file specifier.)

Perforce has its own method of file specification which remains unchanged across operating systems. If a file is specified relative to a client root, it is said to be in *client syntax*. If it is specified relative to the top of the depot, it is said to be in *depot syntax*. A file specified in either manner can be said to have been specified in Perforce syntax.

Perforce file specifiers always begin with two slashes (`//`), followed by the client or depot name, followed by the full pathname of the file relative to the client or depot root directory.

Path components in client and depot syntax are always separated by slashes (`/`), regardless of the component separator used by the local operating system or shell.

An example of each syntax is provided below

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

Wildcards

The Perforce system allows the use of three wildcards:

Wildcard	Meaning
`*`	Matches all characters except slashes within one directory.
`...`	Matches all files under the current working directory and all subdirectories. (matches anything, including slashes, and does so across subdirectories)
`%%1 - %%9`	Positional specifiers for substring rearrangement in filenames, when used in views.

For example:

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

Using revision specifiers

File specifiers can be modified by appending `#` or `@` to them.

The `#` and `@` specifiers refer to specific revisions of files as stored in the depot:

Modifier	Meaning
`file``#`*`n`*	Revision specifier: The *`n`*th revision of *`file`*.
`file``#none`	The nonexistent revision: If a revision of *`file`* exists in the depot, it is ignored. This is useful when you want to remove a file from the client workspace while leaving it intact in the depot, as in **`p4 sync`** *`file`*`#none`. The filespec `#0` can be used as a synonym for `#none` - the nonexistent revision can be thought of as the one that "existed" before the first revision was submitted to the depot.
`file``#0`	
`file``#head`	The head revision (latest version) of *`file`*. Except where explicitly noted, this is equivalent to referring to the file without a revision specifier.
`file``#have`	The revision on the current client: the revision of file last **`p4 sync`**ed into the client workspace.
`file``@`*`n`*	Change number: The revision of *`file`* immediately after changelist *`n`* was submitted.
`file``@=`*`n`*	Change number: The revision of *`file`* at the specified changelist number *`n`*. *`n`* can be a submitted or a shelved change number; it can't be a pending (non-shelved) change number.

Modifier	Meaning
file`@`*labelname*	Label name: The revision of *file* in the label *labelname*.
file`@`*clientname*	Client name: The revision of *file* last taken into client workspace *clientname*. Note that deleted files (that is, files marked for `delete` at their latest revision) are not considered to be part of a workspace.
file`@`*datespec*	Date and time: The revision of *file* at the date and time specified. If no time is specified, the head revision at 00:00:00 on the morning of the date specified is returned. Dates are specified *yyyy*`/`*mm*`/`*dd*`:`*hh*`:`*mm*`:`*ss* or *yyyy*`/`*mm*`/`*dd hh*`:`*mm*`:`*ss* (with either a space or a colon between the date and the time). The datespec `@now` can be used as a synonym for the current date and time.

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

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

Files that have been shelved can also be accessed with the **p4 diff**, **p4 diff2**, **p4 files**, and **p4 print** commands, using the revision specifier `@=`*change*, where *change* is the pending changelist number.

Some 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.

  Revision ranges implicitly start at #1, for example, **p4 fixes //depot/file.c#5** implies all jobs fixed by revisions 1 through 5. (To see only those jobs that were fixed by revision 5, you would have to specify **p4 fixes //depot/file.c#5,5**)

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

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

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

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

Limitations on characters in filenames and entities

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

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

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

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

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

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

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

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

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

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

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

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

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

Views

Synopsis

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

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

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

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

"File Specifications" on page 575 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/...     //ws/...
//depot/dir/... //ws/dir2/...
```

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

```
//depot/dir/... //ws/dir2/...
//depot/...     //ws/...
```

then the file `//depot/dir/file.c` is mapped to `//ws/dir/file.c`, as the first mapping (mapping the file into `//ws/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/v1/... "//ws/version one/..."
```

Special characters in path and file names

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

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

Client Views

Client views are used to map files in the depot to files in client workspaces, and vice versa. A client workspace is an area in which users perform their work; files are synced to a client workspace, opened for editing, edited, and checked back into the depot.

When files are synced, they are copied from the depot to the locations in the client workspace to which they were mapped. Likewise, when files are submitted back into the depot, the mapping is reversed and the files are copied from the client workspace back to their proper locations in the depot.

The following table lists some examples of client views:

Client View	Sample Mapping
Full client workspace mapped to entire depot	`//depot/... //ws/...`
Full client workspace mapped to part of depot	`//depot/dir/... //ws/...`
Some files in the depot are excluded from the client workspace	`//depot/dir/... //ws/...` `-//depot/dir/exclude/... //ws/dir/exclude/...`

Client View	Sample Mapping
Some files in the depot are mapped to a different part of the client workspace	`//depot/...  //ws/...` `//depot/rel1/...  //ws/release1/...`
Files in the client workspace are mapped to different names than their depot names.	`//depot/dir/old.* //ws/renamed/new.*`
Portions of filenames in the depot are rearranged in the client workspace	`//depot/dir/%%1.%%2 //ws/dir/%%2.%%1`
The files do not map the same way in each direction. The second line takes precedence, and the first line is ignored.	`//depot/dir1/... //ws/build/...` `//depot/dir2/... //ws/build/...`
An overlay mapping is used to map files from more than one depot directory into the same place in the workspace.	`//depot/dir1/...  //ws/build/...` `+//depot/dir2/... //ws/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 view intending to use those files. You can do this by adding the newly-mapped branch of the depot to your current client view and performing a **p4 sync** command.

Label Views

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

Label View	Sample Mapping
A new release	`//depot/1.1final/...`
The source code for the new release	`//depot/1.1final/src/...`

A distribution suitable for clients	`//depot/1.1final/bin/...`
	`//depot/1.1final/doc/...`
	`//depot/1.1final/readme.txt`

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

File Types

Synopsis

Perforce supports six base file types:

- **text** files,
- compressed **binary** files,
- native **apple** files on Mac,
- Mac **resource** forks,
- symbolic links (**symlink**s), and
- **unicode** and **utf16** files.

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

When adding files, Perforce first examines the typemap table to see if the system administrator has defined a file type for the file(s) being added. If a match is found, the file's type is set as defined in the typemap table. If a match is *not* found, Perforce examines the first bytes of the file based on the **filesys.binaryscan** configurable (by default, 65536 bytes) to determine whether it is **text** or **binary**, and the files are stored in the depot accordingly.

By default, text file revisions are stored in reverse delta format; newly-added text files larger than the limit imposed by the **filetype.maxtextsize** configurable (by default, 10 MB) are assigned filetype **text +C** and stored in full. Files compressed in the **.zip** format (including **.jar** files) are also automatically detected and assigned the type **ubinary**. Other binary revisions are stored in full, with compression.

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

Perforce administrators can use the type mapping feature (**p4 typemap**) to override Perforce's default file type detection mechanism. This feature is useful for **binary** file formats (such as Adobe PDF, or Rich Text Format) where files can start with large portions of ASCII text, and might otherwise be mistaken for **text** files.

Perforce administrators can use the **filesys.binaryscan** and **filetype.maxtextsize** configurables (see **p4 configure**) to change the default limits of 65536 bytes for text/binary detection, and the 10 MB RCS text file size limit respectively.

Base filetypes

The base Perforce file types are:

Keyword	Description	Comments	Stored as
text	Text file	Synced as text in the workspace. Line-ending translations are performed automatically.	deltas in RCS format

Keyword	Description	Comments	Stored as
`binary`	Non-text file	Synced as binary files in the workspace. Stored compressed within the depot.	full file, compressed
`symlink`	Symbolic link	Perforce applications on UNIX, OS X, recent versions of Windows treat these files as symbolic links. On other platforms, these files appear as (small) text files. On Windows you require `admin` privileges or an appropriate group policy must be set; otherwise, you get text files.	deltas in RCS format
`apple`	Multi-forked Mac file	AppleSingle storage of Mac data fork, resource fork, file type and file creator. For full details, see the Mac client release notes.	full file, compressed, AppleSingle format.
`resource`	Mac 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, see the Mac client release notes.	full file, compressed
`unicode`	Unicode file	Perforce services operating in unicode mode support the `unicode` file type. These files are translated into the local character set specified by `P4CHARSET`. Perforce services not in unicode mode do not support the `unicode` file type. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format
`utf8`	Unicode file	Whether the service is in unicode mode or not, files are transferred as UTF-8 in the client workspace. For details, see the *Internationalization Notes*.	RCS deltas in UTF-8 format without BOM (byte order mark).
`utf16`	Unicode file	Whether the service is in unicode mode or not, files are transferred as UTF-8, and translated to UTF-16 (with byte order mark, in the byte order appropriate for the user's machine) in the client workspace.	RCS deltas in UTF-8 format

Keyword	Description	Comments	Stored as
		For details, see the *Internationalization Notes*.	

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. UTC keywords are better suited to describe events in globally distributed installations. Supported keywords are as follows: • `$Id$` • `$Header$` • `$Date$` Date of submission • `$DateUTC$` Date of submission in UTC time zone • `$DateTime$` Date and time of submission • `$DateTimeUTC$` Date and time of submission in UTC time zone. • `$DateTimeTZ$` Date and time of submission in the server's time zone, but including the actual time zone in the result.

Modifier	Description	Comments
		• `$Change$` • `$File$` • `$Revision$` • `$Author$`
`+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`	Perforce stores the full compressed version of each file revision	Default storage mechanism for **`binary`** files and newly-added **`text`**, **`unicode`**, and **`utf16`** files larger than 10MB.
`+D`	Perforce stores deltas in RCS format	Default storage mechanism for **`text`** files.
`+F`	Perforce stores full file per revision, uncompressed	Useful for large binaries, or for long ASCII files that are not read by users as text, such as PostScript files.
`+S`	Only the head revision is stored	Older revisions are purged from the depot upon submission of new revisions. Useful for executable or **`.obj`** files.
`+S`*`n`*	Only the most recent *`n`* revisions are stored, where *`n`* is a number from 1 to 10, or 16, 32, 64, 128, 256, or 512.	Older revisions are purged from the depot upon submission of more than *`n`* new revisions, or if you change an existing **`+S`***`n`* file's *`n`* to a number less than its current value. Earlier revisions unaffected; see "Usage Notes" on page 591 for details.
`+m`	Preserve original modtime	The file's timestamp on the local filesystem is preserved upon submission and restored upon sync. Useful for third-party DLLs in Windows environments.
`+X`	Archive trigger required	The Perforce service runs an **`archive`** trigger to access the file. See the *Helix Versioning Engine Administrator Guide: Fundamentals* for details.

A file's type is normally preserved between revisions, but can be overridden or changed with the **`-t`** option during **`add`**, **`edit`**, or **`reopen`** operations:

- **`p4 add -t filetype filespec`** adds the files as the specified type.

- **p4 edit -t filetype filespec** opens the file for **edit** as the specified type. The file's type is changed to the specified ***filetype*** only after it is submitted to the depot.

- **p4 reopen -t *filetype filespec*** changes the type of a file already open for **add** or **edit**.

The ***filetype*** argument is specified as **[*basetype*] *+modifiers***. For example, to change **script.sh**'s type to executable text with RCS keyword expansion, use **p4 edit -t text+kx script.sh**.

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

Perforce file types for common file extensions

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

File Type	Perforce file type	Description
.asp	text	Active Server Page file
.avi	binary+F	Video for Windows file
.bmp	binary	Windows bitmap file
.btr	binary	Btrieve database file
.cnf	text	Conference link file
.css	text	Cascading style sheet file
.doc	binary	Microsoft Word document
.dot	binary	Microsoft Word template
.exp	binary+w	Export file (Microsoft Visual C++)
.gif	binary+F	GIF graphic file
.gz	binary+F	Gzip compressed file
.htm	text	HTML file
.html	text	HTML file
.ico	binary	Icon file
.inc	text	Active Server Include file
.ini	text+w	Initial application settings file

File Type	Perforce file type	Description
`.jpg`	`binary`	JPEG graphic file
`.js`	`text`	JavaScript language source code file
`.lib`	`binary+w`	Library file (several programming languages)
`.log`	`text+w`	Log file
`.mpg`	`binary+F`	MPEG video file
`.pdf`	`binary`	Adobe PDF file
`.pdm`	`text+w`	Sybase Power Designer file
`.ppt`	`binary`	Microsoft PowerPoint file
`.xls`	`binary`	Microsoft Excel file

For more about mapping file names to Perforce filetypes, see the **`p4 typemap`** command.

Keyword Expansion

RCS keywords are expanded as follows:

Keyword	Expands To	Example
`$Id$`	File name and revision number in depot syntax.	`$Id: //depot/path/file.txt#3 $`
`$Header$`	Synonymous with `$Id$`.	`$Header: //depot/path/file.txt#3 $`
`$Date$`	Date of last submission in format *`YYYY/MM/DD`*	`$Date: 2010/08/18 $`
`$DateTime$`	Date and time of last submission in format *`YYYY/MM/DD hh:mm:ss`* Date and time are as of the local time on the Perforce service at time of submission.	`$DateTime: 2010/08/18 23:17:02 $`
`$Change$`	Perforce changelist number under which file was submitted.	`$Change: 439 $`
`$File$`	File name only, in depot syntax (without revision number).	`$File: //depot/path/file.txt $`
`$Revision$`	Perforce revision number.	`$Revision: #3 $`

Keyword	Expands To	Example
`$Author$`	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 **`+S`***`n`* (temporary object) modifier tells Perforce to store only the most recent ***`n`*** revisions of the file in the depot. If you change an existing file into a temporary object, subsequent revisions (after the ***`n`***th) will purge the revisions stored after the old head revision, but revisions to the file stored in the depot *before* the **`+S`***`n`* modifier was used will remain unaffected. (Syncing to a non-head revision submitted *after* the **`+S`***`n`* modifier was used will delete the file from your workspace. Such revisions are displayed as **`purge`** operations in the output of **`p4 filelog`**.)
- Running **`p4 integrate`** on temporary object files (**`+S`** and **`+S`***`n`*) does not produce a lazy copy; the integrated **`tempobj`** file consumes additional diskspace on the shared versioning service.
- The modtime (**`+m`**) modifier is a special case: It is intended for use by developers who need to preserve a file's original timestamp.

  If a client workspace uses the **`modtime`** option, the file date is not guaranteed to advance for each revision. For example, if a file is copy integrated ("accept theirs"), its timestamp will reflect that of the source file. If a user checks in a file with an old date, the client workspace file will reflect that same, old date. Normally, Perforce updates the timestamp when a file is synced; the modtime option enables a user to ensure that the timestamp of a file in a client workspace after a **`p4 sync`** will be the original timestamp existing *on the file* at the time of submission (that is, *not* the time at the Perforce versioning service at time of submission, and *not* the time on the user's workstation at the time of sync).

  The most common case where this is useful is development involving the third-party DLLs often encountered in Windows environments. Because the timestamps on such files are often used as proxies for versioning information (both within the development environment and also by the

operating system), it is sometimes necessary to preserve the files' original timestamps regardless of a Perforce user's client settings.

The `+m` modifier on a file allows this to happen; if set, Perforce will ignore the `modtime` ("file's timestamp at time of submission") or `nomodtime` ("date and time on the client at time of sync") option setting of the client workspace when syncing the file, and always restore the file's original timestamp at the time of submit.

- Versions of Perforce prior to 99.1 used a set of keywords to specify file types. The following table lists the older keywords and their current base file types and modifiers:

Old Keyword	Description	Base Filetype	Modifiers
`text`	Text file	`text`	none
`xtext`	Executable text file	`text`	`+x`
`ktext`	Text file with RCS keyword expansion	`text`	`+k`
`kxtext`	Executable text file with RCS keyword expansion	`text`	`+kx`
`binary`	Non-text file	`binary`	none
`xbinary`	Executable binary file	`binary`	`+x`
`ctext`	Compressed text file	`text`	`+C`
`cxtext`	Compressed executable text file	`text`	`+Cx`
`symlink`	Symbolic link	`symlink`	none
`resource`	Mac resource fork	`resource`	none
`uresource`	Uncompressed Mac 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`

Old Keyword	Description	Base Filetype	Modifiers
`xutf16`	Executable UTF-16	`utf16`	`+x`

Configurables

The following table lists all the configurables you can use to customize a Perforce service. Configurable settings might affect the server, the client, or a proxy. The target of a particular configurable is indicated in the table below. The next sections explain how you set configurables, depending on their target.

In the table of configurables that makes up most of this appendix, where a configurable refers to a number of bytes, "K" and "M" abbreviations are interpreted as the appropriate powers of two. For other configurables, "K" and "M" refer to 1,000 and 1,000,000.

Configurables that affect the server

Use the **p4 configure** to set or unset configurables that affect a Perforce server. These configurables are also described in **p4 help configurables**. For more information on the options you have in setting server configurables and on order of precedence, see the description of **p4 configure**.

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

Configurables that affect the client

You can set configurables that affect the client in the following ways (shown in order of precedence):

- As command line global options that are passed at server startup. For example:

```
$ p4 -u bluto -p perforce:1666 sync
```

- As entries in a P4CONFIG file. Set configurables like this:

```
P4USER=bluto
P4PORT=perforce:1666
```

  The following configurables can be set in a config file; you can also set the variables listed for the **p4 help environment** command:

filesys.binaryscan	filesys.bufsize	lbr.verify.out
net.keepalive.count	net.keepalive.disable	net.keepalive.idle
net.keepalive.interval	net.maxwait	net.net.rfc3483
net.tcpsize	sys.rename.max	sys.rename.wait

- As entries in a P4ENVIRO file.

  You can use both P4ENVIRO and P4CONFIG files to define environment variables: use the P4CONFIG file for those variables that have different values for different workspaces and the P4ENVIRO file for those

variables that remain constant for all projects. Values set in a `P4CONFIG` file override those set in a `P4ENVIRO` file.

- As set by the **`p4 set`** command for Windows and OS X. For example:

```
$ p4 set P4PORT=ssl:tea:1666
```

Configurables that affect the proxy

You can set configurables that affect the proxy in the following ways:

- Using a command line option. For example:

```
$ p4p -p tcp64:[::]:1999 -t central:1666 -r /var/proxyroot -v proxy.monitor.level=2
```

- Using environment variables.
- On Windows, using the **`p4 set`** command as follows:

```
C:\> p4 set -S "perforce_proxy" P4POPTIONS="-v myconfig=myvalue"
```

Configurables

Configurable	Client or Server or Proxy?	Default Value	Meaning
`auth.default.method`	Server	`perforce`	The default method to use for authenticating new users. • `perforce` specifies that the user is to be authenticated using Perforce's `db.user` table. This is the default setting. If there are no active LDAP configurations, this setting might cause a new user to be authenticated against an AD/LDAP server, using an authentication trigger if such a trigger exists. • `ldap` specifies the user be authenticated in against an AD/LDAP server without having to use authentication triggers.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			In addition, if you want new users to be automatically created when they have successfully authenticated against an AD/LDAP server, set the configurable `auth.ldap.userautocreate` to a non-zero value.
`auth.id`	Server	`0`	An alphanumeric identifier that must be set for all servers in a distributed configuration if you want to implement single login to the master, which is then valid across all replica instances. You must also set `rpl.forward.login` to one for each replica participating in the distributed configuration.
`auth.ldap.userautocreate`	Server	`0`	If `auth.default.method` is set to `ldap`, a value of `1` for this configurable will cause users to be auto-created when they log in to Perforce and they have been successfully authenticated against an AD/LDAP server using **`p4 login`**.
`auth.ldap.cafile`	Server	none	The path to a file that contains one or more PEM-formatted certificates used to verify the certificate presented by the AD/LDAP server when using SSL or TLS and `auth.ldap.ssllevel` is >=1.

`auth.ldap.order.`*`n`*	Server	none	Specifies the name of the LDAP configuration to use for authentication and the order in which it should be used to search for a given user name. The lowest number confers the highest priority.
			
			You may skip numbers. For example:
			
			`auth.ldap.order.1=UK_LDAP`
			`auth.ldap.order.2=US_LDAP`
			`auth.ldap.order.5=RU_LDAP`
			
			If this configurable has been set, if it specifies an existing LDAP configuration, if LDAP

Configurable	Client or Server or Proxy?	Default Value	Meaning
			authentication is enabled, and if the Perforce server is restarted, authentication trigger support is disabled.
`auth.ldap.pagesize`	Server	??	Specifies the paging limit in bytes for LDAP searches with paged results. Set the configurable to a value less than the LDAP server's result limit. For example, if the LDAP server's limit is 2.0 KB, set the configurable to 1.9 KB.
`auth.ldap.ssllevel`	Server	`0`	Level of SSL certificate validation: • `0`: No validation; default. • `1`: Certificate must be valid, but the common name is not checked. • `2`: Certificate must be valid and the certificate common name matches the AD/LDAP server's host name.
`auth.ldap.timeout`	Server	`30`	The time in seconds to wait before giving up on a connection attempt.
`client.readonly.dir`	Server	none	The path of the directory where db.* files for a read-only client will be placed. For example, if you create a read-only client whose name is `myroc` and `client.readonly.dir` is set to `/perforce/1`, then syncing files using this client will write to the database `/perforce/1/server.dbs/client/`*`hashdir`*`/db.myroc`
`cluster.id`	Server	none	Use `auth.id` instead of `cluster.id`.
`db.monitor.shared`	Server	256 K	The default value sets the maximum size of memory allotted to the db.monitor table, which tracks the p4d commands that are currently running. Setting this configurable to 0 means that the table is written to disk. However, writing the table to memory is recommended to improve performance.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			If the size of the table exceeds the value of `db.monitor.shared`, an error is returned, commands that do not fit in the table are still executed, but they are not recorded in the table.
`db.peeking`	Server	`2`	Enable and configure lockless reads; when enabled, many common commands no longer block other commands attempting to update the database. See the *Helix Versioning Engine Administrator Guide: Fundamentals* for details. `0`: Disable peeking. Behavior is identical to 2013.2 and earlier. `1`: New locking order is enabled, peeking is disabled, (diagnostic use only). `2`: New locking order is enabled, peeking is enabled, `hx`/`dx` optimization on. `3`: New locking order is enabled, peeking is enabled, `hx`/`dx` optimization is off.
`db.replication`	Server	unset	Control behavior of commands that access metadata (`db.*` files) on the Perforce server: `readonly`: User commands that read metadata are accepted; commands that modify metadata are rejected. Equivalent to starting a replica with the **`p4d -M readonly`** option. This configurable cannot be set globally; you must specify a server id.
`dbjournal.bufsize`	Server	`16K`	Buffer size for journal and checkpoint read/write operations.
`dbopen.nofsync`	Server	`0`	Set to `1` to disable `fsync()` call when server closes a `db.*` database file, and permit the OS to determine when to write the modified data.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`defaultChangeType`	Server	none	Default type for new changelists: either `public` or `restricted`. If unset, new changelists are `public`.
`dm.annotate.maxsize`	Server	`10M`	Maximum revision size for **`p4 annotate`**.
`dm.domain.accessforce`	Server	`3600`	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.
`dm.domain.accessupdate`	Server	`300`	Wait this many seconds before requesting a write lock to update an access time.
`dm.grep.maxrevs`	Server	`10K`	Maximum number of revisions that can be searched with **`p4 grep`**.
`dm.integ.engine`	Server	`3`	By default, use new integration engine with **`p4 integrate`**. (The **`p4 merge`** command always uses the v3 integration engine regardless of this setting.) Sites that wish to continue to use the old (2006.1) integration logic must set this configurable to 2 by running **`p4 configure set dm.integ.engine=2`**.
`dm.keys.hide`	Server	`0`	If set to `1` or `2`, **`p4 keys`** requires `admin` access. If set to `2`, **`p4 key`** requires `admin` access.
`dm.password.minlength`	Server	`8`	Default minimum password length for servers where `security` is set to a nonzero value.
`dm.protects.allow.admin`	Server	`0`	Allow Perforce administrators to use `-a`, `-g`, and `-u` with **`p4 protects`**. By default, only superusers can use these options.
`dm.proxy.protects`	Server	`1`	Determine (in accord with the use of IP addresses in the protections table) whether a user can access a server from a given IP address. By default, if a connection comes through an intermediary, the `proxy-` prefix is prepended to the client IP address.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			Set this variable to `0` if you do not want to have connections that come in through an intermediary to have the `proxy-` prefix. For more information, see the **`p4 protect`** command.
`dm.resolve.attribs`	Server	`1`	Enable resolve for attributes set with **`p4 attribute`**.
`dm.rotatelogwithinjnl`	Server	`1`	Set to `0` to disable log rotation after journal rotation. By default, when the journal is rotated, any structured logs are also rotated. Disabling this behavior can help when you're doing frequent journal rotations and you want the log rotated on a different schedule.
`dm.shelve.accessupdate`	Server	300 seconds	When a shelf is viewed or modified, update its access time if its last access time was longer than the limit specified by the value of `dm.shelve.accessupdate` Use the **`p4 -Ztag change -o`** command to display the access time for shelved files.
`dm.shelve.maxfiles`	Server	`10M`	Maximum number of files that can be shelved with **`p4 shelve`**.
`dm.shelve.maxsize`	Server	`0`	Maximum size of a file that can be shelved, or `0` for unlimited.
`dm.shelve.promote`	Server	`0`	Enable to make edge servers always promote shelved files to the commit server (rather than use the `-p` option). Generally, it is a bad idea to enable automatic promotion because it causes a lot of unnecessary file transfers for shelved files that are not meant to be shared.
`dm.user.accessforce`	Server	`3600`	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`dm.user.accessupdate`	Server	`300`	Wait this many seconds before requesting a write lock to update an access time.
`dm.user.loginattempts`	Server	`3`	Number of password attempts before delay. After the third failed login attempt, the user must wait 10 seconds.
`dm.user.noautocreate`	Server	`0`	Control behavior of automatic user creation. `0`: Create users as required. When executed by a nonexistent user, most Perforce commands cause a user to be created. An example of a command that does not create a user is **`p4 info`**. `1`: New users may only be created by running **`p4 user`**. `2`: New users may only be created by superusers running **`p4 user`**.
`dm.user.resetpassword`	Server	`0`	If set, all new users created with a password are forced to reset their password before issuing any commands. This configurable applies only if the passwords for newly created users are set using the `Password:` field of the user specification. The password reset behavior for new users that get initial passwords using the **`p4 passwd`** command after the user is created is not affected by the setting of this configurable.
`filesys.binaryscan`	Client	`64K`	Scan the first `filesys.binaryscan` bytes for binary data when running **`p4 add`**.
`filesys.bufsize`	Client, Server	`4K`	Buffer size for client-side read/write operations.
`filesys.depot.min`	Server	`250M`	Minimum diskspace required for any depot before server rejects commands. (If there is less than `filesys.depot.min` diskspace available for any one depot, commands are rejected for transactions involving all depots.)

Configurable	Client or Server or Proxy?	Default Value	Meaning
			You can use the following abbreviations to specify size: `t` or `T` for tebibytes `g` or `G` for gibibytes `m` or `M` for mebibytes `k` or `K` for kibibytes You can also use a percentage to specify the relative amount of free diskspace required.
`filesys.extendlowmark`	Client	`32K`	Minimum filesize before preallocation (Windows).
`filesys.P4JOURNAL.min`	Server	`250M`	Minimum diskspace required on server journal filesystem before server rejects commands. You can use the following abbreviations to specify size: `t` or `T` for tebibytes `g` or `G` for gibibytes `m` or `M` for mebibytes `k` or `K` for kibibytes You can also use a percentage to specify the relative amount of free diskspace required.
`filesys.P4LOG.min`	Server	`250M`	Minimum diskspace required on server log filesystem before server rejects commands. You can use the following abbreviations to specify size: `t` or `T` for tebibytes `g` or `G` for gibibytes `m` or `M` for mebibytes `k` or `K` for kibibytes You can also use a percentage to specify the relative amount of free diskspace required.
`filesys.P4ROOT.min`	Server	`250M`	Minimum diskspace required on server root filesystem before server rejects commands.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			You can use the following abbreviations to specify size: `t` or `T` for tebibytes `g` or `G` for gibibytes `m` or `M` for mebibytes `k` or `K` for kibibytes You can also use a percentage to specify the relative amount of free diskspace required.
`filesys.TEMP.min`	Server	`250M`	Minimum diskspace required for temporary operations before server rejects commands. You can use the following abbreviations to specify size: `t` or `T` for tebibytes `g` or `G` for gibibytes `m` or `M` for mebibytes `k` or `K` for kibibytes You can also use a percentage to specify the relative amount of free diskspace required.
`filesys.windows.lfn`	Server, Client, Proxy	`1`	Set to `1` to support filenames longer than 260 characters on Windows platforms. Depending on the depth of your workspace path, this might need to be set on the client, server, and/or proxy (which acts as a client). A file name length of up to 32,767 characters is allowed. Each component of the path is limited to 255 characters. The server root or client root cannot be a long path.
`filetype.maxtextsize`	Server	`10M`	Maximum file size for text type detection.
`journalPrefix`	Server	unset	Prefix or directory location for rotated journal.
`lbr.autocompress`	Server	`0`	Enabling this configurable, specifies the storage method as compressed text (`ctext`) rather than RCS format text. The user still sees the file type as `text`.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			It's a good idea to set this variable when using a commit/edge configuration or when sharing archive files between servers.
`lbr.bufsize`	Server, Proxy	`4K`	Buffer size for read/write operations to server's archive of versioned files.
`lbr.proxy.case`	Proxy	`1`	`1`: File paths are always case-insensitive. `2`: File paths are case-insensitive if server is case-insensitive. `3`: File paths are always case-sensitive.
`lbr.replication`	Server	unset	Control behavior of user commands that access versioned files on the Perforce server: `readonly`: User commands that read depot files are accepted; user commands that modify files are rejected. `shared`: This is a synonym for `ondemand` mode. `ondemand`: Replicate versioned files if, and only if, explicitly requested by users of the replica server. `cache`: Commands that reference file content are accepted, but do not automatically transfer files. You may now use this option with edge servers or build servers to enable parallel file transfers. `none`: No access to versioned files is permitted. This configurable cannot be set globally; you must specify a server id. Equivalent to starting a replica **`p4d`** process with one of the `-D readonly`, `-D shared`, (or `-D ondemand`), `-D cache`, or `-D none` options.
`lbr.retry.max`	Server	`50K`	In the event of a failed transfer, a replica will make `lbr.retry.max` attempts to retrieve the file.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`lbr.verify.in`	Server	`1`	Verify contents from the client to server? (`1` for yes, `0` for no)
`lbr.verify.out`	Client, Server	`1`	Verify contents from the server to client? (`1` for yes, `0` for no)
`lbr.verify.script.out`	Server	`1`	Set to `0` to prevent files of type `+X` from having their digest checked when transmitted from server to client. When source watermarking is used, sites have configured a `+X` archive trigger script that returns different results each time a file is sync'd or printed, in order to embed a user-specific string into the file contents during sync. This defeats the digest verification performed when sending the file to disk. Setting `lbr.verify.script.out` disables digest verification in this situation. Other files are still verified normally, as determined by the setting of `lbr.verify.out`.
`minClient`	Server	none	Lowest version of client software permitted to connect to this server, set by **`p4 configure set minClient=`*`version`***.
`minClientMessage`	Server	none	Message to issue if client software is too old, set by **`p4 configure set minClientMessage=`*`message`***.
`monitor`	Server	`0`	Server process monitoring: `0`: Server process monitoring off. `1`: Monitor active commands only. `2`: Monitor both active commands and idle connections. `5`: Monitor both active commands and idle connections, including a list of the files locked by the command for more than one second. `10`: Monitor both active commands and idle connections, including a list of the files locked

Configurable	Client or Server or Proxy?	Default Value	Meaning
			by the command for more than one second, with lock wait times included in the lock information. `25`: Monitor both active commands and idle connections, including a list of the files locked by the command for any duration, with lock wait times included in the lock information. See **p4 monitor** for details.

`monitor.lsof`	Server	none	When set on Unix platforms, enables the use of the **p4 monitor** command to display a list of locked files. Set to the following value:
			
			`$ path/lsof -F pln+`
			
			The value for *path* varies with the version of Unix you are using. For example:
			
			`$ /usr/bin/lsof -F pln`
			
			For more information, see the **p4 monitor** command.

`net.backlog`	Server, Proxy	`128`	Maximum length of queue for pending connections. Consider increasing if users find themselves unable to connect to extremely heavily-loaded servers.
`net.keepalive.count`	Server	`0`	Number of unacknowledged keepalives before failure.
`net.keepalive.disable`	Server	`0`	If non-zero, disable the sending of TCP keepalive packets.
`net.keepalive.idle`	Server	`0`	Idle time (in seconds) before starting to send keepalives.
`net.keepalive.interval`	Server	`0`	Interval (in seconds) between sending keepalive packets.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`net.maxfaultpub`	Proxy	`100`	A value in megabytes that controls the proxy's cache faulting behavior. A single **`p4 sync`** will not publish more than `net.maxfaultpub` megabytes of faults into `pdb.lbr`. You must restart the server after changing the value of this configurable.
`net.maxwait`	Client, Server, Proxy	none	Time, in seconds, before a network connection times out. Best practice is *not* to set server-wide: if set on server, requires that users complete command-line forms within this limit. If set in user's individual `P4CONFIG` file, applies to user's workstation (and requires only that the versioning service reply to user requests within the allotted time limit). You must restart the server after changing the value of this configurable.
`net.mimcheck`	Server, Proxy	`1`	Man-in-the-middle network security level: This enables checks for possible interception or modification of data such as using an SSH tunnel or other TCP forwarder for users with passwords set. `0`: Disable MitM checks. `1`: Check proxy/broker connections in legacy contexts. `2`: Connections from clients are checked for TCP forwarding. `3`: Connections from clients, proxies, and brokers are checked for TCP forwarding. `4`: All connections are checked; client software older than release 2010.1 cannot connect. `5`: Requires that proxies, brokers, and all Perforce intermediate servers have valid logged-in service users associated with

Configurable	Client or Server or Proxy?	Default Value	Meaning
			them. This allows administrators to prevent unauthorized proxies and services from being used. You must restart the server after changing the value of this configurable.
`net.parallel.batch`	Server	`8`	Specifies the number of files in a batch.
`net.parallel.batchsize`	Server	`512K`	Specifies the number of bytes in a batch.
`net.parallel.max`	Server	`0`	A value greater than one enables parallel processing up to the specified number of threads, when syncing a client or submitting files. In addition to setting this variable, you must use the `--parallel` option to the **`p4 sync`** command or the **`p4 submit`** command to further describe the processing desired. If you use `net.parallel.submit.*` configurables to automate parallel processing, you do not need to use the `--parallel` option. Values can range between `0` and `100`. See the **`p4 sync`** command or the **`p4 submit`** command for more information.
`net.parallel.min`	Server	`9`	Specifies the minimum number of files in a parallel sync. A sync that is too small does not initiate parallel file transfers.
`net.parallel.minsize`	Server	`576K`	Specifies the minimum number of bytes in a parallel sync. A sync that is too small does not initiate parallel file transfers.
`net.parallel.submit.threads`	Server	none	For automatic parallel processing: specifies the number of threads to be used for sending files in parallel. The specified threads grab work in batches. The size of a batch is specified using the `net.parallel.submit.batch`configurable. See the **`p4 submit`** command for more information.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`net.parallel.submit.min`	Server	`9`	For automatic parallel processing: specifies the minimum number of files to be sent in a parallel submit. See the **`p4 submit`** command for more information.
`net.parallel.submit.batch`	Server	`8`	For automatic parallel processing: specifies the number of files in a batch. See the **`p4 submit`** command for more information.
`net.parallel.threads`	Server	`none`	Specifies the number of independent network connections that can be used for syncing files concurrently. The specified threads grab work in batches.
`net.reuseport`	Server	`0`	Set `SO_REUSEPORT` for listening socket. You must restart the server after changing the value of this configurable.
`net.rfc3484`	Client, Server	`0`	If `1`, permit the operating system to determine whether IPv4 or IPv6 is used when resolving hostnames. This is applicable only if a host name (either FQDN or unqualified is used). If an IPv4 literal address (e.g. 127.0.0.1) is used, the transport is always `tcp4`, and if an IPv6 literal address (e.g. `::1`) is used, then the transport is always `tcp6`. You must restart the server after changing the value of this configurable.
`net.tcpsize`	Client, Server, Proxy	`512K`	TCP send and receive buffer sizes, set on connection. Consider increasing for high-latency connections, such as the Proxy. Actual buffer size is the larger of this value and that defined by the OS. You must restart the server after changing the value of this configurable.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`proxy.monitor.interval`	Proxy	`10`	Set the proxy monitoring interval. Default is 10 seconds.
`proxy.monitor.level`	Proxy	`0`	`0`: Monitoring disabled (default). `1`: Monitor file transfers only. `2`: Monitor all operations. `3`: Monitor all traffic for all operations.
`rcs.nofsync`	Server	`0`	Set to `1` to disable `fsync()` call when server writes to a versioned file in RCS format, and permit the OS to determine when to write the modified data.
`rejectList`	Server	0	Specifies one or more clients whose requests should be blocked. For more information, see "Blocking Clients" in *Helix Versioning Engine Administrator Guide: Fundamentals*.
`rpl.checksum.auto`	Server	`0`	Level of database table checksum verification to perform when rotating journal. Each level corresponds to a larger set of database tables. `0`: Disable checksums. `1`: Verify the most important system and revision tables. `2`: Verify all of level 1, plus tables that hold metadata that does not vary between replicas. `3`: Verify all metadata, including metadata that is expected to vary on build-farm and edge-server replicas.
`rpl.checksum.change`	Server	`0`	Level of on-the-fly changelist verification to perform. `0`: Perform no verification. `1`: Write journal note at the end of a submit.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			2: Replica verifies changelist summary and writes to **integrity.csv** if the changelist does not match. **3**: Replica verifies changelist summary and writes to **integrity.csv** even if the changelist does match. Setting affects **p4 submit**, **p4 push**, **p4 fetch**, **p4 populate**, and **p4 unzip** commands.
rpl.checksum.table	Server	0	Level of table checksumming to perform. **0**: Perform table-level checksumming only. **1**: Journal notes for table-unload and table-scan are processed by the replica, and are logged to **integrity.csv** if the check fails. **2**: Results of journal note processing in the replica are logged even if the results match.
rpl.compress	Server	0	Enable replica/master network compression: **0**: No data stream compression. **1**: Data streams used for archive transfer to the replica (**p4 pull -u**) are compressed. **2**: Data streams used by **p4 pull -u** and **p4 pull** are compressed. **3**: All data streams (**p4 pull -u**, **p4 pull**, and data streams for commands forwarded to the master or commit server) are compressed. **4**: Compress only the journal pull and journal copy connections between the replica and the master.
rpl.forward.login	Server	0	Set to **1** on each replica to enable single-sign-on authentication for users in a distributed configuration. The **auth.id** configurable must also be the same for all servers participating in a distributed configuration.

Configurable	Client or Server or Proxy?	Default Value	Meaning
			For more information, see "Authenticating users" in *Helix Versioning Engine Administrator Guide: Multi-site Deployment*.
`rpl.jnlwait.adjust`	Server	`25`	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.
`rpl.jnlwait.interval`	Server	`50`	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.
`rpl.jnlwait.max`	Server	`1000`	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.
`rpl.journal.ack`	Server	`1`	The number of standby servers that must acknowledge a persisted transaction before the transaction is visible to a workspace server and before the client is notified of successful completion. The default value means one standbys has to acknowledge a transaction. Do not set this configurable to a higher value than the number of available standby servers.
`rpl.journal.ack.min`	Server	`0`	Set to `0` means that if the standby stops running, transactions will continue to be processed without requiring acknowledgments. If then the master fails, you might lose data. Set to `1` means that if the standby stops running, update transactions will not complete and therefore, if the master then fails, no data from completed transactions are lost.
`rpl.labels.global`	Server	`0`	With a distributed Perforce service, there are both local and global labels. Local labels are restricted to a single edge server, and cannot

Configurable	Client or Server or Proxy?	Default Value	Meaning
			be used on other servers. Global labels are created and updated on the commit server, and are visible to all servers. However, global labels can only be used with global (unbound) client workspaces. For the 2013.2 release, the default is for labels to be local. Set `rpl.labels.global` to `1` to make labels global by default. When this configurable is set to `0`, users can use the `-g` option with the commands **p4 label**, **p4 labelsync**, and **p4 tag** to create or update global labels. When this configurable is set to `1`, the meaning of the `-g` option is inverted to allow updating of local labels.
`rpl.replay.userrp`	Server	`0`	Set to one to replicate the `db.user.rp` table. The `db.user.rp` table contains information about users who have directly logged into and used a replica. It records the ticket that the replica issued to the user and the last time the user accessed the replica. This table is currently journaled, but it is not replicated by default. Typically you would not want to replicate this data. But there are times when this might be needed. For example, if you are chaining a read-only replica to another replica to provide a warm standby for failover, you might want your warm standby to replicate the `db.user.rp` table. This would disable the automatic filtering of `db.user.rp` records, and your replica would then replay (and re-journal) all the `db.user.rp` journal records it receives from its target.
`rpl.verify.cache`	Server	`0`	If set, a replica server will re-verify the integrity of a cached file every time it delivers the file to the user, If the files do not match, it will re-fetch the file from the upstream server. This is computationally expensive

Configurable	Client or Server or Proxy?	Default Value	Meaning
			on the replica and typically only useful in conjunction with Perforce technical support.
`run.users.authorize`	Server	`0`	If set, requires a user to authenticate before running **`p4 users`**.
`security`	Server	`0`	Server security level: `0`: Legacy support: passwords not required, strength requirements unenforced. `1`: Strong passwords required, existing passwords not reset, compatible with pre-2003.2 client software. `2`: Strong passwords required, existing passwords reset, requires 2003.2 or higher client software. `3`: Passwords must be strong, and ticket-based authentication (**`p4 login`**) is required. `4`: All of the above restrictions. Also, authenticated service users must be used for all replica server and remote depot connections to this server.
`server.allowfetch`	Server	`0`	Determines whether changes can be fetched. • If set to `1`, this server can fetch from other servers. • If set to `2`, other servers can fetch from this server. • If set to `3`, both `1` and `2` are allowed.
`server.allowpush`	Server	`0`	Determines whether changes can be pushed. • If set to `1`, this server can push to other servers. • If set to `2`, other servers can push to this server. • If set to `3`, both `1` and `2` are allowed.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`server.allowrewrite`	Server	`0`	If set to a non-zero value, allows this server to run the **`p4 unsubmit`** and **`p4 fetch -u`** commands.
`server.commandlimits`	Server	`0`	Policy for per-command resource limits: `0`: All users may use command-line overrides for `MaxResults`, `MaxScanRows`, and `MaxLockTime` limits defined in the **`p4 group`** specs. `1`: Per-command options may specify lower, but not higher, resource limits. `2`: All command-line resource limit options are silently ignored.
`server.depot.root`	Server	none	The filesystem location with respect to which a relative address given in the `Map:` field of a depot form is evaluated. If it is not set, the `Map:` field relative address is evaluated with respect to the value stored in `P4ROOT`. For more information, see the **`p4 depot`** command.
`server.global.client.views`	Server	0	Controls whether the view maps of a non-stream client on an edge server are made global when a client is modified. View maps of a client on a replica must be made global if that client is to be used as a template on another server. This configurable can be set globally or individually for each server. This allows client maps to be global on most edge servers while keeping them local on those edge servers that don't need or want them to be global. Setting this configurable does not immediately make any client view maps global; that happens only when a client is modified afterwards. Clearing this configurable does not delete the view maps of any clients, but it does prevent subsequent changes to a client's view map from being propagated to other servers. If a client with global view

Configurable	Client or Server or Proxy?	Default Value	Meaning
			maps is deleted, its view maps are also deleted globally regardless of the value of `server.global.client.views`; this is to prevent orphaned view maps. Setting this configurable does not require a server restart.
`server.locks.archive`	Server	`0`	Set this configurable to disable server locks when running the **`p4 archive`** or **`p4 restore`** commands.
`server.locks.dir`	Server	`server.l`	Directory for server locks, specified relative to P4ROOT. To disable server locking, set this configurable to `disabled`. (If `db.peeking` is nonzero (enabled), `server.locks` cannot be `disabled`; you can disable locking by setting `server.locks.sync` to 0.)
`server.locks.global`	Server	`0`	Set this configurable to `1` to make **`p4 lock`** from an edge server take global locks on the commit server by default.
`server.locks.sync`	Server	`0`	When set, the **`p4 sync`** command takes a client workspace lock in shared mode. The default value of `0` prevents sync from taking a client workspace lock. If `db.peeking` is enabled, the `server.locks.dir` directory must exist. The changes to locking behavior that occur when you enable `db.peeking` obviate the need to set `server.locks.dir` to `disabled`, but if performance issues arise with respect to multiple concurrent, large, and/or interrupted **`p4 sync`** commands, you can obtain the old behavior for syncing by setting `server.locks.sync` to `0`.
`server.maxcommands`	Server	`0`	If monitoring is enabled, and if this configurable is set to a nonzero value, the service refuses to accept more than this many simultaneous command requests.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`serverlog.counter.n`	Server	none	The counter name for the structured log file designated by `n`. (For example, if the structured log file is `errors.csv`, `n` is `3`.) See "Logging and structured files" in the *Helix Versioning Engine Administrator Guide: Fundamentals* for more information.
`serverlog.file.n`	Server	none	Server log file name associated with each structured log file. See **`p4 logparse`** for a list of valid filenames. *n* may not exceed 500.
`serverlog.maxmb.n`	Server	none	For each structured log file, the size, in megabytes, at which the associated log file is rotated.
`serverlog.retain.n`	Server	none	For each structured log file, the number of rotated log files to retain on the server at any one time.
`serviceUser`	Server	none	The service user as which a server (or proxy) authenticates against a master server in a replication/proxy configuration, or against a remote server in the context of remote depots.
`spec.hashbuckets`	Server	`99`	Number of buckets (subdirectories) into which files in the spec depot are hashed. Set to `0` to disable hashing, which may slow performance on older filesystems where performance is a function of the number of files per directory.
`ssl.secondary.suite`	Server	`0`	By default, Perforce's SSL support is based on the AES256-SHA cipher suite. To use CAMELLIA256-SHA, set this tunable to `1`.
`startup.n`	Server	none	For replica servers, set `startup.1` through `startup.n` to be **`p4 pull`** threads to be spawned at startup. The `startup.n` configurables are processed sequentially. Processing stops at the first gap

Configurable	Client or Server or Proxy?	Default Value	Meaning
			in the numerical sequence; any commands after a gap are ignored.
`statefile`	Server	`state`	For replica servers, the file used by the server to track the current journal position.
`submit.identity`	Server	none	Enable the generation of global changelist ids. This is relevant for users of the Helix Versioning Engine's distributed versioning (DVCS) features. • `uuid` generates the id in uuid format. • `checksum` generates the id in checksum format. • `serverid` generates the id in serverid +change format. For more information on global changelist ids, see the section "Track a changelist's identity from server to server" in the "Fetching and Pushing" chapter of *Using Helix for Distributed Versioning*.
`submit.noretransfer`	Server	`0`	Always re-transfer files after a failed submit. Set this configurable to have the server check whether files are already in the expected archive location and to not re-transfer files when retrying a failed submit. You can override the set behavior by using the `--noretransfer` option to the **`p4 submit`** command.
`submit.unlocklocked`	Server	`0`	When set, open files that users have locked (with the **`p4 lock`** command) are automatically unlocked after a failed **`p4 submit`**.
`sys.rename.max`	Server	`10`	Limit in microseconds for retrying a failed file rename. Affects Windows `Rename()` retry loop.

Configurable	Client or Server or Proxy?	Default Value	Meaning
`sys.rename.wait`	Server	`1000`	Timeout in microseconds between file rename attempts. Affects Windows `Rename()` retry loop.
`template.client`	Server	none	Specifies the default client to be used as a template if the user omits the `-t` option on the **`p4 client`** command.
`template.label`	Server	none	Specifies the default label to be used as a template if the user omits the `-t` option on the **`p4 label`** command.
`triggers.io`	Server	`0`	If set, specifies that triggers will not receive their parameters via command line variables. Rather, they will receive a dictionary of key/value pairs sent to their STDIN. Triggers can use their dictionary response to reply to the server via STDOUT.
`zerosyncPrefix`	Server	none	If set, changes default behavior of **`p4 sync`** such that if a client workspace begins with this prefix, all sync operations to affected workspaces assume **`p4 sync`** **`-k`**, and do not alter contents of the workspace.

License Statements

Perforce software includes software developed by the University of California, Berkeley and its contributors. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

Perforce software includes software from the Apache ZooKeeper project, developed by the Apache Software Foundation and its contributors. (http://zookeeper.apache.org/)

Perforce software includes software developed by the OpenLDAP Foundation (http://www.openldap.org/).

Perforce software includes software developed Computing Services at Carnegie Mellon University: Cyrus SASL (http://www.cmu.edu/computing/).