

Perforce 2014.1: Command Reference

March 2014

Perforce 2014.1: Command Reference

March 2014

Copyright © 1999-2014 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 497](#).

Table of Contents

About This Manual	xxxv
Introduction	1
Getting help	1
Creating scripts	1
What's New in 2014.1	2
Major Changes	2
New Commands	2
Minor Changes	3
Improving Performance	4
Support for Distributed Configuration	4
p4 add	7
Synopsis	7
Syntax	7
Description	7
Options	8
Usage Notes	8
Examples	9
Related Commands	9
p4 admin	11
Synopsis	11
Syntax	11
Description	11
Options	12
Usage Notes	12
Examples	13
Related Commands	13
p4 annotate	15
Synopsis	15
Syntax	15
Description	15
Options	15
Usage Notes	16
Examples	17
p4 archive	19

Synopsis	19
Syntax	19
Description	19
Options	19
Usage Notes	19
Examples	20
Related Commands	20
 p4 attribute	 21
Synopsis	21
Syntax	21
Description	21
Options	21
Usage Notes	21
 p4 branch	 23
Synopsis	23
Syntax	23
Description	23
Form Fields	23
Options	24
Usage Notes	24
Related Commands	25
 p4 branches	 27
Synopsis	27
Syntax	27
Description	27
Options	27
Usage Notes	27
Related Commands	27
 p4 cachepurge	 29
Synopsis	29
Syntax	29
Description	29
Options	29
Usage Notes	30
 p4 change	 31
Synopsis	31
Syntax	31
Description	31

Form Fields	31
Options	33
Usage Notes	34
Examples	34
Related Commands	34
p4 changelist	37
Synopsis	37
Syntax	37
Description	37
p4 changelists	39
Synopsis	39
Syntax	39
Description	39
p4 changes	41
Synopsis	41
Syntax	41
Description	41
Options	42
Usage Notes	42
Examples	42
Related Commands	43
p4 clean	45
Synopsis	45
Syntax	45
Description	45
Options	45
Usage Notes	46
Related Commands	46
p4 client	47
Synopsis	47
Syntax	47
Description	47
Form Fields	47
Options	50
Usage Notes	51
Options field	52
Processing line endings	53
Working with Streams	54

Working with build servers	54
Examples	55
Related Commands	55
p4 clients	57
Synopsis	57
Syntax	57
Description	57
Options	57
Usage Notes	58
Related Commands	58
p4 configure	59
Synopsis	59
Syntax	59
Description	59
Options	72
Usage Notes	73
Related Commands	73
p4 copy	75
Synopsis	75
Syntax	75
Description	75
Options	75
Usage Notes	76
Examples	76
Related Commands	77
p4 counter	79
Synopsis	79
Syntax	79
Description	79
Options	79
Usage Notes	80
Examples	80
Related Commands	81
p4 counters	83
Synopsis	83
Syntax	83
Description	83
Options	83

Usage Notes	83
Related Commands	83
 p4 cstat	85
Synopsis	85
Syntax	85
Description	85
Options	85
Usage Notes	85
Related Commands	85
 p4 dbschema	87
Synopsis	87
Syntax	87
Description	87
Options	87
Usage Notes	87
Examples	87
 p4 dbstat	89
Synopsis	89
Syntax	89
Description	89
Options	89
Usage Notes	89
 p4 dbverify	91
Synopsis	91
Syntax	91
Description	91
Options	91
Usage Notes	91
 p4 delete	93
Synopsis	93
Syntax	93
Description	93
Options	93
Usage Notes	94
Examples	94
Related Commands	94

p4 depot	95
Synopsis	95
Syntax	95
Description	95
Form Fields	95
Options	96
Usage Notes	97
Providing Map Information	97
Working with Remote Depots	97
Working with Spec Depots	98
Related Commands	98
 p4 depots	 101
Synopsis	101
Syntax	101
Description	101
Options	101
Usage Notes	101
Related Commands	101
 p4 describe	 103
Synopsis	103
Syntax	103
Description	103
Options	103
Usage Notes	104
Related Commands	104
 p4 diff	 105
Synopsis	105
Syntax	105
Description	105
Options	105
Usage Notes	106
Examples	107
Related Commands	107
 p4 diff2	 109
Synopsis	109
Syntax	109
Description	109
Options	110

Usage Notes	110
Examples	111
Related Commands	112
 p4 dirs	113
Synopsis	113
Syntax	113
Description	113
Options	113
Usage Notes	113
Examples	114
Related Commands	114
 p4 diskspace	115
Synopsis	115
Syntax	115
Description	115
Options	115
Usage Notes	115
Related Commands	116
 p4 edit	117
Synopsis	117
Syntax	117
Description	117
Options	117
Usage Notes	118
Examples	118
Related Commands	119
 p4 export	121
Synopsis	121
Syntax	121
Description	121
Options	121
Usage Notes	122
Examples	122
Related Commands	122
 p4 filelog	123
Synopsis	123
Syntax	123
Description	123

Options	124
Usage Notes	124
Examples	125
Related Commands	125
 p4 files	 127
Synopsis	127
Syntax	127
Description	127
Options	127
Usage Notes	128
Examples	128
Related Commands	129
 p4 fix	 131
Synopsis	131
Syntax	131
Description	131
Options	131
Usage Notes	132
Examples	132
Related Commands	133
 p4 fixes	 135
Synopsis	135
Syntax	135
Description	135
Options	135
Usage Notes	136
Examples	136
Related Commands	136
 p4 flush	 137
Synopsis	137
Syntax	137
Warning	137
Description	137
Options	138
Usage Notes	138
Examples	138
Related Commands	139
 p4 fstat	 141

Synopsis	141
Syntax	141
Description	141
Form Fields	141
Options	144
Usage Notes	146
Filters	146
Other Usage Notes	147
Examples	147
Related Commands	148
 p4 grep	 149
Synopsis	149
Syntax	149
Description	149
Options	149
Usage Notes	150
 p4 group	 151
Synopsis	151
Syntax	151
Description	151
Form Fields	151
Options	152
Usage Notes	153
Related Commands	154
 p4 groups	 155
Synopsis	155
Syntax	155
Description	155
Options	155
Usage Notes	156
Examples	156
Related Commands	156
 p4 have	 157
Synopsis	157
Syntax	157
Description	157
Options	157
Usage Notes	157
Examples	158
Related Commands	158

p4 help	159
Synopsis	159
Syntax	159
Description	159
Usage Notes	160
Related Commands	160
 p4 info	 161
Synopsis	161
Syntax	161
Description	161
Options	161
Usage Notes	161
Related Commands	162
 p4 integrate	 163
Synopsis	163
Syntax	163
Description	163
Options	164
Basic Integration Flags	164
Advanced Integration Flags	165
Usage Notes	166
Examples	167
Related Commands	167
 p4 integrated	 169
Synopsis	169
Syntax	169
Description	169
Options	170
Usage Notes	170
Related Commands	170
 p4 interchanges	 173
Synopsis	173
Syntax	173
Description	173
Options	173
Usage Notes	174

p4 istat	175
Synopsis	175
Syntax	175
Description	175
Options	175
Usage Notes	175
Related Commands	175
 p4 job	 177
Synopsis	177
Syntax	177
Description	177
Form Fields	177
Options	178
Usage Notes	178
Examples	178
Related Commands	179
 p4 jobs	 181
Synopsis	181
Syntax	181
Description	181
Options	182
Usage Notes	182
Job Views	182
Other Usage Notes	183
Limitations	184
Examples	184
Related Commands	185
 p4 jobspec	 187
Synopsis	187
Syntax	187
Description	187
Form Fields	187
Options	189
Usage Notes	189
Related Commands	190
 p4 journaldbchecksums	 191
Synopsis	191
Syntax	191

Description	191
Options	191
Usage Notes	192
p4 journals	193
Synopsis	193
Syntax	193
Description	193
Options	195
Usage Notes	196
Related Commands	196
p4 key	197
Synopsis	197
Syntax	197
Description	197
Options	197
Usage Notes	198
Examples	198
Related Commands	198
p4 keys	199
Synopsis	199
Syntax	199
Description	199
Options	199
Usage Notes	199
Related Commands	199
p4 label	201
Synopsis	201
Syntax	201
Description	201
Form Fields	201
Options	202
Usage Notes	203
Examples	203
Related Commands	203
p4 labels	205
Synopsis	205
Syntax	205
Description	205

Options	205
Usage Notes	206
Examples	206
Related Commands	206
 p4 labelsync	207
Synopsis	207
Syntax	207
Description	207
Options	207
Usage Notes	208
Related Commands	208
 p4 license	209
Synopsis	209
Syntax	209
Description	209
Options	209
Usage Notes	209
Examples	210
 p4 list	211
Synopsis	211
Syntax	211
Description	211
Options	211
Usage Notes	211
 p4 lock	213
Synopsis	213
Syntax	213
Description	213
Options	213
Usage Notes	213
Related Commands	213
 p4 lockstat	215
Synopsis	215
Syntax	215
Description	215
Options	215
Usage Notes	215

p4 logappend	217
Synopsis	217
Syntax	217
Description	217
Options	217
Usage Notes	217
 p4 logger	 219
Synopsis	219
Syntax	219
Description	219
Options	219
Usage Notes	219
Related Commands	219
 p4 login	 221
Synopsis	221
Syntax	221
Description	221
Options	221
Usage Notes	221
Examples	222
Related Commands	222
 p4 logout	 223
Synopsis	223
Syntax	223
Description	223
Options	223
Usage Notes	223
Examples	223
Related Commands	224
 p4 logparse	 225
Synopsis	225
Syntax	225
Description	225
Options	225
Usage Notes	226

p4 logrotate	227
Synopsis	227
Syntax	227
Description	227
Options	227
Usage Notes	227
 p4 logschema	 229
Synopsis	229
Syntax	229
Description	229
Options	229
Usage Notes	229
 p4 logstat	 231
Synopsis	231
Syntax	231
Description	231
Options	231
Usage Notes	231
 p4 logtail	 233
Synopsis	233
Syntax	233
Description	233
Options	233
Usage Notes	233
Examples	233
 p4 merge	 235
Synopsis	235
Syntax	235
Description	235
Options	235
Usage Notes	236
Examples	236
Related Commands	237
 p4 monitor	 239
Synopsis	239

Syntax	239
Description	239
Options	240
Usage Notes	240
Examples	241
Related Commands	241
 p4 move	 243
Synopsis	243
Syntax	243
Description	243
Options	243
Usage Notes	243
Examples	244
 p4 obliterate	 245
Synopsis	245
Syntax	245
Warning	245
Description	245
Options	245
Usage Notes	246
Examples	246
Related Commands	247
 p4 opened	 249
Synopsis	249
Syntax	249
Description	249
Options	249
Usage Notes	250
Examples	250
Related Commands	251
 p4 passwd	 253
Synopsis	253
Syntax	253
Description	253
Options	254
Usage Notes	254
Related Commands	255
 p4 ping	 257

Synopsis	257
Syntax	257
Description	257
Options	257
Usage Notes	257
p4 populate	259
Synopsis	259
Syntax	259
Description	259
Options	260
Usage Notes	260
p4 print	261
Synopsis	261
Syntax	261
Description	261
Options	261
Usage Notes	262
Related Commands	262
p4 property	263
Synopsis	263
Syntax	263
Description	263
Options	263
Usage Notes	264
p4 protect	265
Synopsis	265
Syntax	265
Description	265
Form Fields	266
Options	268
Usage Notes	268
Examples	275
Related Commands	276
p4 protects	277
Synopsis	277
Syntax	277
Description	277
Options	277

Usage Notes	277
Related Commands	278
 p4 proxy	 279
Synopsis	279
Syntax	279
Description	279
Options	279
Usage Notes	279
Related Commands	279
 p4 pull	 281
Synopsis	281
Syntax	281
Description	281
Options	282
Usage Notes	283
Related Commands	283
 p4 reconcile	 285
Synopsis	285
Syntax	285
Description	285
Options	285
Usage Notes	286
Related Commands	287
 p4 reload	 289
Synopsis	289
Syntax	289
Description	289
Options	289
Usage Notes	289
Related Commands	290
 p4 rename	 291
Synopsis	291
Syntax	291
Description	291
Usage Notes	291
Examples	291
Related Commands	291

p4 renameuser	293
Synopsis	293
Syntax	293
Description	293
Usage and Limitations	293
Options	294
Usage Notes	294
Related Commands	294
 p4 reopen	 295
Synopsis	295
Syntax	295
Description	295
Options	295
Usage Notes	295
Examples	295
Related Commands	296
 p4 replicate	 297
Synopsis	297
Syntax	297
Description	297
Options	297
Usage Notes	298
Related Commands	298
 p4 resolve	 299
Synopsis	299
Syntax	299
Description	299
Revisions Used to Detect Conflicts	299
Resolve Options and Details	300
Non-Content-Related Resolves	302
Options	303
Usage Notes	305
Related Commands	305
 p4 resolved	 307
Synopsis	307
Syntax	307
Description	307
Options	307

Usage Notes	307
Related Commands	307
 p4 restore	 309
Synopsis	309
Syntax	309
Description	309
Options	309
Usage Notes	309
Related Commands	309
 p4 revert	 311
Synopsis	311
Syntax	311
Description	311
Options	311
Usage Notes	312
Examples	312
Related Commands	312
 p4 review	 313
Synopsis	313
Syntax	313
Description	313
Options	313
Usage Notes	313
Related Commands	314
 p4 reviews	 315
Synopsis	315
Syntax	315
Description	315
Options	315
Usage Notes	315
Related Commands	316
 p4 server	 317
Synopsis	317
Syntax	317
Description	317
Form Fields	317
Options	319
Usage Notes	319

Related Commands	319
p4 serverid	321
Synopsis	321
Syntax	321
Description	321
Options	321
Usage Notes	321
Related Commands	321
p4 servers	323
Synopsis	323
Syntax	323
Description	323
Options	323
Usage Notes	323
Related Commands	323
p4 set	325
Synopsis	325
Syntax	325
Description	325
Options	325
Usage Notes	326
Examples	326
p4 shelve	329
Synopsis	329
Syntax	329
Description	329
Options	330
Usage Notes	330
Related Commands	331
p4 sizes	333
Synopsis	333
Syntax	333
Description	333
Options	333
Usage Notes	334
Examples	334

p4 status	335
Synopsis	335
Syntax	335
Description	335
Options	335
Usage Notes	336
Related Commands	336
 p4 stream	 337
Synopsis	337
Syntax	337
Description	337
Form Fields	337
Options	341
Usage Notes	341
Examples	342
Related Commands	342
 p4 streams	 343
Synopsis	343
Syntax	343
Description	343
Options	343
Usage Notes	343
Examples	343
Related Commands	343
 p4 submit	 345
Synopsis	345
Syntax	345
Description	345
Form Fields	346
Options	346
Usage Notes	348
Examples	349
Related Commands	349
 p4 sync	 351
Synopsis	351
Syntax	351
Description	351
Options	351

Usage Notes	353
Scripting	353
Parallel processing	354
Working with streams	354
Retrying the command	354
Examples	355
Related Commands	356
 p4 tag	 357
Synopsis	357
Syntax	357
Description	357
Options	357
Usage Notes	357
Examples	358
Related Commands	358
 p4 tickets	 359
Synopsis	359
Syntax	359
Description	359
Options	359
Usage Notes	359
Examples	359
Related Commands	359
 p4 triggers	 361
Synopsis	361
Syntax	361
Description	361
Options	361
Usage Notes	361
Examples	362
Related Commands	362
 p4 trust	 363
Synopsis	363
Syntax	363
Description	363
Options	363
Usage Notes	364
 p4 typemap	 365

Synopsis	365
Syntax	365
Description	365
Form Fields	365
Options	366
Usage Notes	366
Examples	366
Related Commands	367
p4 unload	369
Synopsis	369
Syntax	369
Description	369
Options	370
Usage Notes	370
Related Commands	371
p4 unlock	373
Synopsis	373
Syntax	373
Description	373
Options	373
Usage Notes	373
Related Commands	374
p4 unshelve	375
Synopsis	375
Syntax	375
Description	375
Options	375
Usage Notes	376
Related Commands	376
p4 update	377
Synopsis	377
Syntax	377
Description	377
Options	377
Usage Notes	377
Related Commands	377
p4 user	379
Synopsis	379

Syntax	379
Description	379
Form Fields	380
Options	380
Usage Notes	380
Examples	383
Related Commands	383
 p4 users	 385
Synopsis	385
Syntax	385
Description	385
Options	385
Usage Notes	385
Related Commands	386
 p4 verify	 387
Synopsis	387
Syntax	387
Description	387
Options	387
Usage Notes	388
 p4 where	 389
Synopsis	389
Syntax	389
Description	389
Options	389
Usage Notes	389
Examples	390
Related Commands	390
 p4 workspace	 391
Synopsis	391
Syntax	391
Description	391
 p4 workspaces	 393
Synopsis	393
Syntax	393
Description	393

Environment Variables	395
 P4AUDIT	 397
Description	397
Usage Notes	397
Value if not Explicitly Set	397
Notes	397
 P4AUTH	 399
Description	399
Usage Notes	399
Value if not Explicitly Set	399
Examples	399
Notes	399
 P4BROKEROPTIONS	 401
Description	401
Usage Notes	401
Value if not Explicitly Set	401
Notes	401
 P4CHANGE	 403
Description	403
Usage Notes	403
Value if not Explicitly Set	403
Examples	403
Notes	403
 P4CHARSET	 405
Description	405
Usage Notes	405
Value if not Explicitly Set	405
Notes	405
 P4CLIENT	 407
Description	407
Usage Notes	407
Value if not Explicitly Set	407
Examples	407

P4CLIENTPATH	409
Description	409
Usage Notes	409
Value if not Explicitly Set	409
Examples	409
 P4COMMANDCHARSET	 411
Description	411
Usage Notes	411
Value if not Explicitly Set	411
Notes	411
 P4CONFIG	 413
Description	413
Usage Notes	413
Value if not Explicitly Set	413
Examples	413
Notes	413
 P4DEBUG	 415
Description	415
Usage Notes	415
Value if not Explicitly Set	415
Examples	415
Notes	415
 P4DESCRIPTION	 417
Description	417
Usage Notes	417
Value if not Explicitly Set	417
Examples	417
 P4DIFF	 419
Description	419
Usage Notes	419
Value if not Explicitly Set	419
Examples	419
Notes	419

P4DIFFUNICODE	421
Description	421
Usage Notes	421
Value if not Explicitly Set	421
Notes	421
 P4EDITOR	 423
Description	423
Usage Notes	423
Value if not Explicitly Set	423
Examples	423
Notes	423
 P4HOST	 425
Description	425
Usage Notes	425
Value if not Explicitly Set	425
Examples	425
Notes	425
 P4IGNORE	 427
Description	427
Usage Notes	427
Value if not Explicitly Set	427
Examples	427
Notes	427
 P4JOURNAL	 429
Description	429
Usage Notes	429
Value if not Explicitly Set	429
Examples	429
Notes	429
 P4LANGUAGE	 431
Description	431
Usage Notes	431
Value if not Explicitly Set	431

P4LOG	433
Description	433
Usage Notes	433
Value if not Explicitly Set	433
Examples	433
Notes	433
 P4LOGINSSO	 435
Description	435
Usage Notes	435
Value if not Explicitly Set	435
Examples	435
Notes	435
 P4MERGE	 437
Description	437
Usage Notes	437
Value if not Explicitly Set	437
Examples	437
Notes	437
 P4MERGEUNICODE	 439
Description	439
Usage Notes	439
Value if not Explicitly Set	439
Notes	439
 P4NAME	 441
Description	441
Usage Notes	441
Value if not Explicitly Set	441
Examples	441
 P4PAGER	 443
Description	443
Usage Notes	443
Value if not Explicitly Set	443
Examples	443
Notes	443

P4PASSWD	445
Description	445
Usage Notes	445
Value if not Explicitly Set	445
Notes	445
 P4PCACHE	 447
Description	447
Usage Notes	447
Value if not Explicitly Set	447
Notes	447
 P4PFSIZE	 449
Description	449
Usage Notes	449
Value if not Explicitly Set	449
Notes	449
 P4POPTIONS	 451
Description	451
Usage Notes	451
Value if not Explicitly Set	451
Notes	451
 P4PORT	 453
Description	453
Usage Notes	454
Value if not Explicitly Set	454
Examples	455
Notes	455
 P4ROOT	 457
Description	457
Usage Notes	457
Value if not Explicitly Set	457
Notes	457
 P4SSLDIR	 459

Description	459
Usage Notes	459
Value if not Explicitly Set	459
Examples	459
Notes	459
 P4TARGET	 461
Description	461
Usage Notes	461
Value if not Explicitly Set	461
Examples	461
Notes	461
 P4TICKETS	 463
Description	463
Usage Notes	463
Value if not Explicitly Set	463
Examples	463
Notes	463
 P4TRUST	 465
Description	465
Usage Notes	465
Value if not Explicitly Set	465
Notes	465
 P4USER	 467
Description	467
Usage Notes	467
Value if not Explicitly Set	467
Examples	467
Notes	467
 PWD	 469
Description	469
Usage Notes	469
Value if not Explicitly Set	469
Notes	469
 TMP, TEMP	 471
Description	471

Usage Notes	471
Value if not Explicitly Set	471
Notes	471
Additional Information	473
Global Options	475
Synopsis	475
Syntax	475
Options	475
Usage Notes	476
Examples	477
File Specifications	479
Synopsis	479
Syntax forms	479
Wildcards	479
Using revision specifiers	480
Using revision ranges	481
Limitations on characters in filenames and entities	482
Views	485
Synopsis	485
Usage Notes	485
Spaces in path and file names	486
Special characters in path and file names	486
Client Views	486
Branch Views	487
Label Views	488
File Types	489
Synopsis	489
Base filetypes	489
File type modifiers	490
Perforce file types for common file extensions	493
Keyword Expansion	494
Usage Notes	494
License Statements	497

About This Manual

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

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

Function	Where to look
Help	p4 help , p4 info , File Specifications , Views , Global Options , File Types
Client workspace	p4 clean , p4 client , p4 clients , p4 flush , p4 have , 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 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
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 , P4AUDIT , P4AUTH , P4BROKEROPTIONS , P4CHANGE , P4CHARSET , P4COMMANDCHARSET , P4CLIENT , P4CONFIG , P4DEBUG , P4DIFF , P4DIFFUNICODE , P4EDITOR , P4HOST , P4IGNORE , P4JOURNAL , P4LANGUAGE , P4LOG , P4MERGE , P4MERGEUNICODE , P4NAME , P4PAGER , P4PASSWD , P4PCACHE ,

Function	Where to look
	P4PFSIZE , P4POPTIONS , P4PORT , P4ROOT , P4TARGET , P4TICKETS , P4USER , PWD, TMP, TEMP

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

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>.

Getting help

In addition to the material provided in this manual, you can get help for Perforce command 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
```

Creating scripts

You can combine the commands described in this manual in scripts. Perforce works with two types of scripts:

- *Triggers* are 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.
- *Daemons* run at predetermined times, looking for changes to the Perforce metadata. When a daemon determines that the state of the depot has changed in some specified way, it runs other commands. For example, it might send email to parties interested in tracking the specified changes.

For more information about writing scripts, see the *Perforce System Administrator's Guide*.

What's New in 2014.1

The following sections describe the features introduced with the 2014.1 release that are described in this book.

Major Changes

Triggers have been extended in the following ways:

- The new trigger type **command** allows you to specify that triggers execute before or after the specified commands.
- In addition to using trigger variables to support trigger / server communication, triggers can now use a dictionary of key / value pairs accessed via **STDIN** and **STDOUT**. The setting of the **triggers.io** configuration variable determines which method is used
- You can now execute a trigger that is located in the depot.
- Trigger variables have been added that allow you to access trigger definition fields, to determine what kind of server the trigger is running on, to return information about command execution, and so on.

For complete information, see "Triggers and Daemons" in the *Perforce System Administrator's Guide*.

You can specify command options using long option names

Long option names must be preceded by two hyphens:

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

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

```
p4 reopen --explain
```

For complete information, see "Getting Help" above.

New Commands

The following new commands have been introduced in this release:

Reclaim disk space on a replicated server.

The new [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.

Reconcile workspace with depot.	The new p4 clean command syncs the client workspace to the depot so that modified and new files are made to match corresponding depot files. Files added to the workspace are deleted; depot files that have been modified are refreshed with their depot versions. This command is the equivalent to p4 reconcile with the <code>-w</code> flag.
Display a history of checkpoint and journal activity for the server.	Use the new p4 journals command to display this information.
Rename a user.	The p4 renameuser command renames the specified user and modifies associated artifacts to reflect the change: the user record, groups that include the user, properties that apply to the user, and so on.

Minor Changes

The current release of Perforce introduces a host of minor changes to improve performance and provide greater flexibility:

Print files located in the unload depot.	Use the <code>-U</code> flag to the p4 print command. This option allows you to print data about an unloaded client, label, or task stream.
Validate changes before these are submitted with the p4 populate command.	The execution of the <code>p4 populate</code> command now fires a change-submit trigger to allow interested parties to perform some validation before submission.
Log out another user.	As a superuser, you can use the <code>-a</code> flag to the p4 logout command to log out another user.
Backup a client workspace.	Use the <code>-o</code> flag to the p4 unload command.
Set a relative address with respect to a location other than that specified by <code>P4ROOT</code> .	Set the <code>server.depot.root</code> configuration variable to specify that the depot map field be evaluated relative to a location other than that specified by the <code>P4ROOT</code> environment variable. For information, see the description of these variables in the p4 configure command.
The p4 merge command outputs scheduled resolves by default.	This action was previously enabled by the <code>-Or</code> flag.
Display the size of depot files in human readable form.	The <code>-h</code> or <code>-H</code> option to the p4 sizes command displays size in human-readable form, using a scaling factor of 1024 for <code>-h</code> or 1000 for <code>-H</code> . The size displayed will be automatically scaled to bytes, kilobytes, megabytes, gigabytes, or terabytes, as appropriate.
Use variables to specify client template and label template names.	Use the new <code>template.client</code> variable to specify a default client template name; use the <code>template.label</code> variable to specify a default label template name. For information, see the description of these variables in the p4 configure command.

Force all file revisions to be copied.	The new <code>-f</code> flag to the p4 copy command forces 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.
Streams command with <code>-T</code> flag now forces tagged output.	See documentation for p4 streams for more information.

Improving Performance

The following changes can help improve performance.

Delete a list of files that can be used as a label.	The new <code>-d</code> flag to the p4 list command allows you to delete a label file list when a long-running process needs to use and reuse lists within the scope of one session without exhausting the server's process memory.
Synchronize your workspace with the depot using multiple threads to speed up file transfers.	You enable parallel processing by setting the <code>net.parallel.max</code> configuration variable to a value greater than one and by using the <code>--parallel</code> 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.
Set multiple values with a single call to the p4 counter or p4 key commands.	New command syntax allows you to set multiple values with a single call.
New configuration variables to fine tune the server.	You can use the configuration variables <code>rpl.jnlwait.adjust</code> , <code>rpl.jnlwait.interval</code> , <code>rpl.jnlwait.max</code> , <code>sys.rename.max</code> , and <code>sys.rename.wait</code> to fine-tune server performance. For default values, see the description of these variables in the p4 configure command. You should work with Perforce support for best results.

Support for Distributed Configuration

The following changes and additions support working in a distributed configuration.

Use new file type <code>ktext</code> keywords to provide better time/date formats for globally distributed installations.	Three new UTC keywords are added to RCS keywords to describe events in globally distributed installations. These are <code>DateUTC</code> , <code>DateTimeUTC</code> and <code>DateTimeTZ</code> . For more information, see file types .
Client SSO triggers can now determine whether they are connected to an intermediary.	With a distributed configuration in which a proxy or broker acts as an intermediary between the client and the server, the <code>%serverAddress%</code> 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. For information, see the material describing authentication triggers in the "Triggers and Daemons" chapter.

Pending or shelved changes on an edge server are visible throughout a distributed installation.

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.

Promote a shelved change from an Edge server to a Commit server and make it globally visible.

This makes the promoted shelf accessible to other Edge servers participating in the distributed configuration. Use the `-p` flag to the [p4 shelve](#) command to promote the shelf.

Delete a client workspace that has shelved files.

You can use the `-Fs` flag to the [p4 client](#) command to delete a workspace that has shelved files.

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 type] file...
```

Description

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

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` flag. (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` flag 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` flag explicitly specifies a file type, overriding both the typemap table and Perforce's default file type detection mechanism.

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

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

- You open a file for edit, then another user submits a changelist that deletes or moves the file. When you submit your edits, 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** flag, then submit the file.

- You open a file for move/add and another user submits a changelist that deletes the source file. When you submit the move, 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 <i>changelist</i>	Opens the files for add within the specified <i>changelist</i> . If this flag is not used, the files are linked to the default changelist.
-d	Downgrade file open status to simple add.
-f	Use the -f flag to force inclusion of wildcards in filenames. See the File Specifications chapter for details.
-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 <i>filetype</i>	Adds the file as the specified <i>filetype</i> , overriding any settings in the typemap table. Please see the File Types chapter for a list of Perforce file types.
g-opts	See the Global Options section.

Usage Notes

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

- [Wildcards](#) in file specifications provided to **p4 add** are expanded by the local operating system, not by the Perforce 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

<code>p4 add -t binary file.pdf</code>	Assigns a specific file type to a new file, overriding any settings in the typemap table.
<code>p4 add -c 13 *</code>	Opens all the files within the user's current directory for <code>add</code> , and links these files to changelist 13.
<code>p4 add README ~/src/*.c</code>	Opens all <code>*.c</code> files in the user's <code>~/src</code> directory for <code>add</code> ; also opens the <code>README</code> file in the user's current working directory for <code>add</code> . These files are linked to the default changelist.
<code>p4 add -f *.c</code>	<p>Opens a file named <code>*.c</code> for <code>add</code>.</p> <p>To refer to this file in views, or with other Perforce commands, you must subsequently use the hex expansion <code>%2A</code> in place of the asterisk.</p> <p>For more information, see “Limitations on characters in filenames and entities” on page 482.</p>

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 <i>filetype</i>

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
```

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.

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

The **p4 admin journal** command is equivalent to **p4d -jj**. For details, see the *System Administrator's Guide*. The files are created in the server root specified when the Perforce 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** flag is used, all of the form specification types are archived. If the **-s** flag option is used, then only those of the specified *type* are archived. Only those forms that have not yet been archived are created.

The **p4 admin resetpassword** command forces specified users with existing passwords to change their passwords before they can run another command. 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 user to reset their password, use **p4 admin resetpassword -u user**.

Options

-z	For p4 admin checkpoint and p4 admin journal , save the checkpoint and saved journal file in compressed (gzip) format, appending the .gz suffix to the files.
-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.
-a	For p4 admin updatespecdepot , update the spec depot with all current forms.
-s type	For p4 admin updatespecdepot , update the spec depot with forms of the specified type, where type is one of client , depot , branch , label , typemap , group , user , or job .
g-opts	See the Global Options section.

Usage Notes

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

- 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 *Perforce System Administrator's Guide* and *Distributing Perforce*.

Examples

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

Related Commands

To see the status of the last checkpoint [p4 counter](#) lastCheckpointAction

Synopsis

Print file lines along with their revisions.

Syntax

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

Description

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

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

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

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

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

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

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

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

Options

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

-I	Follow integrations into the file. If a line was introduced into the file by a merge, the source of the merge is indicated as the changelist that introduced the line. If that source was itself the result of an integration, that source will be used instead, and so on. The use of the -I option implies the -c option. The -I option cannot be combined with -i .
-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 .
-q	Quiet mode; suppress the one-line header for each file.
-t	Force p4 annotate to display non-text (binary) files.
g-opts	See the Global Options section.

Usage Notes

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

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

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

Examples

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

Synopsis

Archive obsolete revisions to an archive depot.

Syntax

```
p4 [g-opts] archive [ -n -h -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.

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

Options

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

Usage Notes

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

- 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` flag.
- 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` 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

<code>p4 archive file#3</code>	Archive revisions 1, 2, and 3 of <i>file</i> .
<code>p4 archive file#3,3</code>	Archive revision 3 of <i>file</i> .
<code>p4 print -A -o file</code>	Display the contents of an archived <i>file</i> without restoring it. Administrative privileges are not required.

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

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 -0a`.

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.
-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 .
-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.
-v value	The value of the attribute to set. To clear an attribute, omit the -v flag.
g-opts	See the Global Options section.

Usage Notes

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

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

Synopsis

Create or edit a branch mapping and its view.

Syntax

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

Description

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

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

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

Form Fields

Field Name	Type	Description
Branch:	read-only	The branch name, as provided on the command line.
Owner:	mandatory	The owner of the branch mapping. By default, this will be set to the user who created the branch. This field is unimportant unless the Option: field value is locked .
Access:	read-only	The date the branch mapping was last accessed.
Update:	read-only	The date the branch mapping was last changed.
Options:	mandatory	Either unlocked (the default) or locked . If locked , only the Owner: can modify the branch mapping, and the mapping can't be deleted until it is unlocked .
Description:	optional	A short description of the branch's purpose.
View:	mandatory	A set of mappings from one set of files in the depot (the <i>source files</i>) to another set of files in the depot (the <i>target</i>

Field Name	Type	Description
		<i>files</i>). The view maps from one location in the depot to another; it can't refer to a client workspace.
		For example, the branch view
		<code>//depot/main/... //depot/r2.1/...</code>
		maps all the files under <code>//depot/main</code> to <code>//depot/r2.1</code> .

Options

-d	Delete the named branch mapping. Files are not affected by this operation; only the stored mapping from one codeline to another is deleted. Normally, only the user who created the branch can use this flag.
-f	Force flag. Combined with -d , 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 flag).
-i	Read the branch mapping from standard input without invoking an editor.
-o	Write the branch mapping to standard output without invoking an editor.
-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 the Global Options section.

Usage Notes

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

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

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

Branch views can contain multiple mappings. See the [Views](#) chapter for more information on specifying views.

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

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

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

Related Commands

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

Synopsis

List existing branch mappings.

Syntax

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

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

<i>-t</i>	Display the time as well as the date of the last update to the branch.
<i>-m max</i>	List only the first <i>max</i> branch mappings.
<i>-e filter</i>	List only branches matching <i>filter</i> (case-sensitive).
<i>-E filter</i>	List only branches matching <i>filter</i> (case-insensitive).
<i>-u user</i>	List only branches owned by <i>user</i> .
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Related Commands

To create or edit a branch mapping [p4 branch](#)

Synopsis

Reclaim disk space on a replicated server.

Syntax

```
p4 [g-opts] cachepurge [-a | -f N | -m N | -s N ] [-i N -n -R -S N -O -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

-i <i>N</i>	Repeat the command every <i>N</i> seconds. If you omit this option, the command runs only once.
-n	Display a preview of the cachepurge operation without deleting any files.
-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 option is not specified, file names are used to determine order of deletion.
-f <i>N</i>	Delete sufficient file content to leave <i>N</i> number of bytes of free space for the file system.
-m <i>N</i>	Delete <i>N</i> file revisions. The amount of space this frees up depends on the size of the files.
-s <i>N</i>	Delete <i>N</i> bytes of file data. This can be helpful in those cases when you can predict the growth rate of file system resources.
-D <i>file...</i>	Limit action of command to the specified set of 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 <i>N</i>	Do not delete the <i>N</i> 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.
-O	Delete the files from the oldest to the newest; that is, delete older files before deleting newer files.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Synopsis

Create or edit a changelist specification.

Syntax

```
p4 [g-opts] change [ -s ] [ -f | -u ] [[-0] change ]
p4 [g-opts] change -d [ -f -s -0 ] change
p4 [g-opts] change -o [ -s ] [ -f ] [[-0] change ]
p4 [g-opts] change -i [ -s ] [ -f | -u ]
p4 [g-opts] change -t restricted|public [-U user] [ -f | -u | -0 ] change
p4 [g-opts] change -U user [-t restricted|public] [ -f ] change
```

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.)

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

To edit the description of a pending changelist, or to view the fields of a submitted changelist, use [p4 change changelist](#).

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

The command [p4 changelist](#) is an alias for [p4 change](#).

Form Fields

Field Name	Type	Description
Change:	Read-only	Contains the changelist number if editing an existing changelist, or new if creating a new changelist.
Client:	Read-only	Name of current client workspace.
Date:	Read-only	Date the changelist was last modified.
User:	Read-only	Name of current Perforce user. The owner of an empty pending changelist (that is, a pending changelist without any files in it) can transfer

Field Name	Type	Description
		ownership of the changelist to another existing user either by editing this field, or by using the -U user flag.
Status:	Read-only	<p>pending, shelved, submitted, or new. Not editable by the user.</p> <p>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.</p>
Description:	Writable, mandatory	<p>Textual description of changelist. This value <i>must</i> be changed before submission.</p> <p>If you do not have access to a restricted changelist, the description is replaced with a "no permission" message.</p>
Jobs:	List	<p>A list of jobs that are fixed by this changelist.</p> <p>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.</p>
Type:	Writable, optional	<p>Type of change: restricted or public.</p> <p>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.</p> <p>Public changes are displayed without restrictions.</p> <p>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.</p>
Files:	List	The list of files being submitted in this changelist. Files can be deleted from this list, and files that are found in the default changelist can be added.

Options

-d	Delete the changelist. This is usually allowed only with pending changelists that contain no files or pending fixes, but the superuser can delete changelists under other circumstances with the addition of the -f flag.
-f	Force flag. Allows the description, modification date, or user of a submitted changelist to be edited. Editing a submitted changelist requires admin or super access. Superusers and administrators can also overwrite read-only fields when using the -f flag. The -u and the -f flags are mutually exclusive.
-f -d	Forcibly delete a previously submitted changelist. Only a Perforce administrator or superuser can use this command, and the changelist must have had all of its files removed from the system with p4 obliterate .
-o	Write a changelist description to standard output.
-i	Read a changelist description from standard input. Input must be in the same format used by the p4 change form.
-s	Allows jobs to be assigned arbitrary status values on submission of the changelist, rather than the default status of closed . To leave a job unchanged, use the special status of same . On new changelists, the fix status is displayed as the special status ignore . (If the status is left unchanged, the job is not fixed by the submission of the changelist.) This option works in conjunction with the -s option to p4 fix , and is intended for use in conjunction with defect tracking systems.
-0	If a changelist was renumbered on submit, and you know only the original changelist number, use -0 and the original changelist number to view or edit the changelist.
-U user	The -U user flag 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 flag. (Unlike manually editing the User: field in the p4 change form, this option is much more convenient for use within a trigger or script.)
-t type	Change a submitted changelist's type to either restricted or public .
-u	Update a submitted changelist. Only the Jobs: , Description: , or Type: fields can be updated, and only the submitter of the changelist can update the changelist. The -u and the -f flags are mutually exclusive.
g-opts	See the Global Options section.

Usage Notes

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

- 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 <i>changelist</i>
To obtain a description of files changed in a changelist	p4 describe <i>changelist</i>

Synopsis

Create or edit a changelist specification.

Syntax

```
p4 [g-opts] changelist [ -s ] [ -f | -u ] [[-0] change ]
p4 [g-opts] changelist -d [ -f -s -0 ] change
p4 [g-opts] changelist -o [ -s ] [ -f ] [[-0] change ]
p4 [g-opts] changelist -i [ -s ] [ -f | -u ]
p4 [g-opts] changelist -t restricted|public [-U user] [ -f | -u | -0 ] change
p4 [g-opts] changelist -U user [-t restricted|public] [ -f ] change
```

Description

The command [p4 changelist](#) is an alias for [p4 change](#).

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](#).

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 -m max -s status -u user]  
[file[RevRange]...]
```

Description

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

By default, the format of each line is:

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

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

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

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

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

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

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

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

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` flag to view restricted changelists.

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

Options

-i	Include changelists that affected files that were integrated with the specified files.
-t	Display the time as well as the date of each change.
-l	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.
-f	View restricted changes (requires admin permission)
-c <i>client</i>	List only changes made from the named client workspace.
-m <i>max</i>	List only the highest numbered <i>max</i> changes.
-s <i>status</i>	Limit the list to the changelists with the given status (pending , submitted , or shelved)
-u <i>user</i>	List only changes made from the named user.
g-opts	See the Global Options section.

Usage Notes

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

Yes	Yes	list
-----	-----	-------------

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

Examples

<code>p4 changes -m 5 //depot/project/...</code>	Show the last five submitted, pending, or shelved changelists that include any file under the project directory
<code>p4 changes -m 5 -c eds_elm</code>	Show the last five submitted, pending, or shelved changelists from client workspace eds_elm .

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

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` flags. For example, using the `-a` flag 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` flag to override this behavior and ignore the `P4IGNORE` file.

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

Options

<code>-e</code>	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.
<code>-a</code>	Added files: Find files in the workspace that have no corresponding files in the depot and delete them.
<code>-d</code>	Deleted files: Find those files in the depot that do not exist in your workspace and add them to the workspace.
<code>-n</code>	Preview the results of the operation without performing any action.
<code>-I</code>	Do not perform any ignore checking; ignore any settings specified by <code>P4IGNORE</code> for added files.
<code>-l</code>	Display output in local file syntax with relative paths, similar to the workspace-centric view of <code>p4 status</code> .

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

Usage Notes

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

No	No	read
----	----	------

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

```
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](#)

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

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. The [p4 client](#) command is used to create or edit a client workspace specification; invoking this command displays a form in which the user enters the information required by Perforce to maintain the 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 doesn't 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	<p>The client workspace name, as specified in the P4CLIENT environment variable or its equivalents.</p> <p>When called without a <i>clientname</i> argument, p4 client operates on the workspace specified by the P4CLIENT environment variable or one of its equivalents. If called</p>

Field Name	Type	Description
		with a <i>clientname</i> argument on a locked workspace, the workspace specification is read-only.
Owner:	Writable, mandatory	The Perforce user name of the user who owns the workspace. The default is the user who created the workspace.
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. (Note: Reloading a workspace with p4 reload does not affect the access time.)
Host:	Writable, optional	<p>The name of the workstation on which this workspace resides. If included, operations on this client workspace can be run <i>only</i> from this host. If not set, access is allowed from any host.</p> <p>The hostname must be provided exactly as it appears in the output of p4 info when run from that host.</p> <p>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 <i>-H</i> flag to any p4 command, or with the P4HOST environment variable. For a similar mechanism that does provide security, use the IP address restriction feature of p4 protect.</p>
Description:	Writable, optional	A textual description of the workspace. The default text is Created by owner .
Root:	Writable, mandatory	<p>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.</p> <p>If you change this setting, you must physically relocate any files that currently reside there. On Windows client machines, you can specify the root as null to enable you to map files to multiple drives.</p>
AltRoots:	Writable, optional	<p>Up to two optional alternate client workspace roots.</p> <p>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.</p> <p>This enables users to use the same Perforce client workspace specification on multiple platforms, even those with different directory naming conventions.</p>

Field Name	Type	Description
		<p>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.</p> <p>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.</p> <p>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.</p>
Options:	Writable, mandatory	<p>A set of seven switches that control particular workspace options. See the <i>Usage Notes</i>, below, for a listing of these options.</p>
SubmitOptions	Writable, mandatory	<p>Flags to govern the default behavior of p4 submit.</p> <ul style="list-style-type: none"> • submitunchanged <p>All open files (with or without changes) are submitted to the depot. This is the default behavior of Perforce.</p> <ul style="list-style-type: none"> • submitunchanged+reopen <p>All open files (with or without changes) are submitted to the depot, and all files are automatically reopened in the default changelist.</p> <ul style="list-style-type: none"> • revertunchanged <p>Only those files with content, type, or resolved changes are submitted to the depot. Unchanged files are reverted.</p> <ul style="list-style-type: none"> • revertunchanged+reopen <p>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 <i>not</i> reopened in the default changelist.</p> <ul style="list-style-type: none"> • leaveunchanged <p>Only those files with content, type, or resolved changes are submitted to the depot. Any unchanged files are moved to the default changelist.</p> <ul style="list-style-type: none"> • leaveunchanged+reopen <p>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</p>

Field Name	Type	Description
		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 the <i>Usage Notes</i> , below, 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.
StreamAtChang	Writable, optional	<p>A changelist number that sets a back-in-time view of a stream.</p> <p>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.</p> <p>This field is ignored unless the Stream field is also set to a valid stream.</p>
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

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 oopened files or pending changes. (The -f flag permits Perforce administrators to delete locked workspaces owned by other users.) Clients can be deleted even if they have shelved files (see -Fs option).
-f	<p>Allows the last modification date, which is normally read-only, to be set. Administrators can use the -f flag to delete or modify locked workspaces owned by other users.</p> <p>Use of this flag requires admin access granted by p4 protect.</p>
-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 <i>change</i>	When used with -S <i>stream</i> , displays the workspace specification that would have been created for a <i>stream</i> at the moment the <i>change</i> was submitted.
-s	Switch workspace view. To switch the workspace view to a stream, specify -S <i>stream</i> . To switch the view defined for another workspace, specify -t <i>clientname</i> . Switching views is not allowed in a client that has opened files. The -f flag 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.
-S <i>stream</i>	Associates the workspace with the specified stream, which is used to generate its workspace view.
-t <i>clientname</i>	Copy client workspace <i>clientname</i> 's view and options into the View: and Options: field of this workspace. If you specify a default client template using the template.client configure variable, you do not have to specify this option.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- Use quotation marks to enclose depot-side or client side mappings of file or directory names that contain spaces.
- Spaces in 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	<p>If set, unopened files in the workspace are left writable.</p> <p>If allwrite is set, this option overrides noclobber.</p> <p>A setting of allwrite leaves unopened files writable by the current user; it does not set filesystem permissions to ensure writability by any user of a multiuser system.</p>	noallwrite
[no]clobber	<p>If set, a p4 sync overwrites ("clobbers") writable-but-unopened files in the workspace that have the same name as the newly-synced files.</p> <p>If allwrite is set, clobber is implied; even if present, the noclobber option is ignored.</p>	noclobber
[no]compress	<p>If set, the data stream between the user's workstation and the Perforce service is compressed.</p> <p>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.</p>	nocompress
[un]locked	<p>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.)</p>	unlocked

Option	Choice	Default
	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) flag with p4 client .	
[no]modtime	<p>For files <i>without</i> the +m (modtime) file type modifier:</p> <ul style="list-style-type: none"> • If modtime is set, the modification date (on the local filesystem) of a newly synced file is the datestamp <i>on the file</i> when the file was last modified. • If nomodtime is set, the modification date is the date and time <i>of sync</i>, regardless of version. <p>For files <i>with</i> 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.</p> <p>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 more complete discussion of the +m modifier, see the File Types section.</p>	<p>nomodtime (date and time of sync) for most files.</p> <p>Ignored for files with the +m file type modifier.</p>
[no]rmdir	<p>If set, p4 sync deletes empty directories in a workspace if all files in the directory have been removed.</p> <p>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.</p> <p>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.</p>	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	<p>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 LFs untouched.</p> <p>When you sync your workspace, line endings are set to LF. If you edit the file on a Windows machine, and your editor inserts CRs before each LF, the extra CRs do not appear in the archive file.</p> <p>The most common use of the 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.</p>

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_Article/CR-LF-Issues-and-Text-Line-endings

Working with Streams

Without **-s**, the **-S stream** flag 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** flag 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.

Examples

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

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

Synopsis

List all client workspaces currently known to the system.

Syntax

```
p4 [g-opts] clients [-t] [ -u user ] [ [-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

-t	Display the time as well as the date of the last update to the workspace.
-m <i>max</i>	List only the first <i>max</i> client workspaces.
-e <i>filter</i>	List only client workspaces matching <i>filter</i> (case-sensitive).
-E <i>filter</i>	List only client workspaces matching <i>filter</i> (case-insensitive).
-S <i>stream</i>	List client workspaces associated with the specified stream.
-u <i>user</i>	List only client workspaces owned by <i>user</i> .
-U	List only client workspaces unloaded with p4 unload .

-a	List all client workspaces, not just workspaces bound to this server.
-s <i>serverID</i>	List only client workspaces bound to the specified <i>serverID</i> . On an edge server, the -s flag defaults to the edge server's serverID.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Related Commands

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

Synopsis

Manage server configuration variables.

Syntax

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

Description

Use [p4 configure](#) to change the configuration of an active server. The configuration variables are described in [p4 help configurables](#) and in the table below.

Use **p4 configure show** to show the configuration state of the current server, a named server, or any configurable. Configurables are displayed along with an indication of whether a command line flag, environment variable, or [p4 configure](#) command was responsible for the configurable's setting.

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

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.

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

The following configurables are supported:

Configurable	Client or Server?	Default Value	Meaning
dbjournal.bufsize	Server	16K	Buffer size for journal and checkpoint read/write operations.
dbopen.nofsync	Server	0	Set to 1 to disable fsync() call when server closes a db.* database file, and permit the OS to determine when to write the modified data.
db.peeking	Server	0	Enable and configure lockless reads; when enabled, many common commands no

Configurable	Client or Server?	Default Value	Meaning
			<p>longer block other commands attempting to update the database. Installations that make use of this feature will typically set db.peeking to 2. See the <i>System Administrator's Guide</i> for details.</p> <p>0: Disable peeking. Behavior is identical to 2013.2 and earlier.</p> <p>1: New locking order is enabled, peeking is disabled, (diagnostic use only)</p> <p>2: New locking order is enabled, peeking is enabled, hx/dx optimization on.</p> <p>3: New locking order is enabled, peeking is enabled, hx/dx optimization is off.</p>
db.replication	Server	unset	<p>Control behavior of commands that access metadata (db.* files) on the Perforce server:</p> <p>readonly: user commands that read metadata are accepted; commands that modify metadata are rejected.</p> <p>Equivalent to starting a replica with the p4d -M readonly flag.</p>
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.accessupdate	Server	300	Wait this many seconds before requesting a write lock to update an access time.
dm.domain.accessforce	Server	3600	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.
dm.grep.maxrevs	Server	10K	Maximum number of revisions that can be searched with p4 grep .

Configurable	Client or Server?	Default Value	Meaning
dm.integ.engine	Server	3	<p>By default, use new integration engine with p4 integrate. (The p4 merge command always uses the v3 integration engine regardless of this setting.)</p> <p>Sites that wish to continue to use the old (2006.1) integration logic must set this configurable to 2 by running <code>p4 configure set dm.integ.engine=2</code>.</p>
dm.keys.hide	Server	0	If set to 1 or 2, p4 keys requires <code>admin</code> access. If set to 2, p4 key requires <code>admin</code> access.
dm.password.minlength	Server	8	Default minimum password length for servers where <code>security</code> is set to a nonzero value.
dm.protects.allow.admin	Server	0	Allow Perforce administrators to use <code>-a</code> , <code>-g</code> , and <code>-u</code> with p4 protects . By default, only superusers can use these flags.
dm.proxy.protects	Server	1	<p>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 <code>proxy-</code> prefix is prepended to the client IP address.</p> <p>Set this variable to 0 if you do not want to have connections that come in through an intermediary to have the <code>proxy-</code> prefix.</p> <p>For more information, see the <code>p4 protect</code> command.</p>
dm.resolve.attribs	Server	1	Enable resolve for attributes set with p4 attribute .
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.

Configurable	Client or Server?	Default Value	Meaning
<code>dm.user.accessupdate</code>	Server	300	Wait this many seconds before requesting a write lock to update an access time.
<code>dm.user.accessforce</code>	Server	3600	Wait this many seconds before forcibly updating an access time, even if server must wait for a lock.
<code>dm.user.loginattempts</code>	Server	3	Number of password attempts before delay. After the third failed login attempt, the user must wait 10 seconds.
<code>dm.user.noautocreate</code>	Server	0	Control behavior of automatic user creation. 0: A user is created whenever a nonexistent user runs any Perforce command 1: new users may only be created by running p4 user 2: new users may only be created by superusers running p4 user
<code>dm.user.resetpassword</code>	Server	0	If set, all new users created with a password are forced to reset their password before issuing any commands.
<code>filesys.binaryscan</code>	Client	64K	Scan the first <code>filesys.binaryscan</code> bytes for binary data when running p4 add .
<code>filesys.bufsize</code>	Client, Server	4K	Buffer size for client-side read/write operations.
<code>filesys.extendlowmark</code>	Client	32K	Minimum filesize before preallocation (Windows)
<code>filesys.P4ROOT.min</code>	Server	10M	Minimum disk space required on server root filesystem before server rejects commands.
<code>filesys.P4JOURNAL.min</code>	Server	10M	Minimum disk space required on server journal filesystem before server rejects commands.

Configurable	Client or Server?	Default Value	Meaning
<code>filesys.P4LOG.min</code>	Server	10M	Minimum diskpace required on server log filesystem before server rejects commands.
<code>filesys.TEMP.min</code>	Server	10M	Minimum diskpace required for temporary operations before server rejects commands.
<code>filesys.depot.min</code>	Server	10M	Minimum diskpace required for any depot before server rejects commands. (If there is less than <code>filesys.depot.min</code> diskpace available for any one depot, commands are rejected for transactions involving all depots.)
<code>filetype.maxtextsize</code>	Client	10M	Maximum file size for text type detection.
<code>journalPrefix</code>	Server	unset	Prefix or directory location for journal.
<code>lbr.bufsize</code>	Server, Proxy	4K	Buffer size for read/write operations to server's archive of versioned files.
<code>lbr.proxy.case</code>	Proxy	1	<p>1: File paths are always case-insensitive</p> <p>2: File paths are case-insensitive if server is case-insensitive</p> <p>3: File paths are always case-sensitive</p>
<code>lbr.replication</code>	Server	unset	<p>Control behavior of user commands that access versioned files on the Perforce server:</p> <p>readonly: user commands that read depot files are accepted; user commands that modify files are rejected.</p> <p>shared: this is a synonym for ondemand mode.</p> <p>ondemand: replicate versioned files if, and only if, explicitly requested by users of the replica server.</p>

Configurable	Client or Server?	Default Value	Meaning
			<p>cache: commands that reference file content are accepted, but do not automatically transfer files.</p> <p>none: no access to versioned files is permitted.</p> <p>Equivalent to starting a replica p4d process with one of the -D readonly, -D shared, (or -D ondemand), -D cache, or -D none flags.</p>
lbr.retry.max	Server	50K	In the event of a failed transfer, a replica will make lbr.retry.max attempts to retrieve the file.
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)
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	<p>Server process monitoring:</p> <p>0: Server process monitoring off</p> <p>1: monitor active processes only</p> <p>2: monitor both active and idle processes.</p> <p>See p4 monitor for details.</p>
net.backlog	Server, Proxy	10	Maximum length of queue for pending connections. Consider increasing if users find themselves unable to connect to extremely heavily-loaded servers.
net.keepalive.disable	Server	0	If non-zero, disable the sending of TCP keepalive packets

Configurable	Client or Server?	Default Value	Meaning
<code>net.keepalive.idle</code>	Server	0	Idle time (in seconds) before starting to send keepalives
<code>net.keepalive.interval</code>	Server	0	Interval (in seconds) between sending keepalive packets
<code>net.keepalive.count</code>	Server	0	Number of unacknowledged keepalives before failure
<code>net.maxfaultpub</code>	Proxy	100	A value in megabytes that controls the proxy's cache faulting behavior. A single p4 sync will not publish more than <code>net.maxfaultpub</code> megabytes of faults into <code>pdb.lbr</code> .
<code>net.maxwait</code>	Client, Server, Proxy	none	<p>Time, in seconds, before a network connection times out.</p> <p>Best practice is <i>not</i> 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.)</p>
<code>net.mimcheck</code>	Server, Proxy	1	<p>Man-in-the-middle network security level:</p> <p>0: Disable MitM checks.</p> <p>1: Check proxy/broker connections in legacy contexts.</p> <p>2: Connections from clients are checked for TCP forwarding.</p> <p>3: Connections from clients, proxies, and brokers are checked for TCP forwarding.</p> <p>4: All connections are checked; client software older than release 2010.1 cannot connect.</p> <p>5: All intermediate services are checked, and all service users must have valid tickets. Requires 2010.2 server and intermediate services.</p>

Configurable	Client or Server?	Default Value	Meaning
net.parallel.max	Server	0	<p>A value greater than one enables parallel processing up to the specified level, when syncing a client. In addition to setting this variable, you must use the --parallel option to the p4 sync command to further describe the processing desired.</p> <p>Values can range between 0 and 100. See the p4 sync command for more information.</p>
net.reuseport	Server	0	Set SO_REUSEPORT for listening socket.
net.rfc3484	Client, Server	0	If 1 , permit the operating system to determine whether IPv4 or IPv6 is used when resolving hostnames.
net.tcpsize	Client, Server, Proxy	64K	TCP send and receive buffer sizes, set on connection. Consider increasing for high-latency connections, such as the Proxy. Actual buffer size is large of this value and that defined by the OS.
proxy.monitor.level	Proxy	0	<p>0: monitoring disabled (default)</p> <p>1: monitor file transfers only</p> <p>2: monitor all operations</p> <p>3: monitor all traffic for all operations</p>
proxy.monitor.interval	Proxy	10	Set the proxy monitoring interval. Default is 10 seconds.
rsc.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.
rpl.checksum.auto	Server	0	<p>Level of database table checksum verification to perform when rotating journal. Each level corresponds to a larger set of database tables.</p> <p>0: disable checksums</p>

Configurable	Client or Server?	Default Value	Meaning
			<p>1: verify the most important system and revision tables</p> <p>2: verify all of level 1, plus tables that hold metadata that does not vary between replicas.</p> <p>3: verify all metadata, including metadata that is expected to vary on build-farm and edge-server replicas.</p>
<code>rpl.checksum.change</code>	Server	0	<p>Level of on-the-fly changelist verification to perform</p> <p>0: perform no verification</p> <p>1: write journal note at the end of a submit</p> <p>2: replica verifies changelist summary and writes to integrity.csv if the changelist does not match.</p> <p>3: replica verifies changelist summary and writes to integrity.csv even if the changelist does match.</p>
<code>rpl.checksum.table</code>	Server	0	<p>Level of table checksumming to perform.</p> <p>0: perform table-level checksumming only</p> <p>1: journal notes for table-unload and table-scan are processed by the replica, and are logged to integrity.csv if the check fails.</p> <p>2: results of journal note processing in the replica are logged even if the results match.</p>
<code>rpl.compress</code>	Server	0	<p>Enable replica/master network compression:</p> <p>0: No data stream compression.</p>

Configurable	Client or Server?	Default Value	Meaning
			<p>1: Data streams used for archive transfer to the replica (p4 pull -u) are compressed.</p> <p>2: Data streams used by p4 pull -u and p4 pull are compressed.</p> <p>3: All data streams (p4 pull -u, p4 pull, and data streams for commands forwarded to the replica) are compressed.</p>
<code>rpl.forward.all</code>	Server	0	<p>Enable replica forwarding of commands that update data.</p> <p>Required for replica servers that offer the forwarding-replica service.</p>
<code>rpl.jnlwait.adjust</code>	Server	25	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.
<code>rpl.jnlwait.interval</code>	Server	50	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.
<code>rpl.jnlwait.max</code>	Server	1000	Used to tune server performance when a forwarding replica has lots of users. Please consult Perforce Support for guidance in adjusting values.
<code>rpl.labels.global</code>	Server	0	<p>With a distributed Perforce service, there are both local and global labels. 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 can only be used with global (unbound) client workspaces.</p> <p>For the 2013.2 release, the default is for labels to be local. Set rpc.labels.global to 1 to make labels global by default.</p>

Configurable	Client or Server?	Default Value	Meaning
			When this configurable is set to 0, users can use the -g flag 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 flag is inverted to allow updating of local labels.
<code>rpl.verify.cache</code>	Server	0	If set, a replica server will re-verify the integrity of a cached file every time it delivers the file to the user. If the files do not match, it will re-fetch the file from the upstream server. This is computationally expensive on the replica and typically only useful in conjunction with Perforce technical support.
<code>run.users.authorize</code>	Server	0	If set, requires a user to authenticate before running p4 users .
<code>security</code>	Server	0	<p>Server security level:</p> <p>0: Legacy support: passwords not required, strength requirements unenforced</p> <p>1: Strong passwords required, existing passwords not reset, compatible with pre-2003.2 client software</p> <p>2: Strong passwords required, existing passwords reset, requires 2003.2 or higher client software</p> <p>3: Passwords must be strong, and ticket-based authentication (p4 login) is required.</p> <p>4: All of the above restrictions. Also, authenticated service users must be used for all replica server and remote depot connections to this server.</p>
<code>server.commandlimits</code>	Server	0	Policy for per-command resource limits:

Configurable	Client or Server?	Default Value	Meaning
			<p>0: All users may use command-line overrides for MaxResults, MaxScanRows, and MaxLockTime limits defined in the p4_group specs.</p> <p>1: Per-command flags may specify lower, but not higher, resource limits.</p> <p>2: All command-line resource limit flags are silently ignored.</p>
<code>server.depot.root</code>	Server		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.
<code>server.locks.dir</code>	Server	<code>server.locks.dir</code>	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.)
<code>server.locks.sync</code>	Server	1	<p>When set, the p4_sync command takes a client workspace lock in shared mode; set to 0 to prevent sync from taking a client workspace lock.</p> <p>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.</p>

Configurable	Client or Server?	Default Value	Meaning
<code>server.maxcommands</code>	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.
<code>serverlog.file.N</code>	Server	none	Server log file name associated with each structured log file. See p4 logparse for a list of valid filenames.
<code>serverlog.maxmb.N</code>	Server	none	For each structured log file, the size, in megabytes, at which the associated log file is rotated.
<code>serverlog.retain.N</code>	Server	none	For each structured log file, the number of rotated log files to retain on the server at any one time.
<code>serviceUser</code>	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.
<code>spec.hashbuckets</code>	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.
<code>ssl.secondary.suite</code>	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.
<code>startup.n</code>	Server	none	For replica servers, set <code>startup.1</code> through <code>startup.n</code> to be p4 pull threads to be spawned at startup. The <code>startup.n</code> configurables are processed sequentially. Processing stops at the first gap in the numerical sequence; any commands after a gap are ignored.

Configurable	Client or Server?	Default Value	Meaning
statefile	Server	state	For replica servers, the file used by the server to track the current journal position.
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.
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 <code>-t</code> flag on the p4 client command.
template.label	Server	none	Specifies the default label to be used as a template if the user omits the <code>-t</code> flag on the p4 label command.
trigger.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.

Options

<code>set variable=value</code>	Sets the named variable to the provided value.
<code>unset variable</code>	Unsets the named variable.

<code>show</code>	Shows the current configuration of the server currently specified by P4PORT
<code>show allservers</code>	Shows the configuration variables for all servers known to the system
<code>show variable</code>	Shows the setting of a specific configuration variable
<code>show P4NAME</code>	If a Perforce server was invoked with <code>-In P4NAME</code> or with the P4NAME environment variable set to a server name, shows the settings of the named server.
<code>g-opts</code>	See the Global Options section.

Usage Notes

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

- Configurables that affect client-side behavior can also be set in [P4CONFIG](#) files.

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

- The [p4 configure](#) command replaces many of the settings formerly set by [p4 counter](#).
- Changes to most configurables take effect immediately; for example, you do not have to restart the 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`.

- 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. See *Distributing Perforce* for details.

Related Commands

To list all counters and their values [p4 counters](#)

Synopsis

Copy files from one location in the depot to another.

Syntax

```
p4 [g-opts] copy [options] fromFile[rev] toFile
p4 [g-opts] copy [options] -b branch [-r] [toFile[rev]] ...
p4 [g-opts] copy [options] -b branch -s fromFile[rev] [toFile ...]
p4 [g-opts] copy [options] -S stream [-P parent] [-r][-F] [toFile[rev]...]
```

options: -c *change* -n -f -v -m *max* -q

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 flag. 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

-c <i>change</i>	Open the files in the specified pending changelist rather than in the default changelist.
-n	Preview the copy.
-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.
-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.
-m <i>max</i>	Specify the maximum number of files to copy, to limit the size of the operation.

-b <i>branch</i>	Specify a branch view to be used to determine source and target files.
-r [<i>toFile</i> [<i>rev</i>] ...]	Reverse the mappings in the branch view, integrating from the target files to the source files. Requires the -b flag.
-s <i>fromFile</i> [<i>rev</i>] [<i>toFile</i> ...]	Treat <i>fromFile</i> as the source and both sides of the branch view as the target. To restrict the scope of the target further, specify the optional <i>toFile</i> parameter. Overrides the -r flag, if specified. Requires -b .
-S <i>stream</i>	Specify the source stream. Changes are copied to its parent. You can use the -r flag 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.
-P <i>parent</i>	Specify a target stream other than the parent of the source stream. Requires -S .
-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 .
-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.
<i>g-opts</i>	See the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
<i>fromFile</i> : Yes	No	read access for <i>fromFile</i>
<i>toFile</i> : No		open access for <i>toFile</i>

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

<code>p4 copy -S //projectX/dev/...</code>	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

Synopsis

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

Syntax

```
p4 [g-opts] counter countername
p4 [g-opts] counter [-f] countername value
p4 [g-opts] counter [-f] -d countername
p4 [g-opts] counter [-f] -i countername
p4 [g-opts] counter [-f] -m [ pair list ]
```

Description

Counters provide long-term variable storage for scripts that access Perforce. For example, the Perforce review daemon uses a counter (**review**) that stores the number of the last processed changelist. 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* .
- The variant `p4 counter countername value` sets the value of variable *countername* set to *value*. If *countername* does not already exist, it is created.
- The variant `p4 counter -d countername` deletes the counter *countername* .
- The variant `p4 counter -i countername` increments the counter by one and returns the new value.
- 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 '-' followed by the name. See examples below.

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

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

Options

-d <i>countername</i>	Delete variable <i>countername</i> .
-i <i>countername</i>	Increment variable <i>countername</i> 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 <code>p4 help counters</code>).

Never set the **change** counter to a value that is lower than its current value.

-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 '-' followed by the name. See examples below.
g-opts	See the Global Options section.

Usage Notes

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

- If a counter does not exist, its value is returned as zero; counter names are not stored in the database until set to a nonzero value.
- The last changelist number known to the Perforce service (the output of **p4 counter change**) includes pending changelists created by users, but not yet submitted to the depot. If you're writing change review daemons, you might also want to know the changelist number of the last *submitted* changelist, which is the second field of the output of the command:

[p4 changes](#) -m 1 -s submitted
- The last changelist number *successfully* submitted (that is, no longer pending) to the Perforce service is held in the `maxCommitChange` counter.

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 second counter 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

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.

Options

<i>-e nameFilter</i>	List counters with a name that matches the <i>nameFilter</i> pattern, for example: <code>p4 counters -e 'mycounter-*</code>
<i>-m max</i>	List only the first <i>max</i> counters.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Related Commands

To view or change the value of a counter

[p4 counter](#)

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 the [Global Options](#) section.

Usage Notes

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

Related Commands

To check for integrations needed for a stream

[p4 istat](#)

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

<i>tablename</i>	Restrict output to the specified table name.
<i>tableversion</i>	Restrict output to the specified table version.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Examples

```
p4 dbschema db.protect
```

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

Synopsis

Display size or simple statistics for a database table.

Syntax

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

Description

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

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

Options

<i>-a</i>	Display statistics for all tables.
<i>-h</i>	Display a histogram showing distances between leaf pages.
<i>dbtable</i>	Display statistics for only the specified tables (for instance, <i>db.have</i> , <i>db.user</i> , and so on.)
<i>-s</i>	Report file sizes of database tables.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

N/A	N/A	super
-----	-----	-------

- Because [p4 dbstat](#) locks out write access to the database while it scans the tables, use this command with care. You will most often use this command when working with Perforce technical support.

Synopsis

Perform low-level verification of the database tables.

Syntax

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

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* flag, 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 tablename Restrict verification to the specified table name.

-U Perform a less-detailed validation

g-opts See the [Global Options](#) section.

Usage Notes

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

- [p4 dbverify](#) is equivalent to **p4d -xv**.

Synopsis

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

Syntax

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

Description

The [p4 delete](#) command opens file(s) in a client workspace for deletion from the depot. The files are immediately removed from the client workspace, but are not deleted from the depot until the corresponding changelist is 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 <i>changelist</i>	Opens the files for delete within the specified changelist. If this flag is not provided, the files are linked to the default changelist.
-n	Preview which files would be opened for delete, without actually changing any files or metadata.
-k	Delete the file on the shared versioning service, but keep a copy of the deleted file in your workspace.
-v	Delete a file that is not synced into the client workspace. To use this flag, 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, use p4 sync -k file , followed by p4 delete -k file .
g-opts	See the Global Options section.

Usage Notes

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

No	No	open
----	----	------

- A file that has been deleted from the client workspace with [p4 delete](#) can be reinstated in the client workspace and removed from the pending changelist with [p4 revert](#). To do this, you must revert the deletion before submitting the changelist.
- Perforce does not prevent users from opening files that are already open; its default scheme is to allow multiple users to open a file simultaneously, and then resolve file conflicts with [p4 resolve](#). To prevent someone else from opening a file once you've opened it, use [p4 lock](#). To determine whether or not another user already has a particular file open, use [p4 opened -a file](#).

Examples

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

Synopsis

Create or edit a depot specification.

Syntax

```

p4 [g-opts] depot depotname
p4 [g-opts] depot -d [-f] depotname
p4 [g-opts] depot -o 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 form. Depots can be of type **local**, **stream**, **remote**, **archive**, **spec**, or **unload**.

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.
Description:	Writable	A short description of the depot's purpose. Optional.
Type:	Writable	<p>local, remote, spec, stream, unload, or archive.</p> <p>Local depots are writable, and are the default depot type. Files reside in the server's root directory.</p> <p>Stream depots are also writable, but contain <i>streams</i>, a type of branch that includes hierarchy and policy.</p> <p>Remote depots reside on other servers, and cannot be written to. See “Working with Remote Depots” on page 97 for more information.</p> <p>The spec depot, if present, archives edited forms. See “Working with Spec Depots” on page 98 for more information.</p>

Field Name	Type	Description
		<p>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 <i>System Administrator's Guide</i>.</p> <p>Archive depots are 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.</p>
Address:	Writable	<p>If the Type: is remote, the address should be the P4PORT address of the remote server.</p> <p>If the Type: is local or spec, this field is ignored.</p>
Suffix:	Writable	<p>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 98 for more information.</p> <p>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.</p> <p>If the Type: is local or remote, this field is ignored.</p>
Map:	Writable	<p>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/....</p> <p>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.</p> <p>For more information, see "Providing Map Information" on page 97.</p>
SpecMap:	Writable	For spec depots, an optional description of which specs should be saved, expressed as a view.

Options

-d depotname	<p>Delete the depot <i>depotname</i>. The depot must not contain any files; the Perforce superuser can remove files with p4 obliterate.</p> <p>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.</p>
---------------------	--

-f	By default, when you delete a depot, the directory specified by the Map: field (typically under P4ROOT) must be empty. Use the -f flag 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.
g-opts	See the Global Options section.

Usage Notes

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

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/dopots/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 *Perforce System Administrator's Guide*.

Remote depots are accessed by a virtual user named **remote** (or, if configured, by the service user configured for the service that originates the request), and by default, all files on any 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 *System Administrator's Guide*.

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.

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

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`.

Options

g-opts See the [Global Options](#) section.

Usage Notes

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

Related Commands

To create a remote depot or a new local depot [p4 depot](#)

To remove all traces of a file from a depot [p4 obliterate](#)

Synopsis

Provides information about changelists and the changelists' files.

Syntax

```
p4 [g-opts] describe [ -dflags -s -S -f -O ] 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** flag.

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

-f	Force the display of descriptions for restricted changelists. This flag 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.
-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.

-dflags	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the <i>Usage Notes</i> below for a flag listing.
g-opts	See the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	read; list for <code>p4 describe -s</code>

The diff flags supported by [p4 describe](#) are:

Flag	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 <i>num</i> 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 <i>num</i> lines of context, in a form compatible with the <code>patch(1)</code> utility.
-dl	ignore line-ending (CR/LF) convention when finding diffs
-db	ignore changes made within whitespace; this flag implies -dl .
-dw	ignore whitespace altogether; this flag implies -dl .

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

Synopsis

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

Syntax

```
p4 [g-opts] diff [-dflags -f -m max -Od -sa -sb -sd -se -sr -sl -t] [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

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

-se	Show only the names of unopened files in the client workspace that are different than the revision in the depot.
-sr	Show only the names of opened files in the client workspace that are identical to the revision in the depot.
-sl file...	Every unopened <i>file</i> is compared with the depot, and listed with a status of same , diff , or missing . If you use the -f flag together with the -sl flag, files that are open for edit are also compared and their status is listed.
-t	Diff the revisions even if the files are not of type text .
g-opts	See the Global Options section.

Usage Notes

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

- The diff flags supported by [p4 diff](#) are:

Flag	Name
-dn	RCS output format, showing additions and deletions made to the file and associated line ranges.
-dc[num]	context output format, showing line number ranges and <i>num</i> 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 <i>num</i> 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 flag implies -dl .
-dw	ignore whitespace altogether; this flag implies -dl .

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

p4 diff -dub file

specifies a unified diff that ignores changes in whitespace.

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

- 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

<code>p4 diff file#5</code>	Compare the client workspace revision of file <code>file</code> to the fifth depot revision.
<code>p4 diff @1999/05/22</code>	Compare all open files in the client workspace to the revisions in the depot as of midnight on May 22, 1999.
<code>p4 diff -du file</code>	Run the comparison on file <code>file</code> , displaying output in a format suitable for the <code>patch(1)</code> utility.
<code>p4 diff -sr p4 -x - revert</code>	<p>Revert all open, unchanged files.</p> <p>This differs from 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.</p> <p>The first command shows all open, unchanged files. The second command (running <code>p4 -x</code> and taking arguments, one per line, from standard input, abbreviated as <code>"-"</code>) reverts each file in that list.</p> <p>(This is the UNIX version of this command; it uses a pipe. Most operating systems have some equivalent way of performing these operations in series).</p> <p>For more information about the <code>-x</code> option to <code>p4</code>, see the Global Options section.</p>

Related Commands

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

Synopsis

Compare two depot file revisions.

Syntax

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

[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

-q	Quiet diff. Display only the header; if <i>file1</i> and <i>file2</i> are identical, display only " <i>file1</i> - no differing files" as the output.
-Od	Limit output to only those files that differ.
-dflags	Runs the diff routine with one of a subset of the standard UNIX diff flags. See the <i>Usage Notes</i> below for a listing of these flags.
-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 <i>from</i> or <i>to</i> file specifications.
-S stream [-P parent]	Diff a stream with its parent. To diff the stream with a stream other than its configured parent, specify <i>-P</i> .
-t	Diff the file revisions even if the file(s) are not of type text .
-u	Generate unified output format, showing added and deleted lines with sufficient context for compatibility with the patch(1) utility. Only those files that differ are included. File names and dates remain in Perforce syntax.
g-opts	See the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	No	read access necessary for both file revisions

- The diff flags supported by [p4 diff2](#) are:

Flag	Name
-dn	RCS output format, showing additions and deletions made to the file and associated line ranges.
-dc[num]	context output format, showing line number ranges and <i>num</i> 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 <i>num</i> lines of context, in a form compatible with the patch(1) utility.

Flag	Name
-dl	ignore line-ending (CR/LF) convention when finding diffs
-db	ignore changes made within whitespace; this flag implies -dl .
-dw	ignore whitespace altogether; this flag implies -dl .

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

```
p4 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](#)

Synopsis

List the immediate subdirectories of specified depot directories.

Syntax

```
p4 [g-opts] dirs [-C -D -H] [-S stream] 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** flag.

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.
g-opts	See the Global Options section.

Usage Notes

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

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

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

Examples

<code>p4 dirs //depot/projects/*</code>	Returns a list of all the immediate subdirectories of <code>//depot/projects</code> .
<code>p4 dirs //depot/a/* //depot/b/*</code>	Returns a list of all immediate subdirectories of <code>//depot/a</code> and <code>//depot/b</code> .
<code>p4 dirs //depot/...</code>	The <code>"..."</code> 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

Synopsis

Display disk space information on the server.

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.

Options

<i>P4ROOT</i>	Report disk space available for filesystem holding P4ROOT .
<i>P4JOURNAL</i>	Report disk space available for filesystem holding P4JOURNAL .
<i>P4LOG</i>	Report disk space available for filesystem holding P4LOG (server log).
<i>TEMP</i>	Report disk space available for filesystem holding temporary files. If not defined, uses P4ROOT on Windows, and <i>/tmp</i> on Unix.
<i>journalPrefix</i>	Report disk space available for filesystem holding a non-standard journal location.
<i>depot</i>	Report disk space available for filesystem holding the specified <i>depot</i> .
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- 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.

- [p4 df](#) is an alias for [p4 diskpace](#).

Related Commands

To configure Perforce's behavior when diskpace is low

[p4 configure](#)

Synopsis

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

Syntax

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

Description

[p4 edit](#) opens files for editing within the client workspace. The specified file(s) are linked to a changelist, but the files are not actually changed in the depot until the changelist is 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 flag is not provided, the files are linked to the default changelist.
-t type	Stores the new file revision as the specified type, overriding the file type of the previous revision of the same file. 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 the File Types section for a list of file types.
-k	Keep existing workspace files; mark the file as open for edit even if the file is not in the client 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.
g-opts	See the Global Options section.

Usage Notes

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

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

<code>p4 edit -t text+k doc/*.txt</code>	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.
<code>p4 edit -t +l //depotname/...</code>	Implements pessimistic locking (exclusive-open) for all files in a depot. After this changelist is submitted, only one user at a time will be able to edit files in the depot named depotname .
<code>p4 edit -c 14 ...</code>	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.
<code>p4 edit status%40jan1.txt</code>	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 482.](#)

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

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` flag 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

<code>-c</code>	Specifies a checkpoint number or position token of the form <i>checkpointnum#byteoffset</i> .
<code>-j</code>	Specify a journal number or position token of the form <i>journalnum/byteoffset</i> .
<code>-f</code>	Format the output so that non-textual datatypes are formatted appropriately.
<code>-r</code>	Display raw journal output; this argument applies to journals only.
<code>-J prefix</code>	Specifies a filename prefix for the journal, such as that used with <code>p4d -jc prefix</code>
<code>-l lines</code>	Limit output to the specified number of <i>lines</i> of journal records.
<code>-F filter</code>	Limit output to records that match the specified <i>filter</i> pattern. For example, <code>-F "table = db.configure"</code>
<code>-T tableexcludelist</code>	Supply a list of database tables (for example, <code>db.have</code> and <code>db.client</code>) 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"`

-P *filterpattern*

Limit output to records that match the specified filter pattern. Multiple filter patterns can be specified with multiple `-P` flags.

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.

g-opts

See the [Global Options](#) section.

Usage Notes

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

- Compressed journals or checkpoints are not supported.

Examples

`p4 export -T "db.have db.working"`

Run [p4 export](#), but ignore records in the `db.have` and `db.working` tables.

Related Commands

To replicate metadata from one server to another

[p4 replicate](#)

To pull journal records (and file content) from a master server to a replica server

[p4 pull](#)

Synopsis

Print detailed information about files' revisions.

Syntax

```
p4 [g-opts] filelog [-c change -h -i -l -L -t -m max -p -s] 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** flag is used).
- *user* is the name of the user who submitted the revision;
- *client* is the name of the client workspace from which the revision was submitted;
- *type* is the [type](#) of the file at the given revision; and
- *description* is the first 30 characters of the corresponding changelist's description.

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

Options

-c <i>change</i>	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.
-p	When used with the -h flag, do not follow content of promoted task streams. This flag is useful when there are many child task streams branched from the supplied <i>file</i> argument.
-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 flag 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.
-t	Display the time as well as the date.
-m <i>max</i>	List only the first <i>max</i> changes per file output.
-s	Display a shortened form of output by ignoring non-contributory integrations (for example, integrations involving "branch into" or copy into" operations are not displayed)
g-opts	See the Global Options section.

Usage Notes

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

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

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

Examples

<code>p4 filelog //depot/proj1/...</code>	Display the revision history for every file under the depot's proj1 directory.
<code>p4 filelog file1.c@100,@120</code>	Display the revision history for file1.c from changelists 100 through 120.
<code>p4 filelog file1.c#have,#head</code>	If you do not have the latest revision of file1.c , display revision history since your last sync.
<code>p4 filelog file1.c file1.h</code>	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

Synopsis

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

Syntax

```
p4 [g-opts] files [-a] [-A] [-e] [-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.
-m max	Limit output to the first <i>max</i> files.
-U unloadfile	List only files in the unload depot. See p4 unload for details.

*g-opts*See the [Global Options](#) section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	list
<ul style="list-style-type: none"> 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 <i>highest</i> 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, <i>all</i> 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

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

-d	Delete the fix record for the specified job at the specified changelist. The job's status will not change.
-s status	<p>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).</p> <p>If the changelist to which you're linking the job been submitted, the status value is immediately reflected in the job's status.</p> <p>If the changelist is pending, the job status is changed on submission of the changelist, provided that the -s flag is also supplied to p4 submit and the desired status appears next to the job in the p4 submit form's Jobs: field.</p> <p>To leave a job unchanged, use the special status of same.</p>
g-opts	See the Global Options section.

Usage Notes

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

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

Wrong number of words for field 'Jobs'.

is displayed.

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

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

Examples

```
p4 fix -c 201 job000141 job002034
```

Mark two jobs as being fixed by changelist 201.

If changelist 201 is still **pending**, the jobs' status is changed to **closed** when the changelist is submitted.

```
p4 fix -c 201 -s suspended job002433
```

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

Requires use of the `-s` flag with [p4 submit](#).

Related Commands

To add or delete a job from a pending changelist	p4_change
To add or delete a job from the default changelist	p4_submit
To view a list of connections between jobs and changelists	p4_fixes
To create or edit a job	p4_job
To list all jobs, or a subset of jobs	p4_jobs
To change the format of jobs at your site (superuser only)	p4_jobspec
To read information about the format of jobs at your site	p4_jobspec -o

Synopsis

List jobs and the changelists that fix them.

Syntax

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

Description

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

If invoked without arguments, [p4 fixes](#) displays all fix records. Fix records are displayed in the following format:

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

You can limit the listed fixes by combining the following flags when calling [p4 fixes](#):

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

Options

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

Usage Notes

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

Examples

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

Related Commands

To create or edit an existing job	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 (<i>superuser only</i>)	p4 jobspec
To read information about the format of jobs at your site	p4 jobspec -o

Synopsis

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

Syntax

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

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](#).

Description

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 than the corresponding [p4 sync](#).

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

Options

-f	Force the flush. Perforce performs the flush even if the client workspace already has the file at the specified revision. If the file is writable, it is overwritten. This flag does not affect open files, but it <i>does</i> override the noclobber client option.
-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 the Global Options section.

Usage Notes

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

Yes	Yes	read
-----	-----	------

- Because [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 than 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

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.

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

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

To filter the output on some function of the form fields (for example, all files larger than a certain size and with a certain filetype), use the *-F filter* flag.

To limit output to the set of fields specified in a *fields* argument, use the *-T fields* flag. The 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
<i>attr-name</i>	Attribute value for name	<i>attr-myAttr critical</i>
<i>attrProp-name</i>	Set if <i>attr-name</i> is a propagating attribute	<i>attrProp-myAttr</i>
<i>clientFile</i>	Local path to file (in local syntax by default, or in Perforce syntax with the <i>-Op</i> option)	<i>/staff/userid/src/file.c</i> (or <i>//workspace/src/file.c</i> in Perforce syntax)
<i>depotFile</i>	Depot path to file	<i>//depot/src/file.c</i>
<i>movedFile</i>	Name in depot of moved to/ from file	<i>//depot/src/file.c</i>
<i>path</i>	Local path to file	<i>//workspace/src/file.c</i>

Field Name	Description	Example/Notes
isMapped	Set if the file is open for add and it 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 the chapter on File Types .)
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
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
fileSize	File length in bytes (requires -Ol option)	63488
action	Open action, if opened in your workspace	one of add , edit , delete , branch , move/add , move/delete , integrate , import , purge , or archive .

Field Name	Description	Example/Notes
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
change	Open changelist number, if opened in your workspace	75331
resolved	The number, if any, of resolved integration records	5
unresolved	The number, if any, of unresolved integration records	2
reresolvable	The number, if any, of re-resolvable integration records	1
otherOpen	The number of other users who have the file open, blank if no other users have the file open	1, 2, 3... <i>n</i> , preceded by <i>n</i> records listing the users (0 through <i>n</i>-1) with otherOpen<i>n</i> , otherAction<i>n</i> , and otherLock<i>n</i> fields as applicable. For example: ... otherOpen 3 otherOpen0 user1@ws1 otherOpen1 user2@ws2 otherOpen2 user3@ws3
otherOpen<i>n</i>	For each user with the file open, the workspace and user with the open file	user123@workstation9
otherLock	Present and set to null if another user has the file locked, otherwise not present	otherLock
otherLock<i>n</i>	For each user with the file locked, the workspace and user holding the lock	user123@workstation9 Because only one user at a time can lock a file, if <i>n</i> is set, <i>n</i> is always 0.
otherAction<i>n</i>	For each user with the file open, the action taken	one of add , edit , delete , branch , move/add , move/delete , integrate , import , purge , or archive .

Field Name	Description	Example/Notes
<code>otherChangen</code>	For every changelist with the file open, the changelist	75612
<code>openattr-name</code>	For every changelist with the file open, the attribute value for name	<code>attr-name</code>
<code>openattrProp-name</code>	Set if <code>attr-name</code> is a propagating attribute	<code>attrProp-name</code>
<code>ourLock</code>	Present and set to null if the current user has the file locked, otherwise not present	<code>ourLock</code>
<code>resolveActionn</code> <code>resolveBaseFilen</code> <code>resolveBaseRevn</code> <code>resolveFromFilen</code> <code>resolveStartFromRevn</code> <code>resolveEndFromRevn</code>	Pending integration action, base file, base revision number, from file, starting, and ending revision, respectively.	For pending integration record information, use the <code>-Or</code> option.
<code>totalFileCount</code>	The number of files examined.	Appears in the first file's output when you use the <code>-m max</code> option in conjunction with one of the <code>-Sx</code> or <code>-r</code> sorting options.

Options

<code>-A pattern</code>	Restrict displayed attributes to those that match the specified <i>pattern</i> . For example, for the selected files, <code>-A foo*</code> displays only attributes whose name starts with foo.
<code>-F filter</code>	List only those files that match the criteria specified by <i>filter</i> . Please see the <i>Usage Notes</i> below for a discussion of filters.
<code>-L</code>	For scripting purposes, report file information on a list of valid file arguments in full depot syntax with a valid revision number.
<code>-T fields</code>	List only those fields that match the field names specified by <i>fields</i> . The list of field names can be separated by spaces or commas.
<code>-c change</code>	Display only files affected after the given changelist number. This operation is much faster than using a revision range on the affected files.
<code>-e change</code>	Display only files affected by the given changelist number. This option is much faster than using a revision range on the affected files.
<code>-m max</code>	Produce fstat output for only the first <i>max</i> files.
<code>-r</code>	Sort the output in reverse order.

-0a	Output attributes set by p4 attribute .
-0d	Output the digest of an attribute.
-0e	Output attribute values encoded as hex.
-0f	Output all revisions for the given files, suppressing the other[...] and resolve[...] fields.
-0l	Output a fileSize field displaying the length of the file and a digest field for each revision. p4 fstat -e shelvedChange -Rs -0l reports the file size and digest of files shelved at the specified change.
-0p	Display the clientFile in Perforce syntax, as opposed to local syntax.
-0r	Display pending integration record data for files open in the current workspace.
-0s	Shorten output by excluding client workspace data (for instance, the clientFile field).
-Rc	Limit output to files mapped into the current workspace.
-Rh	Limit output to files on your have list; that is, to files synced to the current workspace.
-Rn	Limit output to files opened at revisions not at the head revision.
-Ro	Limit output to open files in the current workspace.
-Rr	Limit output to open files that have been resolved.
-Rs	Limit output to shelved files. Requires -e changelist option. p4 fstat -e shelvedChange -Rs -0l reports the file size and digest of files shelved at the specified change.
-Ru	Limit output to open files that are unresolved.
-St	Sort by filetype.
-Sd	Sort by date.
-Sr	Sort by head revision.
-Sh	Sort by have revision.
-Ss	Sort by filesize.
-U	Include files in the unload depot when displaying data. See p4 unload for details.
g-opts	See the Global Options section.

The **-s** global option (which prefixes each line of output with a tag describing the type of output as **error**, **warning**, **info**, **text**, or **exit**) can be particularly useful when used with [p4 fstat](#).

Usage Notes

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

The only argument required for the **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.

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**.

Other Usage Notes

- If you use **-e changelist** with the **-Ro** option, only pending changes are considered, so that files open for add are included in the output.
- For files containing the special characters @, #, *, and %, the **clientFile** displays the special character, and the **depotFile** displays the filename containing the ASCII expression of the character's hexadecimal value.
- The **size** and **digest** fields are based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.
- The **-L** flag 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.
- The syntax of [p4 fstat](#) was changed in Release 2004.2. The older **-C**, **-H**, **-W**, **-P**, **-I**, and **-s** options are supported for compatibility purposes.

Examples

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

<code>p4 fstat -F "clientFile=c:\\ws\\file.c" //depot/main/...</code>	If a path contains backslashes, escape them with backslashes.
<code>p4 fstat -F "clientFile~=c:\\ws\\[Ff]ile.c" //depot/main/...</code>	Use the ~= regular expression modifier to specify a regexp that matches <code>File.c</code> and <code>file.c</code>
<code>p4 fstat -Ol -F "fileSize < 1024 & headType=text" //depot/main/...</code>	Display information on all text files under <code>//depot/main/...</code> that are smaller than 1024 bytes in length.
<code>p4 fstat -T 'depotFile, headRev' file.c</code>	Display only the <code>depotFile</code> and <code>headRev</code> fields for <code>file.c</code> .

Related Commands

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

Synopsis

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

Syntax

```
p4 [g-opts] grep [ options ] -e pattern file [revRange]...
```

options: -a -i -n -v -Anum -Bnum -Cnum -l -L -T -s -F -G

Description

The [p4 grep](#) command searches for lines that match a given regular expression.

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

Options

-e pattern	The <i>patterns</i> used by p4 grep are regular expressions comparable to those used in UNIX; their syntax is fully defined in the output of p4 help grep .
-a	Search all revisions within the specified range, rather than only the highest revision in the range
-i	Perform case-insensitive pattern matching. (By default, matching is case-sensitive.)
-n	Display a matching line number after the file revision number.
-v	Display files with non-matching lines.
-F	Interpret the pattern as a fixed string
-G	Interpret the pattern as a regular expression.
-L	Display the name of each selected file from which no output would normally have been displayed; scanning stops at the first match.
-l	Display the name of each selected file from which output would have been displayed; scanning stops at the first match.
-s	Suppress error messages from files with more than 4096 characters in a single line. (By default, p4 grep abandons these files and reports an error)
-t	Treat binary files as text. (By default, only files of type text are selected for pattern matching.)

-A <i>num</i>	Display <i>num</i> lines of trailing context after matching lines.
-B <i>num</i>	Display <i>num</i> lines of trailing context before matching lines.
-C <i>num</i>	Display <i>num</i> lines of output context.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

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

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 ] 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, limit the maximum amount of data that can be retrieved from Perforce by particular users with a single command, and to set the timeout period for [p4 login](#) tickets.

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

Form Fields

Field Name	Type	Description
Group:	Read-only	The name of the group, as entered on the command line.
MaxResults:	Writable	The maximum number of results that members of this group can access from the service from a single command. The default value is <code>unset</code> . See the <i>Usage Notes</i> below for more details.
MaxScanRows:	Writable	The maximum number of rows that members of this group can scan from the service from a single command. The default value is <code>unset</code> . See the <i>Usage Notes</i> below for more details.
MaxLockTime:	Writable	The maximum length of time (in milliseconds) that any one operation can lock any database table when scanning data. The default value is <code>unset</code> . See the <i>Usage Notes</i> below for more details.
Timeout:	Writable	The duration (in seconds) of the validity of a session ticket created by p4 login . The default value is 43200 seconds (12 hours). To create a ticket that does not expire, set the <code>Timeout:</code> field to <code>unlimited</code> .

Field Name	Type	Description
PasswordTimeout:	Writable	The length of time (in seconds) for which passwords for users in this group remain valid. To disable password aging, use a value of unset .
Users:	Writable, multi-line	The Perforce usernames of the group members. Each user name must be typed on its own line, and should be indented.
Subgroups:	Writable, multi-line	<p>Names of other Perforce groups.</p> <p>To add all users in a previously defined group to the group you're presently working with, include the group name in the 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.</p> <p>Every member of any previously defined group you list in the Subgroups: field will be a member of the group you're now defining.</p>
Owners:	Writable, multi-line	<p>Names of other Perforce users.</p> <p>Group owners without super access are permitted to administer this group, provided that they use the -a flag.</p> <p>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.</p>

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 flag is used.
-d groupname	Delete group <i>groupname</i> . The members of the group are affected only if their access level or maxresults value changes as a result of the group's deletion.
-i	Read the form from standard input without invoking the user's editor. The new group specification replaces the previous one.
-o	Write the form to standard output without invoking the user's editor.
g-opts	See the Global Options section.

Usage Notes

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

- 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 2592000 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.

If any of the `MaxResults`, `MaxScanRows`, or `MaxLockTime` 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` and `MaxLockTime` limits are computed in the same way.)

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

- To unload a workspace or label, a user must be able to scan *all* the files in the workspace's have list and/or files tagged by the label. Administrators should set `MaxScanRows` and `MaxResults` high enough that users will not need to ask for assistance with [p4 unload](#) or [p4 reload](#) operations.
- To display a group's `maxresults`, `maxscanrows`, `maxlocktime`, and `timeout` limits, use `p4 groups -v groupname`.

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

Related Commands

To modify users' access levels

[p4 protect](#)

To view a list of existing groups

[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

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

Usage Notes

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

- To see all the members of a particular group, use [p4_group](#) -o *groupname*. This variation of [p4_group](#) requires only **list** access.

Examples

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

Related Commands

To create or edit an existing group of users	p4_group
To view a list of all the members and specifications of a particular group	p4_group -o <i>groupname</i>
To set Perforce access levels for the members of a particular group	p4_protect

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 the [Global Options](#) section.

Usage Notes

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

No	No	list
----	----	------

- Some Perforce documentation refers to a client workspace's *have list*. The have list is the list of files reported by [p4 have](#), and is the list of file revisions that have been most recently synced from the depot.

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
```

In each of these two pairs of commands:

```
p4 sync //depot/name/...#4
p4 have //depot/name
```

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](#)

Synopsis

Provide on-line help for Perforce.

Syntax

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

Description

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

[p4 help](#) with no arguments lists all the available [p4 help](#) options. [p4 help command](#) provides help on the named *command*. [p4 help keyword](#) takes the following keywords as arguments:

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

Usage Notes

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

Related Commands

To view information about the current Perforce configuration

[p4 info](#)

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](#):

```
User name: joe
Client name: joes_client
Client host: joes_workstation
Client root: /usr/joe/projects
Current directory: /usr/joe/projects/source
Client address: 192.168.0.123
Server address: p4server:1666
Server root: /usr/depot/p4d
Server date: 2012/01/28 12:11:47 -0700 PDT
Server uptime: 752:41:33
Server version: P4D/FREEBSD/2012.1/406375 (2012/01/25)
Server license: P4Admin <p4adm> 20 users (expires 2013/01/01)
Server license-ip: 10.0.0.2
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.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Related Commands

To read Perforce's help files	p4 help
To display Perforce Proxy connection information	p4 proxy
To view version information for your Perforce application	p4 -V

Synopsis

Open files for branching or merging.

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] fromFile [revRange]
```

options: -c changelist -Di -f -h -Oflags -n -m max -Rflags -q -v

Description

When you've made changes to a file that need to be propagated to another file, start the process with [p4 integrate](#). The simplest form of this command 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* doesn't yet exist, *fromFile* is copied to *toFile*, then *toFile* is opened for **branch** in the client workspace.
- If *toFile* exists, and shares a common ancestor with *fromFile* as above, then *toFile* is opened for **integrate**. You can then use [p4 resolve](#) to propagate all of, portions of, or none of the changes in *fromFile* to *toFile*.

The [p4 resolve](#) command uses *fromFile* as *theirs*, *toFile* as *yours*, and (if the resolve was scheduled with [p4 integrate](#), the closest common ancestor of *fromFile* as *base*, or (if the resolve was scheduled with [p4 merge](#)) the file with the most edits in common.

- If both *toFile* and *fromFile* exist, but *toFile* shares no common ancestor with *fromFile*, the integration is rejected.
- 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 flags modify this behavior. See the *Options* section for details.)

The process is complete when you [p4 submit toFile](#) to the depot.

To specify multiple files, use wildcards in *fromFile* and *toFile*. Any wildcards used in *fromFile* must match identical wildcards in *toFile*. Perforce compares the *fromFile* pattern to

the *toFile* pattern, creates a list of *fromFile*/*toFile* pairs, and performs an integration on each pair.

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

In most cases, you can use the `p4 merge` and `p4 copy` commands to propagate changes between streams (or branches). When `p4 integrate` schedules files for resolve, it selects the closest common ancestor as the base, and `p4 merge` selects (as the base) the revision with the most edits in common with the source and target.

Options

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

Basic Integration Flags

<code>-b branchname [toFiles...]</code>	<p>Integrate the files using the <i>sourceFile</i>/<i>targetFile</i> mappings included in the branch view of <i>branchname</i>. If the <i>toFiles</i> argument is included, include only those target files in the branch view that match the pattern specified by <i>toFiles</i>.</p> <p>If a revision range is supplied with <i>toFiles</i>, the range refers to source revisions, not target revisions.</p>
<code>-n</code>	Display the integrations this command would perform without actually performing them.
<code>-v</code>	<p>Open files for branching without copying <i>toFiles</i> into the client workspace.</p> <p>Without this flag, <code>p4 integrate</code> copies newly-branched <i>toFiles</i> into the client workspace from <i>fromFiles</i>. When the <code>-v</code> (<i>virtual</i>) flag is used, With this flag, you can save time by not copying <i>toFiles</i> to the client workspace. Instead, you can fetch them with <code>p4 sync</code> when you need them.</p>
<code>-c changelist</code>	<p>Open the <i>toFiles</i> for <i>branch</i>, <i>integrate</i>, or <i>delete</i> in the specified pending changelist.</p> <p>If this option is not provided, the files are opened in the default changelist.</p>
<code>-q</code>	<p>Quiet mode; suppresses normal output messages about the list of files being integrated, copied, or merged. Messages regarding errors or exceptional conditions are displayed.</p>

*g-opts*See the [Global Options](#) section.

Advanced Integration Flags

-b branchname -s fromFile[RevRange] [ToFiles...] In its simplest form, `p4 integrate -b branchname -s fromFile` allows you to integrate files using the source/target mappings included in the branch view of *branchname*, but include only those source files that match the patterns specified by *fromFile*.

In its more complicated form, when both *fromFile* and *toFile* are specified, integration is performed bidirectionally: first, integration is performed from *fromFile* to *toFile*; then integration is performed from *toFile* to *fromFile*.

This variation of [p4 integrate](#) was written to provide some needed functionality to graphical Perforce 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* flag 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.

-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 <i>max</i> files.
-Ob	The -Ob flag outputs the base revision for the merge (if any).
-Or	The -Or flag outputs the resolves that are being scheduled.
-Rb	The -Rb flag schedules a branch resolve instead of branching the target files automatically.
-Rd	The -Rd flag schedules a delete resolve instead of branching the target files automatically.
-Rs	The -Rs flag skips cherry-picked revisions that have already been integrated. Using this flag 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 override the configured parent and integrate to a different target stream, specify -P .

Usage Notes

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

- [p4 integrate](#) can be abbreviated as [p4 integ](#).
- *FromFiles* are often called *source files*, and *toFiles* are often called *target files*.
- Any *toFiles* that [p4 integrate](#) needs to operate on must be included in the [p4 client](#) view.
- By default, files that have been opened for **branch** or **integrate** with [p4 integrate](#) are read-only in the client workspace. You can edit these files before submitting them using [p4 edit](#) to reopen the file for **edit**.
- 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 +Sn) do not produce a lazy copy; the integrated `tempobj` file consumes additional disk space on the server.
- Two integration engines are supported in the 2013.2 release.

The current integration engine (v3) provides improved base selection semantics and expanded resolve capability, and has been improved since its introduction in 2011. *As of release 2013.2, the v3 engine is the default integration for Perforce.*

The old integration engine (v2) was introduced in 2006.1 and is still supported for backwards compatibility.

New installations of Perforce do not need to make any changes in order to take advantage of the new integration engine. Sites that wish to continue using the old (2006.1) integration logic must, as of release 2013.2, manually set the `dm.integ.engine` configurable to 2, instead of its default value of 3:

```
p4 configure set dm.integ.engine=2
```

For further information, see the Perforce Knowledge Base:

http://answers.perforce.com/articles/KB_Article/Generation-3-Integration-Engine

Examples

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

Related Commands

To create or edit a branch 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

Synopsis

Show integrations that have been submitted.

Syntax

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

Description

The [p4 integrated](#) command shows the integration history of the selected files, in the format:

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

where:

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

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

Integrate Action	What the User Did During the p4 resolve Process
delete into	<i>file</i> was integrated into <i>partner-file</i> , and <i>file</i> had been previously deleted.
edit from	<i>file</i> was integrated from <i>partner-file</i> , and <i>file</i> was edited within the p4 resolve process. This allows you to determine whether the change should ever be integrated back; automated changes (<i>merge from</i>) needn't be, but original user edits (<i>edit from</i>) performed during the resolve should be.
edit into	<i>file</i> was integrated into <i>partner-file</i> , and <i>partner-file</i> was reopened for <i>edit</i> before submission.
add from	<i>file</i> was integrated from a deleted <i>partner-file</i> , and <i>partner-file</i> was reopened for <i>add</i> (that is, someone restored a deleted file by syncing back to a pre-deleted revision and adding the file).
add into	<i>file</i> was integrated into previously nonexistent <i>partner-file</i> , and <i>partner-file</i> was reopened for <i>add</i> 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* flag is used, only files integrated from the source to target files in the branch view are shown.

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

Options

g-opts See the [Global Options](#) section.

Usage Notes

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

Related Commands

To see a list of integrations that have not yet been resolved [p4 resolve -n](#)

To view a list of integrations that have been resolved but not yet submitted [p4 resolved](#)

To perform an integration

[p4 integrate](#)

To view the actions taken for all revisions of a particular file
(including all the files from which that particular file was
integrated)

[p4 filelog \[-i\] file](#)

Synopsis

Report changes not yet integrated.

Syntax

```
p4 [g-opts] interchanges [options] fromFile[revRange] toFile
p4 [g-opts] interchanges [options] -b branchname [toFile[RevRange]...]
p4 [g-opts] interchanges [options] -b branchname -s fromFile[revRange]
    [toFile...]
p4 [g-opts] interchanges [options] -S stream -P parent [file[RevRange]]
    [toFile...]
```

options: -f -l -r -t

Description

The [p4 interchanges](#) command lists changes that have not been integrated from a set of source files to a set of target files.

Options

<i>-b branchname</i>	Use the source and target as defined by the specified branch specification.
<i>-b branchname -s fromFile[RevRange] [ToFiles...]</i>	Preview bidirectional integrations (used by Perforce applications; see p4 integrate for details.)
<i>-S stream [-P parent]</i>	Display integrations pending between the stream and its parent. To treat another stream as the parent, specify <i>-P</i> .
<i>-f</i>	List files that require integration. For partially integrated-changelists, files might be listed even if they were integrated individually.
<i>-l</i>	Long form: include full text of the changelist description
<i>-r</i>	Reverse source and target (that is, reverse the direction of the integration).

<code>-t</code>	Display full date and time that changelist was submitted. By default, only the date is displayed.
<code>g-opts</code>	See the Global Options section.

Usage Notes

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

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.)

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

Usage Notes

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

- The **-c** flag 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](#)

Synopsis

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

Syntax

```
p4 [g-opts] job [ -f ] [ jobName ]
p4 [g-opts] job -d jobName
p4 [g-opts] job -o [ jobName ]
p4 [g-opts] job -i [ -f ]
```

Description

A *job* is a written-language description of work that needs to be performed on files in the depot. It might be a description of a bug (for instance, "the scroll mechanism isn't working correctly") or an enhancement request (for instance, "please add a flag that forces a certain operation to occur") or anything else requiring a change to some files under Perforce control.

Jobs are similar to changelist descriptions in that they both describe changes to the system as arbitrary text, but whereas changelist descriptions describe completed work, jobs tell developers what work needs to be done.

Jobs are created and edited in forms displayed by [p4 job](#). The user enters the textual description of the job into the form, along with information such as the severity of the bug, the developer to whom the bug is assigned, and so on. 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 `jobNNNNNN` is created, where `NNNNNN` is a sequential six-digit number. You can change the job's name within the form before quitting the editor. If [p4 job](#) is called with a *jobname* argument, a job of that name is created; if that job already exists, it is edited.

Once a job has been created, you can link the job to the changelist(s) that fix the job with [p4 fix](#), [p4 change](#), or [p4 submit](#). When a job is linked to a changelist, under most circumstances the job's status is set to **closed**. (See the *Usage Notes* below for more information).

Form Fields

These are the fields as found in the default job form. 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 <code>jobNNNNNN</code> , where <code>NNNNNN</code> 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 created.
Description:	Writable	An arbitrary text description of the job.

Options

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

Usage Notes

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

- If the Perforce superuser has eliminated field ID# 102 (the **Status:** field) with [p4 jobspec](#), Perforce is unable to close jobs when the changelists to which they are linked are submitted. Please see the [p4 jobspec](#) page and the *Perforce System Administrator's Guide* for more information.
- After a job has been created or changed, Perforce indexes the job so that [p4 jobs-e](#) can locate the job quickly. The index keys are *word, filename* 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 jobNNNNNN .
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

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 flags and arguments. If you specify a file pattern, the jobs listed will be limited to those linked to changelists affecting particular files. The **-e** flag can be used to further limit the listed jobs to jobs containing certain words.

Jobs are listed in alphanumeric order (or, if you use the **-r** flag, in reverse alphanumeric order) by name, one job per line. The format of each line is:

```
jobname on date by user *status* description
```

The *description* is limited to the first 31 characters, unless the **-l** (long) flag is used.

If any of the **date**, **user**, **status**, or **description** fields have been removed by the Perforce superuser with [p4 jobspec](#), the corresponding value will be missing from each job's output.

To limit the list of jobs to those that have been fixed by changelists that affected particular files, use [p4 jobs filespec](#). The files or file patterns provided can contain revision specifiers or a revision range.

Options

-e <i>jobview</i>	List only those jobs that match the criteria specified by <i>jobview</i> . Please see the <i>Usage Notes</i> below for a discussion of job views.
-i <i>files...</i>	Include jobs fixed by changelists that affect files integrated into the named files.
-l	Output the full description of each job.
-m <i>max</i>	Include only the first <i>max</i> jobs, sorted alphanumerically. If used with the -r flag, the last <i>max</i> jobs are included.
-r	Display jobs in reverse alphabetical order by job name.
-R	Rebuild the job table and reindex each job. Reindexing the table is necessary either when upgrading from version 98.2 or earlier, or when upgrading from 99.1 to 2001.1 or higher and you wish to search your body of existing jobs for strings containing punctuation.
g-opts	See the Global Options section.

Usage Notes

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

Job Views

Use **p4 jobs -e *jobview*** to limit the list of jobs to those that contain particular words. You can specify that the search terms be matched only in particular fields, or anywhere in the text of the job. You can use jobviews to match jobs by values in date fields, though there are fewer options for dates than there are for text. Job fields of type **bulk** are not indexed for searching.

Text matching is case-insensitive. All alphanumeric strings (including words including embedded punctuation) separated by whitespace are indexed as words.

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

Spaces between search terms in jobviews act as boolean AND operations. To find jobs that contain any of the terms (boolean OR), separate the terms with the "|" character.

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

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

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

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

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

Additionally, you can use the NOT operator (^) to negate the sense of some comparisons. (See *Limitations* below for details).

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 backslash (\\).

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

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

If you're not sure of a field's type, run [p4 jobspec -o](#), which outputs the job specification used at your site. The [p4 jobspec](#) field called **Fields:** contains the job fields' names and datatypes. See [p4 jobspec](#) for a discussion of the different field types.

Other Usage Notes

- The [p4 user](#) form has a **JobView:** field that allows a jobview to be linked to a particular user. After a user enters a jobview into this field, any changelists he creates automatically list jobs that match the jobview in this field. The jobs that are fixed by the changelist can be left in the form, and the jobs that aren't should be deleted.

- [p4 jobs](#) sorts its output alphanumerically by job name, which also happens to be the chronological order in which the jobs were entered. If you use job names other than the standard Perforce names, this ordering may not help much.
- The `-m max -r` construct displays the last *max* jobs in alphanumeric order, not the *max* most recent jobs, but if you're using Perforce's default job naming scheme (jobs numbered like `job001394`), alphanumeric job order is identical to order by entry date.
- You can use the `*` wildcard to determine if a text field contains a value or not by checking for the jobview `"field=*"` ; any non-null value for *field* matches.
- When querying for jobs using the `-e jobview` option, be aware of your operating system and command shell's behavior for parsing, quoting, and escaping special characters, particularly when using wildcards, logical operators, and parentheses.

Limitations

- Jobviews cannot be used to search for jobs containing null-valued fields. In other words, if a field has been deleted from an existing job, then the field is not indexed, and there is no jobview that matches this "deleted field" value.
- The jobview NOT operator (^) can be used only after an AND within the jobview. Thus, the jobviews `"gui ^name=joe"` and `"gui&^name=joe"` are valid, while the jobviews `"gui|^name=joe"` and `"^name=joe"` are not.
- The `*` wildcard is a useful way of getting around both of these limitations.

For instance, to obtain all jobs without the string `"unwanted"`, query for `'job=* ^unwanted'`. All jobs will be selected by the first portion of the jobview and logically ANDed with all jobs NOT containing the string `"unwanted"`.

Likewise, because the jobview `"field=*"` matches any *non* -null value for *field*, (and the *job* field can be assumed not to be null), you can search for jobs with null-valued fields with `"job=* ^field=*"`

- You cannot currently search on space-delimited fields with conditionals. For example, instead of using `p4 jobs -e "field=word1 word2"`, you must use `p4 jobs -e "field=word1 field=word2"`.

Examples

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

<code>p4 jobs -e "gui Submitted-By=joe"</code>	List all jobs that contain the word gui in any field and the word joe in the Submitted-By: field.
<code>p4 jobs -e "gui ^Submitted-By=joe"</code>	List all jobs that contain the word gui in any field and any value <i>other than</i> joe in the Submitted-By: field.
<code>p4 jobs -e "window*"</code>	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.
<code>p4 jobs -e window.c</code>	List all jobs referring to window.c in any field.
<code>p4 jobs -e "job=* ^unwanted"</code>	List all jobs not containing the word unwanted in any field.
<code>p4 jobs -e "(fast quick)&date>1998/03/14"</code>	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 .
<code>p4 jobs -e "fast quick" //depot/proj/...</code>	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 jobview that includes jobs matching the jobview in all new changelists	p4 user

Synopsis

Edit the jobs template.

Syntax

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

Description

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

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

Form Fields

Field Name	Description
Fields:	<p>A list of field definitions for your site's jobs, one field per line. Each line is of the form <i>code name datatype length fieldtype</i>.</p> <ul style="list-style-type: none">• code: a unique integer that identifies the field internally to Perforce. The code must be between 106 and 199. Codes 101 to 105 are reserved for Perforce use; see the <i>Usage Notes</i> below for more details.• name: the name of the field. This can be changed at any time, while the code should not change once jobs have been created. <p>Field names must not contain spaces.</p> <ul style="list-style-type: none">• 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
	<ul style="list-style-type: none"> • length: recommended length for display boxes in GUI clients accessing this field. Use a value of 0 to let a Perforce application choose its own value. • fieldtype: does the field have a default value? Is it required? Is it read-only? Possible values are: <ul style="list-style-type: none"> • optional: field can take any value or be erased. • default: a default value is provided; it can be changed or erased. • required: a default value is provided; it can be changed but the user must enter a value. • once: read-only; the field value is set once to a default value and is never changed. • always: read-only; the field's value is set to a new default when the job is edited. This is useful only with the \$now and \$user variables; it allows you to change the date a job was modified and the name of the modifying user.
Values:	<p>Contains a lists of fields and valid values for select fields.</p> <p>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:</p> <p>JobType bug/request/problem.</p>
Presets:	<p>Contains a list of fields and their default values for each field that has a fieldtype of default, required, once, or always.</p> <p>Each line must contain the field name and the default value, separated by a space. For example:</p> <p>JobType bug</p> <p>Any one-line string can be used, or one of three built-in variables:</p> <ul style="list-style-type: none"> • \$user: the user who created the job • \$now: the current date • \$blank: the phrase <enter description here> <p>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.</p> <p>See the <i>Usage Notes</i> below for special considerations for field 102.</p>
Comments:	<p>Textual comments that appear at the top of each p4 job form. Each line must begin with the comment character #.</p>

Field Name	Description
	See the <i>Usage Notes</i> below for special considerations for these comments if your users need to enter jobs through P4V, the Perforce Visual Client.

Options

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

Usage Notes

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

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

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

Fields 102 through 105 are reserved for use by Perforce 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 **Preset:** 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 **Preset:** 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 **Preset:** field for field 102 (job status). To change the behavior of [p4 fix](#), [p4 change](#), and [p4 submit](#) to leave job status unchanged, use the special *fixStatus* of same.

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

Related Commands

To create, edit, or view a job	p4_job
To attach a job to a changelist	p4_fix
To list jobs	p4_jobs
To list jobs attached to specific changelists or changelists attached to specific jobs	p4_fixes

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 contain 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** parameters, or by using one of three levels of verification.

Verification levels are controlled by the **rpl.checksum.auto** configurable or the **-l level** parameter. 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, and 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 *Distributing Perforce*.

Options

-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 <i>tablee</i>	Compute checksums for all tables except those listed.
-l <i>level</i>	Specify a level for checksumming; each level corresponds to a larger set of tables. These levels correspond to the levels used by the <code>rpl.checksum.auto</code> configurable.
-u <i>filename</i> -t <i>tablename</i>	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.
-z	Compress the file when unloading a table.
-s -t <i>tablename</i>	Scan the specified database table.
-b [<i>blocksize</i>]	When scanning tables, scan <i>blocksize</i> records per block. The default is 5000. For each block, the server computes a block checksum and writes it as a journal note. Replica servers will automatically verify these blocks when processing these notes. This flag 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.
-v <i>version</i>	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.
-c <i>changeid</i>	Compute a checksum for an individual submitted changelist. The checksum is written as a journal note, and replica servers will automatically verify the checksum of the change when they process these notes.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- For more about administering Perforce in distributed or replicated environments, see *Distributing Perforce*.

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

... 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

... 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
```

Use the global -F flag 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	Unix start time of the command that ran. See type field to determine which command was executed.
--------------	--

<i>startDate</i>	Human readable form of start value.
<i>end</i>	Unix end time of the command that ran.
<i>endDate</i>	Human readable form of end value.
<i>pid</i>	The process id of the command whose execution produced this record. This value can be useful in searching for related entries in other logs.
<i>type</i>	<p>Indicates the command whose execution produced this record. Types include the following:</p> <ul style="list-style-type: none"> • <i>journal</i>: refers to the p4 -J command or the p4 admin journal command. • <i>checkpoint</i>: refers to the p4 -jc command or p4 admin checkpoint command. • <i>replay</i>: refers to the p4d -jr command • <i>schedule</i>: refers to taking a checkpoint on a replicated server. This produces two records: one for the checkpoint and another for the schedule.
<i>flags</i>	Flags passed to the command implied by the type filed.
<i>jnum</i>	The checkpoint number. A value of -1 indicates that the journal number is unknown.
<i>jfile</i>	The name of a journal or checkpoint file that was input to the command implied by the type field
<i>jdate</i>	Unix time when the filed specified by <i>jfile</i> was created.
<i>jdateDate</i>	Human readable format of <i>jdate</i> value.
<i>jdigest</i>	The digest of the checkpoint file.
<i>jsize</i>	The size of the file specified by <i>jfile</i> .
<i>jtype</i>	The type of the file specified by <i>jtype</i> .

Options

<i>-F filter</i>	List only the records that satisfy the filter expression. For instructions on constructing the filter expression, see the <i>-e</i> option to the jobviews command.
<i>-T fields</i>	Limit output to the specified fields. Separate fields using a comma or a space.
<i>-m max</i>	Limit output to the specified number of records.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Related Commands

To create a checkpoint.	p4 admin checkpoint
To create a journal.	p4 admin journal

Synopsis

Display, set, or delete a key / value pair.

Syntax

```
p4 [g-opts] key name
p4 [g-opts] key name value
p4 [g-opts] key [-d] name
p4 [g-opts] key [-i] name
p4 [g-opts] key [-m] pair list
```

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.
- The variant `p4 key -d name` deletes the specified key.
- The variant `p4 key -i name` increments a numeric key.
- The variant `p4 [g-opts] 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 '-' followed by the name. See examples below.

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

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

<code>-d name</code>	Delete key <i>name</i> from the Perforce service.
<code>-i name</code>	Increment key <i>name</i> by 1 and return the new value. This option can only be used with numeric keys.
<code>g-opts</code>	See the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	list to display a key's value; (admin if <code>dm.keys.hide</code> is set to 2) review to set a new value

Examples

<code>p4 key mykey 12</code>	Set the value of mykey to 12 . If mykey does not exist, it is created. Requires review access.
<code>p4 key mykey</code>	Display the value of mykey . If mykey does not exist, its value is displayed as 0 . Requires list access.
<code>p4 key -m mykey 5 mynewkey 4</code>	Set two keys.
<code>p4 key -m - mykey - mynewkey</code>	Delete two keys.
<code>p4 key -m mykey 6 - mynewkey</code>	Set one key; delete one key.

Related Commands

To list all keys and their values

[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

<code>-e <i>nameFilter</i></code>	List keys with a name that matches the <i>nameFilter</i> pattern, for example: <code>p4 keys -e 'mycounter-*</code>
<code>-m <i>max</i></code>	List only the first <i>max</i> keys.
<code>g-opts</code>	See the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	list (admin if <code>dm.keys.hide</code> is set to 1 or 2)

Related Commands

To view or change the value of a key

[p4 key](#)

Synopsis

Create or edit a label specification and its view.

Syntax

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

Form Fields

Field Name	Type	Description
Label:	Read-only	The label name as provided in the invoking command.
Owner:	Writable, optional	The label's owner. By default, the user who created the label. Only the owner of a label can update which files are tagged with the label.
Update:	Read-only	The date the label specification was last modified.
Access:	Read-only	The date and time the label was last accessed, either by running p4 labelsync on the label, or by otherwise referring to a file with the label revision specifier @label . (Note: Reloading a label with p4 reload does not affect the access time.)
Description:	Writable, optional	An optional description of the label's purpose.

Field Name	Type	Description
Options:	Writable	<p>Flags to control behavior and storage location of labels</p> <ul style="list-style-type: none"> • 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 <code>db.label</code>, 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	<p>An optional revision specification for an automatic label.</p> <p>If you use the <code>#</code> character to specify a revision number, you must use quotes around it in order to ensure that the <code>#</code> is parsed as a revision specifier, and not as a comment field in the form.</p>
View:	Writable	<p>A list of depot files that can be tagged with this label. No files are actually tagged until p4 labelsync is invoked.</p> <p>Unlike client views or branch views, which map one set of files to another, label views consist of a simple list of depot files. Please see the Views chapter for more information.</p>
ServerID:	Writable, optional	<p>If set, restricts usage of the label to the named server. If unset, this label may be used on any server.</p>

Options

-d [-f]	Delete the named label if it's unlocked . The -f flag forces the deletion even if the label is locked . (Deleting a locked label requires admin or super access.)
-i	Read the label definition from standard input without invoking the editor.
-o	Write the label definition to standard output without invoking the editor.
-f	Allow the Update: field's date to be set. Can be used with either the -i flag or the -t flag for the same purpose.
-g	In distributed environments, use the -g flag to control whether the label is local to an edge server, or globally available from the commit server.
-t <i>template</i>	Copy label <i>template</i> 's view and options into the View: and Options: fields of this label. You can specify a default label template using the <code>template.label</code> configure variable. If you do so, you do not have to specify this option.
g-opts	See the Global Options section.

Usage Notes

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

N/A	N/A	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** flag 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** flag 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

Synopsis

Display list of defined labels.

Syntax

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

-t	Display the time as well as the date of the last update to the label.
-u user	List only labels owned by <i>user</i> .
-e filter	List only labels matching <i>filter</i> (case-sensitive).
-E filter	List only labels matching <i>filter</i> (case-insensitive).
-m max	List only the first <i>max</i> labels.
-U	List labels in the unload depot. For details, see p4 unload .
-a	List all labels, not just labels bound to this server. This flag may not be used with a file specification.
-s serverID	List only those labels bound to the specified <i>serverID</i> . This flag may not be used with a file specification.

g-opts See the [Global Options](#) section.

Usage Notes

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

- To see a list of files tagged by a particular label, use [p4 files](#) @*labelname*.
- 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* flags.

Examples

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

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 flag 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 <i>labelname</i>	Specify the label to be applied to file revisions
-n	Display what p4 labelsync would do without actually performing the operation.
-q	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
g-opts	See the Global Options section.

Usage Notes

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

Yes	Yes	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. The `-g` flag 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` flag 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` flag 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

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`.

Without a valid license, the versioning service limits itself to either 20 users and 20 client workspaces (and unlimited files), or to an unlimited number of users and workspaces (but with a limit of 1000 files).

Options

-o	Display the current license file on the standard output.
-i	Read in a new license file from the standard input.
-u	Report license limits and show how many entities (users or files) are in use with respect to these limits.
g-opts	See the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	super (admin for <code>-u</code> flag)

Examples

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

Synopsis

Create a temporary list of files that can be used as a label.

Syntax

```
p4 [g-opts] list [ -l labelname ] [ -C clientworkspace ] [ -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

-l <i>labelname</i>	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.
-C <i>clientwork</i>	List only the files mapped through the specified <i>clientworkspace</i> .
-M	When run against a forwarding replica, forward the p4 list command to the master server.
-d <i>labelname</i>	Delete the the specified list.
g-opts	See the Global Options section.

Usage Notes

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

Synopsis

Lock an opened file against changelist submission.

Syntax

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

Description

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

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

Options

-c <i>changelist</i>	Lock only files included in changelist <i>changelist</i>
-----------------------------	--

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

Usage Notes

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

Related Commands

To unlock locked files	p4 unlock
------------------------	---------------------------

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

Synopsis

Report lock status of database tables.

Syntax

```
p4 [g-opts] lockstat [-c client | -C]
```

Description

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

Options

<code>-c <i>client</i></code>	Report whether or not the specified client workspace is locked for a read or write operation.
<code>-C</code>	Report all client workspaces that are locked for read/write operations.
<code>g-opts</code>	See the Global Options section.

Usage Notes

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

Synopsis

Add a line to any user log files.

Syntax

```
p4 [g-opts] logappend [ -a args ... ]
```

Description

The `p4 admin 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 the [Global Options](#) section.

Usage Notes

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

- At present, the only two log files that include user log events are `user.csv` and `all.csv`.

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

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

Usage Notes

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

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

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 ]
```

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.

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.

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.
g-opts	See the Global Options section.

Usage Notes

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

N/A	N/A	list
-----	-----	------

- The default timeout value of 43200 seconds (12 hours) is defined on a per-group basis in the [p4 group](#) form.
- To create tickets that do not expire, set the timeout value to **unlimited** in the [p4 group](#) form.
- 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` flag causes the service to issue a ticket that is valid on the specified host IP address. This flag 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

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

Related Commands

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

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** option.

Options

- | | |
|-----------------------|---|
| -a <i>user</i> | Log out all users of the ticket by invalidating the ticket on the service. If you specify a user name, that user is logged out. You must have super user access to be able to log out a user other than yourself. |
| <i>g-opts</i> | See the Global Options section. |

Usage Notes

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

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

Synopsis

Parse a structured log file and return data.

Syntax

```
p4 [g-opts] [-e] [-T fields...] [-F filter] [-s offset] [-m max] logfile
```

Description

The `p4 admin logparse` command parses the indicated structured *logfile* and returns the log data in tagged format.

Valid names for structured log files are:

<code>all.csv</code>	All loggable events (commands, errors, audit, etc...)
<code>commands.csv</code>	Command events (command start, command compute, command end)
<code>errors.csv</code>	Error events (errors-failed, errors-fatal)
<code>audit.csv</code>	Audit events (audit, purge)
<code>track.csv</code>	Command tracking (track-usage, track-rpc, track-db)
<code>user.csv</code>	User events; one record every time a user runs p4 logappend .
<code>events.csv</code>	Server events (startup, shutdown, checkpoint, journal rotation, etc.)
<code>integrity.csv</code>	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. To enable all recording of all eight log types, set the following configurables:

```
p4 configure set serverlog.file.1=all.csv
p4 configure set serverlog.file.2=commands.csv
p4 configure set serverlog.file.3=errors.csv
p4 configure set serverlog.file.4=audit.csv
p4 configure set serverlog.file.5=track.csv
p4 configure set serverlog.file.6=user.csv
p4 configure set serverlog.file.7=events.csv
p4 configure set serverlog.file.8=integrity.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

<code>-e</code>	Display special characters as hex-encodings
-----------------	---

<code>-T field,field</code>	Limit displayed fields to those listed
<code>-F filter</code>	limits output to records that match the filter pattern
<code>-s f_offset</code>	Starts parsing at the given file offset as returned in the <code>f_offset</code> field
<code>-m max</code>	Limit the number of lines returned
<code>g-opts</code>	See the Global Options section.

Usage Notes

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

- Structured logs differ from the basic error log ([P4LOG](#)) and audit log ([P4AUDIT](#)). To read the basic error log, use the [p4 logtail](#) command.

Synopsis

Rotate one or more structured log files.

Syntax

```
p4 [g-opts] logrotate [-l logname]
```

Description

The `p4 admin logrotate` command rotates the named logfile, or rotates all structured logs if the `-l logname` flag 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 the [Global Options](#) section.

Usage Notes

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

Synopsis

Describe the schema of a log record type.

Syntax

```
p4 [g-opts] logschema [ -a | recordtype ]
```

Description

The `p4 admin 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 the [Global Options](#) section.

Usage Notes

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

Synopsis

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

Syntax

```
p4 [g-opts] logstat [-l logname]
```

Description

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

By default, [p4 logstat](#) does not include the sizes of structured log files. To report the size of a structured log file, you must explicitly specify the log file's name (as defined by its corresponding `serverlog.file.N` configurable) with the `-l` flag.

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.

`g-opts` See the [Global Options](#) section.

Usage Notes

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

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

Options

-s start	The offset (from the beginning of the file), in bytes.
-b blocksize	The block size, in bytes. The default is 8192 bytes.
-m maxBlocks	The maximum number of blocks to output. Ignored unless -s is also specified.
-l log	If specified, the name of the log to display.
g-opts	See the Global Options section.

Usage Notes

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

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

Examples

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

Synopsis

Merge one set of files into another.

Syntax

```
p4 [g-opts] merge [options] fromFile[revRange] toFile
p4 [g-opts] merge [options] -b branch [-r] [toFile[revRange] ...]
p4 [g-opts] merge [options] -b branch -s fromFile[revRange] [toFile ...]
p4 [g-opts] merge [options] -S stream [-P parent] [-F] [-r]
  [toFile[revRange] ...]
```

options : -c *change* -n -O*flags* -m *max* -q

Description

The [p4 merge](#) command is a simplified form of the [p4 integrate](#) command that merges a set of changes from source to target files. The command outputs the scheduled resolves. This command is intended for use with streams, but is also perfectly usable for traditional Perforce branches.

With streams, you use [p4 merge](#) to keep a child stream up to date with a more stable parent stream, to ensure 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.

In most cases, you can use the [p4 merge](#) and [p4 copy](#) commands to propagate changes between streams (or branches). The default behavior of [p4 integrate](#) is to schedule files for resolve by selecting the closest common ancestor as the base; [p4 merge](#) selects (as the base) the revision with the most edits in common with the source and target.

Options

-c <i>change</i>	Specifies an existing pending changelist in which the files are to be opened.
-n	Preview the merge.
-m <i>max</i>	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.
-b <i>branch</i>	Specifies a branch view to be used to determine source and target files.

-r [<i>toFile</i> [<i>revRange</i>] ...]	Reverses the mappings in the branch view, integrating from the target files to the source files. Requires the -b flag.
-s <i>fromFile</i> [<i>revRange</i>] [<i>toFile</i> ...]	Treat <i>fromFile</i> 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 flag, if specified. Requires -b .
-S <i>stream</i>	Specifies the source stream. Files are merged to its parent stream. To merge from parent to child, specify the -r flag, which reverses the direction.
-P <i>parent</i>	Merge to the specified parent stream. Overrides the parent defined in the source stream specification.
-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 .
-Ob	The -Ob flag outputs the base revision for the merge (if any).
-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 the Global Options section.

Usage Notes

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

Examples

```
p4 merge -r -S //projectX/bruno_dev
p4 resolvep4 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

Synopsis

Display Perforce process information

Syntax

```
p4 [g-opts] monitor show [ -a ] [ -e ] [ -l ] [-s R|T|P|I ]
p4 [g-opts] monitor terminate id
p4 [g-opts] monitor clear id
p4 [g-opts] monitor clear all
p4 [g-opts] monitor pause id
p4 [g-opts] monitor resume id
```

Description

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

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

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

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` (where *status* refers to Running, Terminated, Paused, or Idle processes) to restrict the display to processes in the specified state.

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

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

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

Long-running tasks such as [p4 verify](#) or [p4 pull](#) can be controlled with the `p4 monitor pause` and `p4 monitor resume` commands.

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

Options

-a	Show all arguments associated with the process (for example, <code>edit file.c</code> , or <code>sync -f //depot/src/...</code>). Perforce user names are truncated to 10 characters, and each line is limited to a total of 80 characters of output.
-e	Show environment information including invoking Perforce application (if known), host IP address, and workspace name.
-l	Show all arguments in long form; that is, without truncating user names or the list of command line arguments.
-s <i>status</i>	Restrict the display to processes in the Running, Terminated, Paused, or Idle states.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

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

Examples

<code>p4 monitor show</code>	Show Perforce processes information (commands only). Requires list access only.
<code>p4 monitor show -l</code>	Show arguments and commands, without limits on line length. Requires super access.
<code>p4 monitor show -a</code>	Show arguments and commands, limited to 80 characters per line of output. Requires super access.
<code>p4 monitor terminate 123</code>	Instruct the Perforce service to mark process 123 for termination. Requires super access.
<code>p4 monitor clear all</code>	Clears the monitor table of all entries. Requires super access.

Related Commands

To turn on monitoring	p4 configure set monitor=1
To turn off monitoring	p4 configure set monitor=0

Synopsis

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

Syntax

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

Description

The [p4 move](#) command takes a file already opened for edit or add and moves it to the destination provided.

An open file can be moved many times before it is submitted; moving a file back to its original location undoes the pending move, leaving it open for edit. Using [p4 revert](#) on a moved file both undoes the move and reverts the unsubmitted content.

Options

-c <i>change</i>	If a changelist number is provided, the files are opened in the numbered pending changelist.
-t <i>filetype</i>	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 the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
<i>fromFile</i> : Yes <i>toFile</i> : No	No	read access for <i>fromFile</i> write access for <i>toFile</i>

- 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**.

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 the 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** flag is required; without it, [p4 obliterate](#) merely reports what it would do without actually performing the obliteration.

If you specify a single revision (for instance, **p4 obliterate file#3**), only that revision of the file is obliterated. If you specify a revision range (for instance, **p4 obliterate file#3,5**), only the revisions in that range are obliterated.

Options

-y <i>filespec</i>	Perform the obliterate operation. Without this flag, 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 flag 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 flag with -b .

-a	Skip the (potentially resource-intensive) search of <code>db.archmap</code> 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 <code>db.have</code> 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 the Global Options section.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
Yes	Yes	admin
<ul style="list-style-type: none"> <code>p4 obliterate</code> 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 (<code>erase</code>, <code>rm</code>, 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 <code>p4 archive</code> and <code>p4 restore</code> 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 <code>p4 filelog</code> after obliterating revisions #2 and #3 might look like this: <pre>... #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'</pre> 		

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](#)

Synopsis

List files that are open in pending changelists.

Syntax

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

If file specifications are provided as arguments to [p4 opened](#), only those files that match the file specifications are included in the report.

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

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

where:

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

Options

-a List opened files in all client workspaces.

	In distributed environments, this option lists only those files opened in other workspaces on your edge server; files opened on other edge servers do not appear.
-c <i>change</i>	List the files in pending changelist <i>change</i> . To list files in the default changelist, use p4 opened -c default .
-C <i>workspace</i>	List only files that are open in the specified client <i>workspace</i> .
-u <i>user</i>	List only those files that were opened by <i>user</i> .
-m <i>max</i>	List only the first <i>max</i> 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.
-x	In distributed environments, list all files that have the +1 filetype (exclusive open) over all servers.
g-opts	See the Global Options section.

Usage Notes

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

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

- Locked files appear in the output of [p4 opened](#) with an indication of ***locked***. On UNIX, you can find all locked files you have open with the following command:

```
p4 opened | grep "*locked*"
```

This lists all open files you have locked with [p4 lock](#).

- **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](#)

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](#) flag. 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](#) flag 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 sitewide) 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 *System Administrator's Guide*.

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 *System Administrator's Guide*.

Options

-O <i>oldpassword</i>	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.
-P <i>newpassword</i>	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 the Global Options section.

Usage Notes

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

- Passwords can be up to 1024 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

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 <i>count</i>	Number of messages per test.
-t <i>transmittime</i>	Transmit data for <i>transmittime</i> (maximum 6000) seconds.
-i <i>iterations</i>	Repeat the test for the specified number of <i>iterations</i> .
-f	Flood mode: the service transmits continuously, sending the next message without waiting for the Perforce application to confirm receipt of the prior message.
-p <i>pausetime</i>	Pause for <i>pausetime</i> seconds between tests, up to 120 seconds. To disable pausing, specify a pausetime of 0.
-s <i>sendsize</i>	Size of the service-to-user message, up to a maximum value of 10,000,000 bytes.
-r <i>receivesize</i>	Size of the user-to-service message, up to a maximum value of 100,000 bytes.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- Like the operating system's counterpart, [p4 ping](#) can flood the network with traffic.

Synopsis

Branch a set of files as a one-step operation.

Syntax

```
p4 [g-opts] populate [options] fromFile[rev] toFile
p4 [g-opts] populate [options] -b branch [-r] [toFile[o] ...]
p4 [g-opts] populate [options] -b branch -s fromFile[rev] [toFile ...]
p4 [g-opts] populate [options] -S stream [-P parent] [-r]
    [toFile[rev]...]
```

Options: -d *description* -f -m *max* -n

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.

Options

-S stream	Use a stream's view; the source is the stream itself, and the target is the stream's parent.
-P parent	When used with -S stream , specify a parent stream other than the stream's actual parent.
-b branch	Use the view in a user-defined <i>branch</i> specification; the source is the left side of the branch view and the target is the right side of the branch view.
-r	Reverse direction of integration (from target to source, rather than from source to target)
-s	If used with -b branch , treat <i>fromFile</i> as the source, and use both sides of the user-defined branch view as the target. (Optional <i>toFile</i> arguments may be given to further restrict the scope of the target file set.) The -r argument is ignored when -s is used.
-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.)
-n	Preview the operation without actually doing anything
-o	Display a list of files created by the p4 populate command
-m max	Limit the action to the first <i>max</i> files.
g-opts	See the Global Options section.

Usage Notes

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

Synopsis

Print the contents of a depot file revision.

Syntax

```
p4 [g-opts] print [-a -A -k -o outfile -q -m max -U] file[revRange] ...
```

Description

The [p4 print](#) command writes the contents of a depot file to standard output. A revision range can be included; in this case, only the files with revisions in the specified range are printed, and by default, only the highest revision in that range is listed. (To output each file at every revision within a specified revision range, use [p4 print -a](#).) 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) flag.

By default, RCS keywords are expanded. To suppress keyword expansion, use the [-k](#) (keyword) flag.

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.
-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.
-m max	Print only the first <i>max</i> files.
-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 the [Global Options](#) section.

Usage Notes

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

- Because most terminals are unable to display UTF16 content, the default behavior of the [p4 print](#) command is to return UTF8 content. You can override this behavior by bypassing terminal output entirely and specifying an output file, for example:

```
p4 print -q -o outputfile //depot/file
```

If your terminal supports UTF16 output, specify standard output as the output file:

```
p4 print -q -o - //depot/ file
```

- [p4 print](#)'s file arguments can take a revision range. By default, only the highest revision matched by any particular file is printed (that is, when no range is specified, the implied range is **#1**, **#head**, and the highest revision is **#head**). To print all files in a specified (or implied) range, use the **-a** option.
- Because [p4 print](#)'s output can be quite large when called with highly non-restrictive file arguments (for instance, `p4 print //depot/...` prints the contents of all files in the depot), it may be subject to a **maxresults** limitation as set in [p4 group](#).
- In many cases, redirecting [p4 print](#)'s output to a file via your OS shell will suffice.

The **-o** option is intended for users who require the automatic setting of file type and/or permission bits. This is 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

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.
-d ...	Delete a property value. Requires admin access.
-l ...	List one or more property values. Performance is substantially improved when you supply a -n name argument.
-n name	Specify the name of the property.
-v value	Specify the value of the property.
-s sequence	Specify the sequence number of the property. If not specified, the default value is 1.

-u user	Specify the user to whom this property applies.
-g group	Specify the user group to which this property applies.
-m max	Limit output to the first max matching properties.
-T taglist	Limit the fields that are displayed to the fields listed in <i>taglist</i> . Syntax is that used by p4 fstat .
-F filter	Limit the properties displayed to those that match the <i>filter</i> pattern. Syntax is that used by p4 fstat .
-A	List properties for all users and groups, as well as the property sequence number of each property value. Requires <code>admin</code> access.
g-opts	See the Global Options section.

Usage Notes

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

- Perforce administrators can use [p4 property](#) to centrally manage P4V's performance settings and selectively enable/disable features.

See the *System Administrator's Guide* for details.

Synopsis

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

Syntax

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

Description

Use [p4 protect](#) to control Perforce permissions. You can use [p4 protect](#) to:

- Control which files particular users can access
- Manage which commands particular users are allowed to use
- Combine the two, allowing one user to write one set of files but only be able to read other files
- Grant permissions to groups of users, as defined with [p4 group](#)
- Grant or deny specific 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 these IP addresses can run Perforce.

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

The permission levels and access rights are described in the table below:

Permission Level / Right	What the User Can Do
<code>list</code>	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.
<code>read</code>	The user can do everything permitted with <code>list</code> access, and also run any command that involves reading file data, including p4 print , p4 diff , p4 sync , and so on.
<code>=read</code>	If this right is denied, users cannot use p4 print , p4 diff , or p4 sync on files.
<code>open</code>	This gives the user permission to do everything she can do with <code>read</code> 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.

Permission Level / Right	What the User Can Do
=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. 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.

Form Fields

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

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

Column	Description
	<p>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.</p> <p>How the system forms host addresses depends on the setting of the dm.proxy.protects variable. By default, this variable is set to 1. This means that if the client host uses some intermediary (proxy, broker, replica) to access the server, the proxy- prefix is prepended to the client host address to indicate that the connection is not direct. If you specify proxy-* for the Host field, that will affect all connections made via proxies, brokers, and replicas. A value like proxy-10.0.0.5 identifies a client machine with an IP address of 10.0.0.5 that is connected to the server through an intermediary.</p> <p>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.</p>
Depot File Path	<p>The depot file path this permission is granted on, in Perforce depot syntax. The file specification can contain Perforce wildcards.</p> <p>To exclude this mapping from the permission set, use a dash (-) as the first character of this value.</p> <p>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.</p>

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

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

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

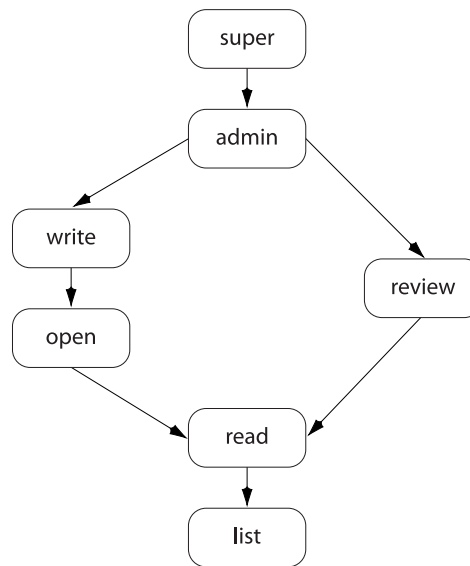
Options

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

Usage Notes

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

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



- The specific rights of `=read`, `=open`, `=write`, and `=branch` can be used to override the automatic inclusion of lower access levels. This makes it possible to deny individual rights without having to then re-grant lesser rights.

For example, if you want administrators to have the ability to run administrative commands, but to deny them the ability to make changes in certain parts of the depot, you could set up a permissions table as follows:

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

In this example, user **joe** can perform administrative functions, 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.

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

Command	Access Level	Notes
add	open	
admin	super	
annotate	read	
archive	admin	
attribute	write	The -f flag to set the attributes of submitted files requires admin access.
branch	open	The -f flag to override existing metadata or other users' data requires admin access.
branches	list	
change	open	The -o flag (display a change on standard output) requires only list access. The -f flag to override existing metadata or other users' data requires admin access.
changes	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
client	list	The -f flag to override existing metadata or other users' data requires admin access.
clients	list	
configure	super	
copy	list	list access to the source files; open access to the destination files.
counter	review	list access to at least one file in any depot is required to view an existing counter's value;

Command	Access Level	Notes
		review access is required to change a counter's value or create a new counter.
counters	list	
cstat	list	
dbschema	super	
dbstat	super	
dbverify	super	
delete	open	
depot	super	The -o flag to this command, which allows the form to be read but not edited, requires only list access.
depots	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
describe	read	The -s flag to this command, which does not display file content, requires only list access.
diff	read	
diff2	read	
dirs	list	
diskspace	super	
edit	open	
export	super	
filelog	list	
files	list	
fix	open	
fixes	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
flush	list	
fstat	list	

Command	Access Level	Notes
grep	read	
group	super	<p>The -o flag to this command, which allows the form to be read but not edited, requires only list access.</p> <p>The -a flag to this command requires only list access, provided that the user is also listed as a group owner.</p> <p>The -A flag requires admin access.</p>
groups	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
have	list	
help	none	
info	none	
integrate	open	The user must have open access on the target files and read access on the source files.
integrated	list	
interchanges	list	
istat	list	
job	open	<p>The -o flag to this command, which allows the form to be read but not edited, requires only list access.</p> <p>The -f flag to override existing metadata or other users' data requires admin access.</p>
jobs	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
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 <code>dm.keys.hide</code> configurable is set to 2.
keys	list	admin access is required if the <code>dm.keys.hide</code> configurable is set to 1 or 2.
label	open	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot. The <code>-f</code> flag to override existing metadata or other users' data requires admin access.
labels	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
labelsync	open	
license	super	The <code>-u</code> flag, 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	
monitor	list	super access is required to terminate or clear processes, or to view arguments.

Command	Access Level	Notes
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 <i>-a</i> , <i>-g</i> , and <i>-u</i> flags.
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 <i>-f</i> to reload other users' workspaces and labels.
reopen	open	
replicate	super	
resolve	open	
resolved	open	
restore	admin	
revert	list	
review	review	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.
reviews	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

Command	Access Level	Notes
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 <code>p4 shelve -f -d</code>
sizes	list	
status	open	
stream	open	
streams	list	
submit	write	
sync	read	
tag	list	
tickets	none	
triggers	super	
typemap	admin	The <code>-o</code> flag 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 <code>p4 unload -f</code> to unload other users' workspaces and labels.
unlock	open	The <code>-f</code> flag to override existing metadata or other users' data requires admin access.
unshelve	open	
update	list	
user	list	<p>This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.</p> <p>The <code>-f</code> flag (which is used to create or edit users) requires super access.</p>
users	list	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

Command	Access Level	Notes
		If the <code>run.users.authorize</code> configurable is set to 1, you must also authenticate yourself to the server before you can run p4 users .
<code>verify</code>	<code>admin</code>	
<code>where</code>	<code>list</code>	This command doesn't operate on specific files. Permission is granted to run the command if the user has the specified access to at least one file in any depot.

- At 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.
- In the course of normal operation, you'll primarily grant users `list`, `read`, `write`, and `super` access levels. The `open` and `review` access levels are used less often.
- Those commands that list files, such as [p4 describe](#), will only list those files to which the user has at least `list` access.
- Some commands (for instance, [p4 change](#), when editing a previously submitted changelist) take a `-f` flag that requires `admin` or `super` access.
- The `open` access level gives the user permission to change files but not submit them to the depot. Use this when you're temporarily freezing a codeline, but don't want to stop your developers from working, or when you employ testers who are allowed to change code for their own use but aren't allowed to make permanent changes to the codeline.
- The `review` access level is meant for review daemons that need to access counter values.
- If you write a review daemon that requires both `review` and `write` access, but shouldn't have `super` access, grant the daemon both `review` and `write` access on two separate lines of the protections table.
- To limit or eliminate the use of the files on a particular server as a remote depot from another server (as defined by [p4 depot](#)), create protections for user `remote` (or for the service user by which the other server authenticates itself). Remote depots are accessed either by the service user associated with the user's Perforce service, or by a virtual user named `remote`.
- For further information, see the *Protections* chapter of the *System Administrator's Guide*.

Examples

Suppose that user `joe` is a member of groups `devgroup` and `buggroup`, as set by [p4 group](#), 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 isn't excluded by any subsequent lines.
10.14.10.1	p4 print //depot/proj/README	Fails. The third line removes all of Joe's permissions on any files in this directory. (If the second protection and the third protection had been switched, then the subsequent protection would have overridden this one, and Joe would have succeeded).
192.168.100.123	p4 print //depot/proj/README	Succeeds. Joe'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 doesn't 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](#)

Synopsis

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

Syntax

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

Description

Use the [p4 protects](#) command to display the lines from the protections table that apply to a user, group, or set of files.

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

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

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

Options

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

Usage Notes

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

- 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](#)

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.

Options

g-opts See the [Global Options](#) section.

Usage Notes

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

- 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](#)

Synopsis

Retrieve metadata or versioned files from a Perforce master server to a replicate.

Syntax

```
p4 [g-opts] pull [-J prefix] [-i n] [-b n] [-T tableexcludelist]
    [-P serverid]
p4 [g-opts] pull -u [ -i n ]
p4 [g-opts] pull -l [ -s | -j [ -J prefix ] ]
p4 [g-opts] pull -d -f file -r revision
```

Description

The [p4 pull](#) command instructs the current replica server to retrieve either journal records or file contents from a target server specified by [P4TARGET](#).

A replica server can run multiple [p4 pull](#) commands against the same master server. To replicate both metadata and file contents, you must run two [p4 pull](#) commands simultaneously: one [p4 pull](#) (without the `-u` flag) to replicate the master server's metadata, and at least one [p4 pull](#) (with the `-u` flag) to replicate the server's versioned files.

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

The `-b` flag specifies a wait time after a failed pull attempt. If `-b` is not specified, [p4 pull](#) retries every 60 seconds.

Use the `-u` flag to retrieve file contents instead of journal records.

Use the `-l` flag to display a list of pending file content transfers. If `-s` is specified along with `-l`, a summary of pending file transfers is displayed:

```
File transfers: n active/m total, bytes: nnn active/mmmmm total.
```

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
```

If your master server uses a non-default journal location, use the `-J prefix` flag or `journalPrefix` configurable to specify the journal file location on the master server.

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.

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](#).

Options

-u	Transfer archive files instead of journal records. If you omit this option, the command retrieves journal records.
-l	List pending file content transfers
-l -s	Display a summary of pending file content transfers
-l -j	Display the current journal state on the replica and the master
-d -f file -r rev	<p>Cancel a pending file content transfer, where <i>file</i> and <i>rev</i> refer to a depot file and a specific revision.</p> <p>Note that this is not the normal Perforce file and revision data, but rather than archive file and revision. Use the p4 pull -l command to get the correct file name and revision.</p>
-i interval	<p>Specify a polling interval for content retrieval. The smallest interval is one second. If you omit this option, the command runs once and exits.</p> <p>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.</p>
-b interval	Specify a polling interval for retries after failed retrieval attempts. The default value is 60 seconds.
-J prefix	Specify a prefix for the journal file; overrides journalPrefix configurable.
-T tableexcludelist	<p>Supply a list of database tables (for example, db.have and db.client) to exclude from the replica's journal records.</p> <p>To specify multiple tables, double-quote the list and separate the table names with spaces. The table names must begin with "db.", following the naming convention used for database files in the server root directory. Table names can also be separated by commas. For example, -T db.have,db.client or -T "db.have db.client"</p>
-P serverid	<p>Filter data from <i>serverid</i> according to the ArchiveDataFilter: and ClientDataFilter: and RevisionDataFilter: fields in the specified server's p4 server form.</p> <p>The p4 pull command now defaults to -P serverid if the replica has a serverid and no -P flag has already been specified.</p>

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.

g-opts

See the [Global Options](#) section.

Usage Notes

Can File Arguments Use Revision Specifier?

N/A

Can File Arguments Use Revision Range?

N/A

Minimal Access Level Required

super

- In most situations, server replication with [p4 pull](#) is preferable to [p4 replicate](#).
- Except for testing purposes, [p4 pull](#) is rarely run from the command line. Instead, use the [p4 configure](#) command to configure the server to start the [p4 pull](#) processes every time the replica server starts.
- 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.
- For more about configuring Perforce to run in a replicated environment, see *Distributing Perforce*.

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](#)

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.

Syntax

```
p4 [g-opts] reconcile [-c changelist] [-e -a -d -f -I -l -n -m -w] [file ...]
```

Description

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:

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.)

To limit the scope of [p4 reconcile](#) to add, edit, or delete, use the **-a**, **-e**, or **-d** flags.

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

To improve performance when reconciling changes to large files, the **-m** flag 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** flag to override this behavior and ignore the [P4IGNORE](#) file.

Options

-c <i>changelist</i>	Open the files to be reconciled in the specified pending changelist. If you omit this argument, the files are opened in the default changelist.
-e	Edit files: Find files in the workspace that have been modified outside of Perforce, and open them for edit.
-a	Add files: Find files in the workspace that are not under Perforce control and open them for add.
-d	Delete files: Find files missing from the workspace, but present in the depot; open these files for delete, but only if these files are in the user's have list.

-n	Preview the results of the operation without performing any action.
-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 .
-l	Display output in local file syntax with relative paths, similar to the workspace-centric view of p4 status .
-m	<p>Compare the file submit time (in the depot) with the file modification time (in the workspace) to determine whether the file has changed.</p> <p>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.</p> <p>You can use the -m option only if the workspace was last synced with the client 'modtime' set or if the files were synced with the file +m type. This option is only relevant if you are using reconcile to find changed files rather than files that were deleted or added.</p> <p>For more information, see http://answers.perforce.com/articles/KB_Article/%0A%0AFile-Modification-Times.</p>
-w	<p>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.</p> <p>The use of p4 reconcile with this flag is the same as using the p4 clean command. For information on the use of other flags when using p4 reconcile with the -w flag, please see the description of the p4 clean command. The same flags can be used with both.</p> <p>This option requires read permission.</p>
g-opts	See the Global Options section.

Usage Notes

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

- The [p4 reconcile](#) command produces output in depot syntax. To see file names and paths in local syntax, you must either use the **-l** flag 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` flag 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](#)

Synopsis

Reloads the specified workspace, label, or task stream from the unload depot.

Syntax

```
p4 [g-opts] reload [-f] [-c workspace | -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` flag to reload workspaces and/or labels owned by other users.

Options

<code>-c workspace</code>	Reload the specified client workspace from the unload depot.
<code>-l label</code>	Reload the specified label from the unload depot.
<code>-s stream</code>	Reload the specified task stream from the unload depot.
<code>-f</code>	Administrator force flag; allows reloading of labels and workspaces owned by other users. Requires <code>admin</code> access.
<code>-p address</code>	In distributed environments, the <code>-p</code> flag can be used to reload an unloaded client workspace from the remote Edge Server specified by <code>address</code> , 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.
<code>g-opts</code>	See the Global Options section.

Usage Notes

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

- 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](#)

Synopsis

Renaming files under Perforce.

Syntax

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

Description

The command [p4 rename](#) is an alias for [p4 move](#).

You can rename multiple files by including matching wildcards in *fromFile* and *toFile*.

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
<i>fromFile</i> : Yes <i>toFile</i> : No	No	read access for <i>fromFile</i> write access for <i>toFile</i>

- Files must be open for edit before they can be moved.
- Prior to release 2009.1, moving files was accomplished by using [p4 integrate](#) to copy *fromFile* into a new *toFile*, using [p4 delete](#) to delete *fromFile*, and then using [p4 submit](#) to store these file changes in the depot.

Examples

```
p4 edit //depot/d1/...
```

Moving files from d1 to d2.

```
p4 move //depot/d1/... //depot/d2/...
```

```
p4 integ //depot/d1/... //depot/d2/...
```

Moving a set of files prior to 2009.1:

```
p4 delete //depot/d1/...
```

```
p4 submit
```

- [p4 integrate](#) copies all the files in the d1 directory to the d2 directory.
- [p4 delete](#) deletes all files in d1.
- [p4 submit](#) sends these changes to the depot in a single changelist.

Related Commands

Moving a file

[p4 move](#)

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

Synopsis

Rename a user and modify all database records that mention the user.

Syntax

```
p4 [g-opts] renameuser --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.

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.

- 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.

g-opts See the [Global Options](#) section.

Usage Notes

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

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

Synopsis

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

Syntax

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

Description

[p4 reopen](#) has two different but related uses:

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

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

Options

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

Usage Notes

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

Examples

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

Reopen all open files as text files with keyword expansion.

<code>p4 reopen -c 410 //depot/proj1/... //.../README</code>	Move all open files under directory <code>//depot/proj1</code> or that are named <code>README</code> to pending changelist <code>410</code> .
<code>p4 reopen -c default -t binary+S //....exe</code>	Move all open <code>.exe</code> 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 <i>changelist</i>
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 <i>changelist</i> -t <i>type</i>
To open a file for add under a particular pending changelist and as a particular type	p4 add -c <i>changelist</i> -t <i>type</i>
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 <i>depotname</i> .	p4 edit -t +l <i>//depotname/...</i>

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

-j <i>token</i>	Specify a journal number or position token of the form <i>journalnum/byteoffset</i> from which to start replicating metadata. If this flag is specified, it overrides any state file specification.
-s <i>statefile</i>	Specify a state file which tracks the most recent journal position. You can also use the statefile configurable to specify the state file.
-i <i>interval</i>	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 <i>interval</i> of 0.
-J <i>prefix</i>	Specifies a filename prefix for the journal, such as that used with p4d -jc <i>prefix</i>
-T <i>tables</i>	Supply a list of database tables (for example, db.have) to exclude from export.
-k	Keep the pipe to the <i>command</i> subprocess open between polling intervals.
-x	Exit the p4 replicate command when journal rotation is detected.
-R	The -R flag causes p4 replicate to attempt reconnection to the server in the event of connection loss or serious error. A polling interval must be specified with -i <i>interval</i> .
-o <i>savefile</i>	Specify a file for output. If a <i>command</i> subprocess is specified, both the subprocess and the specified savefile are provided with the output.
g-opts	See the Global Options section.

Usage Notes

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

N/A	N/A	super
-----	-----	-------

- Use [p4 replicate](#) in situations where you need to replicate 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 *Distributing Perforce* 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
--	---------------------------

Synopsis

Resolve conflicts between file revisions.

Syntax

```
p4 [g-opts] resolve [-Aflags -dflags -aflags -f -n -N -o -t -v -cchange] [file...]
```

Description

Use [p4 resolve](#) to combine the contents of two files or file revisions into a single file revision in your workspace. Two situations require the use of [p4 resolve](#) before a file can be submitted:

- When a simple conflict exists: the revision of a file last synced to the client workspace is not the head revision at the time of the submit.

For example, Alice does a [p4 sync](#) followed by a [p4 edit](#) of file `file.c`, and Bob does the same thing. Alice [p4 submits](#) `file.c`, and then Bob tries to submit `file.c`. Bob's submit fails because if his version of `file.c` were to be accepted into the depot, Alice's changes to `file.c` would no longer be visible. Bob must resolve the conflict before he can submit the file.

- When [p4 integrate](#) has been used to schedule the integration of changes from one file (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
<i>yours</i>	The revision of the file in the client workspace	The file to which changes are being propagated (in integration terminology, this is the <i>target</i> file). Changes are made

Term	Meaning when Resolving Conflicts	Meaning when Resolving for Integration
		to the version of this file in the client workspace, and this file is later submitted to the depot.
<i>theirs</i>	The head revision of the file in the depot.	The file revision in the depot from which changes are being propagated (in integration terminology, this is the <i>source</i> file). This file is not changed in the depot or the client workspace.
<i>base</i>	The file revision synced to the client workspace before it was opened for edit.	The previously-integrated revision of <i>theirs</i> . The latest common ancestor of both <i>yours</i> and <i>theirs</i> .
<i>merge</i>	A file version generated by Perforce from <i>yours</i> , <i>theirs</i> , and <i>base</i> . The user can edit this revision during the resolve process if the file is a text file.	Same as the meaning at left.

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 flags.

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
e	edit merged	Edit the preliminary merge file generated by Perforce.	no
ey	edit yours	Edit the revision of the file currently in the workspace.	yes
et	edit theirs	Edit the revision in the depot with which the workspace revision conflicts (usually the head revision). This edit is read-only.	yes
dy	diff yours	Show diffs between <i>yours</i> and <i>base</i> .	no
dt	diff theirs	Show diffs between <i>theirs</i> and <i>base</i> .	no
dm	diff merge	Show diffs between <i>merge</i> and <i>base</i> .	no
d	diff	Show diffs between <i>merge</i> and <i>yours</i> .	yes
m	merge	Invoke the command: P4MERGE <i>base theirs yours merge</i> To use this option, you must set the environment variable P4MERGE to the	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
		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 <i>yours</i> , ignoring changes that may have been made in <i>theirs</i> .	yes
at	accept theirs	Accept <i>theirs</i> into the client workspace as the resolved revision. The revision (<i>yours</i>) that was in the client workspace is overwritten. When resolving simple conflicts, this option is identical to performing 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 <i>merged</i> file into the client workspace as the resolved revision without any modification. The revision (<i>yours</i>) originally in the client workspace is overwritten.	no
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 (<i>yours</i>) originally in the client workspace is overwritten.	no
a	accept	Keep Perforce's recommended result: <ul style="list-style-type: none"> • if <i>theirs</i> is identical to <i>base</i>, accept <i>yours</i>; • if <i>yours</i> is identical to <i>base</i>, accept <i>theirs</i>; • if <i>yours</i> and <i>theirs</i> are different from <i>base</i>, and there are no conflicts between <i>yours</i> and <i>theirs</i>; accept <i>merge</i>; 	no

Dialog Option	Short Meaning	What it Does	Available by Default for Binary Files?
		<ul style="list-style-type: none"> otherwise, there are conflicts between <i>yours</i> and <i>theirs</i>, 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
<i>n</i> yours	<i>n</i> non-conflicting segments of <i>yours</i> are different than <i>base</i> .
<i>n</i> theirs	<i>n</i> non-conflicting segments of <i>theirs</i> are different than <i>base</i> .
<i>n</i> both	<i>n</i> non-conflicting segments appear identically in both <i>theirs</i> and <i>yours</i> , but are different from <i>base</i> .
<i>n</i> conflicting	<i>n</i> segments of <i>theirs</i> and <i>yours</i> are different from <i>base</i> and different from each other.

If there are no conflicting chunks, it is often safe to accept Perforce's generated merge file, 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 *P4 User's Guide*. To constrain the process to one type of resolve, use the **-A** flag. :

Flag	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** flag following the **-A** flag. 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

- aflags** Skip the resolution dialog, and resolve the files automatically as follows:
- am**
 - af**
 - **-am**: Automatic Mode. Automatically accept the Perforce-recommended file revision: if *theirs* is identical to *base*, accept *yours*; if *yours* is identical to *base*, accept *theirs*; if *yours* and *theirs* are different from *base*, and there are no conflicts between *yours* and *theirs*; accept *merge*; otherwise, there are conflicts between *yours* and *theirs*, so skip this file.
 - **-ay**: Accept *Yours*, ignore *theirs*.
 - **-at**: Accept *Theirs*. Use this flag with caution, as the file in the client workspace will be overwritten!
 - **-as**: Safe Accept. If either *yours* or *theirs* is different from *base*, (and the changes are in common) accept that revision. If both are different from *base*, skip this file.

- **-af**: Force Accept. Accept the *merge* file no matter what. If the *merge* file has conflict markers, they will be left in, and you'll need to remove them by editing the file.

-Aflags	Action (non-content) resolves: Constrain the type of resolve to branching, deletion, file type change, or move/rename.
-Aa	
-Ab	<ul style="list-style-type: none"> • -Ab: Resolve attributes set by p4 attribute
-Ac	
-Ad	<ul style="list-style-type: none"> • -Ab: Resolve file branching; that is, integrations where the source is edited and the target is deleted
-At	
-Am	<ul style="list-style-type: none"> • -Ac: Resolve file content changes • -Ad: Integrations where the source is deleted and target is deleted • -At: Filetype changes • -Am: Move and renames <p>For details, see the <i>P4 User's Guide</i> and "Non-Content-Related Resolves" on page 302.</p>
-dflag	<p>When merging files, ignore specified differences in whitespace or line-ending convention. (If you use these flags, and the files differ by whitespace only, p4 resolve uses the text in the workspace file.)</p> <ul style="list-style-type: none"> • -db: Ignore whitespace-only changes (for instance, a tab replaced by eight spaces) • -dw: Ignore whitespace altogether (for instance, deletion of tabs or other whitespace) • -dl: Ignore differences in line-ending convention
-f	Allow already resolved, but not yet submitted, files to be resolved again.
-n	List the files that need resolving without actually performing the resolve.
-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 the [Global Options](#) section.

Usage Notes

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

- [p4 resolve](#) works only with files that have been scheduled for resolve. Three operations schedule files for resolution:

- Integrating the file with [p4 integrate](#) 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 than the current head revision; the submit fails, and the file is scheduled for resolve.
- Running [p4 sync](#) instead of running [p4 submit](#) on the open file. Nothing is copied into the client workspace; instead, the file is scheduled for resolve. (The only benefit of scheduling files for resolve with [p4 sync](#) instead of a failed submit is that the submit will not fail).

When [p4 resolve](#) is run with no file arguments, it operates on all files in the client workspace that have been scheduled for resolve.

- 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

Synopsis

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

Syntax

```
p4 [g-opts] resolved [-o] [file...]
```

Description

[p4 resolved](#) lists files that have been resolved, but have not yet been submitted. The files are displayed one per line in the following format:

localFilePath - *action* from *depotFilePath*#*revisionRange*

where *localFilePath* is the full path name of the resolved file on the local host, *depotFilePath* is the path of the depot file relative to the top of the depot, *revisionRange* is the revision range that was integrated, and *action* is one of `merge`, `branch`, or `delete`.

If file pattern arguments are provided, only resolved, unsubmitted files that match the file patterns are included.

Although the name [p4 resolved](#) seems to imply that only files that have gone through the [p4 resolve](#) process are listed, this is not the case. A file is also considered to be resolved if it has been opened by [p4 integrate](#) for `branch`, opened by [p4 integrate](#) for `delete`, or has been resolved with [p4 resolve](#).

Options

-o Output the base file name and revision that was used during the resolve.

g-opts See the [Global Options](#) section.

Usage Notes

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

Related Commands

To see a list of integrations that have been submitted [p4 integrated](#)

To view a list of integrations that have not yet been resolved [p4 resolve](#) -n

To schedule the propagation of changes from one file to another [p4 integrate](#)

To resolve file conflicts, or to propagate changes as scheduled by
[p4 integrate](#) [p4 resolve](#)

Synopsis

Restore old archived revisions from an archive depot.

Syntax

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

Description

The [p4 restore](#) command transfers archives from a named *depot* of type **archive** back to their original locations in a local depot. After being restored, the revisions' action is restored to whatever it was before it was archived.

Options

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

Usage Notes

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

- Storage for the archive depot must be mounted unless you are using the **-n** flag.

Related Commands

To create a depot	p4 depot
To archive files into an archive depot	p4 archive
To obliterate files without archiving them	p4 obliterate

Synopsis

Discard changes made to open files.

Syntax

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

Description

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

When you revert files you opened with [p4 delete](#), the files are reinstated in the client workspace. When you revert files that have been opened by [p4 add](#), Perforce leaves the client workspace files intact. When you revert files you've opened with [p4 integrate](#), Perforce removes the files from the client workspace. When you revert files you've opened with [p4 move](#), only the file open for `move/add` can be reverted.

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: <ul style="list-style-type: none"> • open for edit but have unchanged content and unchanged filetype; or • open for integrate via p4 integrate and have not yet been resolved with p4 resolve.
-n	List the files that would be reverted without actually performing the revert. This lets you make sure the revert does what you think it does before actually reverting the files.
-k	Keep workspace files; the file(s) are removed from any changelists and Perforce records that the files as being no longer open, but the file(s) are unchanged in the client workspace.
-w	Files that are open for <code>add</code> are to be deleted (wiped) from the workspace when reverted.
-c <i>changelist</i>	Reverts only those files in the specified changelist.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

No	No	list
----	----	------

- [p4 revert](#) differs from most Perforce commands in that it usually *requires* a file argument. The files that are reverted are those that lie in the intersection of the command line file arguments and the client view.

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

- Reverting a file that has been opened for **edit** will overwrite any changes you have made to the file since the file was opened. It may be prudent to use **p4 revert -n** to preview the results before running [p4 revert](#).

Examples

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

Related Commands

To open a file for add	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

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://public.perforce.com/wiki/P4Review>

and is documented in the *Perforce System Administrator's Guide*.

Options

-c changelist	List all submitted changelists above and including <i>changelist</i> .
-t countername	List all submitted changelists above the value of the Perforce counter <i>countername</i> .
-c changelist -t countername	Set the value of counter <i>countername</i> to <i>changelist</i> . This command has been replaced by p4 counter , but has been maintained for backwards compatibility.
g-opts	See the Global Options section.

Usage Notes

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

- The commands [p4 review](#), [p4 reviews](#), and [p4 counter](#) are all intended for use by external programs that call Perforce.
- The warnings applicable to [p4 counter](#) apply here as well.

Related Commands

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

Synopsis

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

Syntax

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

Description

The [p4 reviews](#) command is intended for use in external programs that call Perforce.

Users subscribe to review files by providing file patterns in the **Reviews:** field in their [p4 user](#) form.

`p4 reviews -c changelist` lists each user who has subscribed to review any files included in the submitted changelist *changelist*. The alternate form, (`p4 reviews file...`), lists the users who have subscribed to review any files that match the file patterns provided as arguments. If you provide no arguments to [p4 reviews](#), all users who have subscribed to review any files are listed.

Options

-C clientworkspace List all users who have subscribed to review any files opened in the specified workspace *clientworkspace*.

-c changelist List all users who have subscribed to review any files included in submitted changelist *changelist*.

g-opts See the [Global Options](#) section.

Usage Notes

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

- The syntax `p4 reviews -c changelist file...` ignores the file arguments entirely.
- [p4 reviews](#) is an unusual command. It was created to support external daemons, but it does nothing without the **Reviews:** field of the [p4 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

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 serverID
p4 [g-opts] server -i
```

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 **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 *Distributing Perforce*.

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.
Type:	Writable	Server executable type. One of: <ul style="list-style-type: none"> • server • proxy • broker
Services:	Writable	Services provided by this server. One of: <ul style="list-style-type: none"> • standard (a standard Perforce server) • replica (a read-only replica server) • broker (a p4broker process) • proxy (a p4p caching proxy)

Field Name	Type	Description
		<ul style="list-style-type: none"> • commit-server (a central server in a distributed installation) • edge-server (a node in a distributed installation) • forwarding-replica (a replica that has been configured to forward commands that involve database writes to a master server) • build-server (a replica that supports build automation) • P4AUTH (a server that provides authentication) • P4CHANGE (a server that provides change numbering)
Name:	Writable	The P4NAME associated with this server.
Address:	Writable	The P4PORT used by this server.
Description:	Writable	An optional description for this server.
ClientDataFilter:	Writable	<p>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.</p> <p>To include client data, use the syntax:</p> <pre>//client-pattern/...</pre> <p>To exclude client data, use the syntax:</p> <pre>-//client-pattern/...</pre> <p>All patterns are specified in client syntax.</p>
RevisionDataFilter:	Writable	<p>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.</p> <p>To include depot data, use the syntax:</p> <pre>//depot/pattern/...</pre> <p>To exclude depot data, use the syntax:</p> <pre>-//depot/pattern/...</pre> <p>All patterns are specified in depot syntax.</p>

Field Name	Type	Description
ArchiveDataFilter:	Writable	<p>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 will not be transferred until they are referenced by a replica command which needs the file content.</p> <p>Files specified in ArchiveDataFilter: field are transferred to the replica regardless of whether any users of the replica have made requests for their content.</p> <p>To automatically transfer files on submit, use the syntax:</p> <pre>//depot/pattern/...</pre> <p>To exclude files from automatic transfer, use the syntax:</p> <pre>-//depot/pattern/...</pre> <p>All patterns are specified in depot syntax.</p>

Options

-g	Generate a new serverID as part of the form.
-d serverID	Delete the named server specification.
-o	Write the named server specification to standard output.
-i	Read a server specification from standard input.
g-opts	See the Global Options section.

Usage Notes

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

Related Commands

To change a server's ID after creation

[p4 serverid](#)

To list all known servers

[p4_servers](#)

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.

Options

serverID	If supplied, update <code>server.id</code> with the unique ID of the server.
-----------------	--

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

Usage Notes

Can File Arguments Use Revision Specifier?	Can File Arguments Use Revision Range?	Minimal Access Level Required
N/A	N/A	list, or super to set the server ID

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.

Related Commands

To edit or view a server specification
--

p4 server

To list all known servers

[p4_servers](#)

Synopsis

Display list of all server specifications.

Syntax

p4 [*g-opts*] servers

Description

[p4 servers](#) lists all server specifications stored at a master Perforce server.

An operator type user may not execute this command.

Options

g-opts See the [Global Options](#) section.

Usage Notes

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

- The output of [p4 servers](#) may be easier to parse if you retrieve it in tagged form:

p4 -ztag servers

Related Commands

To edit or view a server specification	p4 server
To set a server's unique ID	p4 serverid

Synopsis

Set Perforce variables in the Windows registry or OS X system settings.

Syntax

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

Description

Both Perforce applications and the shared versioning service make use of certain system variables.

On Windows (or OS X), you can set the values of these variables in the registry (or user or system preferences) with [p4 set](#); on other operating systems, Perforce uses environment variables for the same purpose.

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.

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

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

To view a list of the values of all Perforce variables, use [p4 set](#) without any arguments. 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 or OS X, this displays either the environment variable (if set), or the value in the registry (or system settings) and whether it was defined with [p4 set](#) (for the current user) or [p4 set -s](#) (for the local machine).

[p4 set](#) can be used on Linux and UNIX to view the values of variables, but if you try to use [p4 set](#) to set variables on these operating systems, Perforce displays an error message.

Options

-s Set the value of the registry variable (or system settings) for the local machine.

On Windows, without this flag, [p4 set](#) sets the variables in the HKEY_CURRENT_USER hive; when you use the **-s** flag (and have Windows administrative privileges), the variables are set in the HKEY_LOCAL_MACHINE hive.

On OS X, without this flag, [p4 set](#) sets the variables in your `com.perforce.environment` property list in the `~/Library/Preferences` folder; when you use the **-s** flag (and have administrative privileges), the variables are set in the system `/Library/Preferences` folder.

These locations are reflected in the output of [p4 set](#) on Windows and OS X.

-S *svcname* Set the value of the registry variables as used by service *svcname*. You must have Windows administrator privileges to do this.

The **-S** flag is ignored on OS X.

g-opts See the [Global Options](#) section.

Usage Notes

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

- You'll find a listing and discussion of the Perforce variables in the [Environment Variables](#) section of this manual.
- Changes to registry 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 or OS X, 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 option](#) flags, 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 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 or OS X, 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.

Synopsis

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

Syntax

```
p4 [g-opts] shelve [-p] [files]
p4 [g-opts] shelve [-a option] [-p] -i [-f | -r ]
p4 [g-opts] shelve [-a option] [-p] -r -c changelist
p4 [g-opts] shelve [-a option] [-p] -c changelist [-f] [file ...]
p4 [g-opts] shelve -d -c changelist [-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.

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` flags are also used to change the shelved file content. For more information about the use of shelves in distributed environments, see *Usage Notes* below and also the *Distributed Perforce* manual.

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 shelf, the file must first be opened in the shelf's changelist; use [p4 reopen](#) to move an opened file from one changelist to another.

Options

-f	Force the overwriting of any existing shelved files in a pending changelist with the contents of their client workspace copies. Perforce administrators can use this flag with -d to force the discarding of shelved files in a specified changelist.
-i	Read a changelist description from standard input. Input must be in the same format used by the p4 shelve form. (When modifying an existing changelist with shelved files, this option also implies -c).
-a option	The <i>submitunchanged</i> (default) option shelves all files. The <i>leaveunchanged</i> option shelves only the changed files; it leaves the unchanged files opened at the numbered pending changelist.
-c changeli	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. Only the user and workspace that owns the pending changelist can discard its shelved files. (Administrators can use -f to discard files.)
-r	Replace all shelved files in the changelist with the files that are opened in your workspace. (To replace the files in the changelist with files open at a specific change number, use the -c or -i options.) Only the user and workspace associated with a pending changelist can replace its shelved files.
-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 <i>Usage Notes</i> for more information.
g-opts	See the Global Options section.

Usage Notes

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

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** flag 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:

```
shelve -p -f -c myChange
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:

- If you're not on the server owning the shelf, you can't unshelve a remote promoted shelf into already-open local files.
- If you're not on the server owning the shelf, you can't unshelve into a different branch (**-b -S**)
- 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](#)

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 1024 for **-h** or 1000 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 75883921 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 disk space 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	Show size using the most economical units: bytes, kilobytes, megabytes, gigabytes, or terabytes. If you omit this option, sizes are shown in bytes.
-b <i>blocksize</i>	Display results in blocks of <i>blocksize</i> bytes. Each accumulated file size is rounded up to the nearest <i>blocksize</i> bytes.
-m <i>max</i>	Limit output to <i>max</i> lines of output.
-h or -H	Display size in human-readable form, using a scaling factor of 1024 for -h or 1000 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.
-z	When calculating size information, exclude lazy copies.
-A	Display files in archive depots. See p4_archive for details.
-U <i>unloadfile</i>	List only file sizes in the unload depot. See p4_unload for details.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- The [p4 sizes](#) command is functionally similar to the UNIX `du` command.
- If no revision range is specified, the implicit revision range of `#1` through `#head` is assumed.
- File sizes are based on the normalized (UNIX linefeed convention) and uncompressed version of the depot file, regardless of how the file is represented when synced to a client workspace.

Examples

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

Synopsis

Open files for add, delete, and/or edit in order to reconcile a workspace with changes made outside of Perforce.

Syntax

```
p4 [g-opts] status [ -e -a -d -f -I ]
p4 [g-opts] status -A [ -e -a -d -f -I ]
```

Description

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.)

To limit the scope of [p4 status](#) to add, edit, or delete, use the **-a**, **-e**, or **-d** flags.

By default, [p4 status](#) does not check files and/or paths mentioned in the [P4IGNORE](#) file. Use the **-I** flag to override this behavior and ignore the [P4IGNORE](#) file.

Options

-A	Add, edit, <i>and</i> 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 .
-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 .
g-opts	See the Global Options section.

Usage Notes

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

No	No	open
----	----	------

- The [p4 status](#) command produces output in local syntax. To see file names and paths in depot syntax, use [p4 reconcile](#).
- When called without arguments, [p4 status](#) only previews the results of the workspace reconciliation. You must use either `p4 status -A` (or some combination of the `-e`, `-a`, or `-d` flags) to actually open the files in a changelist.

Related Commands

To reconcile a workspace that has been modified outside Perforce

[p4 reconcile](#)

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]
```

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.

For a detailed discussion of streams, refer to the *P4 User's 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 <i>//depotname/streamname</i> .
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 <i>streamname</i> portion of the stream path.

Field Name	Type	Description
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 <code>//depotname/streamname</code> .
Type:	Writable, mandatory	<p>The stream's type determines the expected flow of change. Valid stream types are mainline, virtual, development, and release.</p> <ul style="list-style-type: none"> • mainline <p>The mainline stream is the parent of all streams in the stream depot. Every stream depot must have at least one mainline stream.</p> • virtual <p>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.</p> • release <p>More stable than the mainline. Release streams copy from the parent and merge to the parent.</p> • development <p>Less stable than the mainline. Development streams expect to merge from parent streams and copy to the parent.</p> • task <p>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.</p> <p>The default is stream type is development.</p>
Description:	Writable, optional	Description of the stream.
Options:	Writable	<p>Settings that configure stream behavior as follows:</p> <ul style="list-style-type: none"> • [un]locked

Field Name	Type	Description
		<p>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.</p> <ul style="list-style-type: none"> • [all owner]submit <p>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.</p> <ul style="list-style-type: none"> • [no]toparent <p>Specifies whether integrations from the stream to its parent are expected. The default is toparent.</p> <ul style="list-style-type: none"> • [no]fromparent <p>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.</p> <p>A virtual stream must have its flow options set to notoparent and nofromparent.</p> <p>Flow options are ignored for mainline streams</p>
Paths:	Writable	<p>Paths define how files are incorporated into the stream structure. Specify paths using the following format:</p> <p><i>path_type view_path [depot_path]</i></p> <p>where <i>path_type</i> is a single keyword, <i>view_path</i> is a file path with no leading slashes, and the optional <i>depot_path</i> is a file path beginning with <i>//</i>.</p> <p>The default path is share ...</p> <p>Valid path types are:</p> <ul style="list-style-type: none"> • share view_path <p>Specified files can be synced, submitted, and integrated to and from the parent stream.</p> <ul style="list-style-type: none"> • isolate view_path <p>Specified files can be synced and submitted, but cannot be integrated to and from the parent stream.</p>

Field Name	Type	Description
		<ul style="list-style-type: none"> • import <i>view_path</i> [<i>depot_path</i>] <p>Specified files can be synced, but cannot be submitted or integrated to and from the parent stream. The <i>view_path</i> is mapped as in the parent stream's view, or to an (optional) <i>depot_path</i>.</p> <p>The <i>depot_path</i> may include a changelist specifier; clients of that stream 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: <i>//depot/import/...@1000</i>. Revisions from changelists greater than 1000 will be automatically hidden from most commands.</p> <p>The changelist limits in effect for a given client are displayed in a read-only client spec field called Change View.</p> <ul style="list-style-type: none"> • exclude <i>view_path</i> <p>Specified files cannot be synced, submitted or integrated to and from the parent stream.</p> <p>By default, streams inherit their structure from the parent stream (except mainlines, which have no parent).</p> <p>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 <i>shared</i> path to an <i>isolated</i> path, but if the parent stream defines a path as <i>isolated</i>, its child cannot restore full access by specifying the path as <i>shared</i>.)</p> <p>Note that the <i>depot_path</i> is relevant only when the <i>path_type</i> is <i>import</i>.</p>
Remapped:	Writable, optional	<p>Reassigns the location of workspace files. To specify the source path and its location in the workspace, use the following syntax:</p> <p><i>view_path_1 view_path_2</i></p> <p>where <i>view_path_1</i> and <i>view_path_2</i> 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:</p> <p>... projectX/...</p> <p>Line ordering in the Remapped: field is significant: if more than one line remaps the same files, the later line takes</p>

Field Name	Type	Description
		precedence. Remappings are inherited by child streams and the workspaces associated with them.
Ignored:	Writable, optional	<p>A list of file or directory names to be ignored in client views. For example:</p> <pre> /tmp # ignores files named "tmp" /tmp/... # ignores directories named "tmp" .tmp # ignores file names ending in .tmp </pre> <p>Lines in the Ignored: field can appear in any order. Ignored files and directories are inherited by child stream client views.</p>

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 flag 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.
-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 .
g-opts	See the Global Options section.

Usage Notes

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

- 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

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

<i>-F filter</i>	Filter the output according to the contents of specified fields.
<i>-T fields</i>	Limit field output to fields specified in a list of <i>fields</i> . Field names may be separated by a space or comma. Intended for scripting. This option forces tagged output.
<i>-U</i>	Display task streams unloaded with p4 unload .
<i>-m max</i>	Maximum number of streams to list.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

Examples

List the streams in the jam depot	<code>p4 streams //jam/...</code>
List the release streams owned by Bruno	<code>p4 streams -F "Owner=bruno Type=release"</code>

Related Commands

Create, edit or delete a stream

[p4 stream](#)

Synopsis

Send changes made to open files to the depot.

Syntax

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

Description

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

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

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

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

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

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

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.

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

Form Fields

Field Name	Type	Description
Change:	Read-only	The change number, or new if submitting the default changelist.
Client:	Read-only	Name of current client workspace.
User:	Read-only	Name of current Perforce user.
Status:	Read-only, value	One of pending , submitted , or new . Not editable by the user. The status is new when the changelist is created; pending when it has been created but has not yet been submitted to the depot with p4 submit , and submitted when its contents have been stored in the depot with p4 submit .
Description:	Writable	Textual description of changelist. This value <i>must</i> be changed.
Jobs:	List	A list of jobs that are fixed by this changelist. This field does not appear if there are no relevant jobs. Any job that meets the jobview criteria as specified on the p4 user form are listed here by default, but can be deleted from this list.
Type:	Writable, value	Type of change: restricted or public . A restricted shelved or committed changelist denies access to users who do not own the changelist and who do not have list permission to at least one file in the changelist. A restricted pending (unshelved) changelist denies access to non-owners of the changelist. Public changes are displayed without these restrictions.
Files:	List	A list of files being submitted in this changelist. Files can be deleted from this list, but cannot be changed or added.

Options

-c *changelist* Submit changelist number *changelist*.

Changelists are assigned numbers either manually by the user with [p4 change](#), or automatically by Perforce when submission of the default changelist fails.

-e <i>shelvedchange</i>	<p>Submit shelved changelist number <i>shelvedchange</i>.</p> <p>The -e flag submits a shelved changelist without transferring file 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.</p> <p>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 are no longer available to be unshelved or submitted.</p>
-d <i>description</i>	<p>Immediately submit the default changelist with the <i>description</i> supplied on the command line, and bypass the interactive form. This option is useful when scripting, but does not allow for jobs to be added, nor for the default changelist to be modified.</p>
-f <i>submitoption</i>	<p>Override the SubmitOptions: setting in the p4 client form. Valid <i>submitoption</i> values are:</p> <ul style="list-style-type: none">• 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 <i>not</i> reopened in the default changelist.• leaveunchanged Only those files with content or type changes are submitted to the depot. Any unchanged files are moved to the default changelist.

	<ul style="list-style-type: none"> • leaveunchanged+reopen <p>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.</p>
-i	Read a changelist specification from standard input. Input must be in the same format as that used by the p4 submit form.
-r	Reopen files for edit in the default changelist after submission. Files opened for add or edit in will remain open after the submit has completed.
-s	<p>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.</p> <p>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.)</p> <p>This option works in conjunction with the -s option to p4 fix, and is intended for use in conjunction with defect tracking systems.</p>
g-opts	See the Global Options section.

Usage Notes

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

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

The file pattern parameter to [p4 submit](#) can only be used when submitting the default changelist.

- A progress indicator is available for [p4 submit](#) if you request it with **p4 -I submit**.

Examples

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

Related Commands

To create a new, numbered changelist	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

Synopsis

Update the client workspace to reflect the contents of the depot.

Syntax

```
p4 [g-opts] sync [-f][-L][-k][-n][-N][-q][-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]...]
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 newly synced files are not available for editing until opened with [p4 edit](#) or [p4 delete](#). Newly synced files are read-only; [p4 edit](#) and [p4 delete](#) make the files writable. Do not use your operating system's commands to make the files writable; instead, use Perforce to do this for you.

Options

- | | |
|-----------|---|
| -f | Force the sync. Perforce performs the sync even if the client workspace already has the file at the specified revision. If the file is writable, it is overwritten. |
| | This flag does not affect open files, but it <i>does</i> override the noclobber client option. |

-k	<p>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.</p> <p>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.</p> <p>If your administrator has set the zerosyncPrefix configurable, <i>all</i> workspaces with names that begin with the specified prefix assume sync -k.</p>
-L	<p>For scripting purposes, perform the sync on a list of valid file arguments in full depot syntax with a valid revision number.</p>
-m max	<p>Sync only the first <i>max</i> files specified.</p>
-n	<p>Display the results of the sync without actually performing the sync.</p> <p>This lets you make sure that the sync does what you think it does before you do it.</p>
-N	<p>Display a summary of the expected network traffic associated with a sync, without performing the sync.</p> <p>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 disk-space-limited.</p> <p>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've inadvertently included a folder tree that holds several gigabytes of assets <i>before</i> attempting to sync your newly-configured workspace.</p>
-p	<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.</p> <p>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.</p>
-q	<p>Quiet operation: suppress normal output messages. Messages describing errors or exceptional conditions are not suppressed.</p>
-s	<p>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.</p> <p>If your client workspace specification has both the allwrite and noclobber options set, this check is performed by default.</p>
--parallel	<p>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.</p> <ul style="list-style-type: none">• threads=N sends files concurrently using <i>N</i> independent network connections. The specified threads grab work in batches.

- **batch=N** specifies the number of files in a batch.
- **batchsize=N** specifies the number of bytes 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.
- **minsize=N** specifies the minimum number of bytes in a parallel sync. A sync that is too small will not initiate parallel file transfers.

See "Parallel Processing" below for more information.

g-opts See the [Global Options](#) section.

Usage Notes

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

- If the client 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**.

Scripting

The **-m max** flag is useful when combined with the **-n** flag for efficient scripting. For example, a command like **p4 sync -n -m 1** does not sync any files, but displays only one line of output if there are any files to be synced, or a message indicating that the workspace is up to date. Without the **-m 1** option, the output could conceivably be thousands of lines long, all of which would be discarded.

The **-L** flag 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 flag 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.

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

<code>p4 sync</code>	<p>Copy the latest revision of all files from the depot to the client workspace, as mapped through the client view.</p> <p>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.</p>
<code>p4 sync file.c#4</code>	Copy the fourth revision of <code>file.c</code> to the client workspace, with the same exceptions as in the example above.
<code>p4 sync //depot/proj1/...@21</code>	<p>Copy all the files under the <code>//depot/proj1</code> directory from the depot to the client workspace, as mapped through the client view.</p> <p>Don't copy the latest revision; use the revision of the file in the depot after changelist 21 was submitted.</p>
<code>p4 sync @labelname</code>	<p>If <i>labelname</i> 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 <i>labelname</i>.</p> <p>Files listed in <i>labelname</i>, but not in the workspace view, are not copied into the workspace.</p> <p>Files <i>not</i> listed in <i>labelname</i> are deleted from the workspace. (That is, <i>@labelname</i> is assumed to apply to all revisions up to, and including, the revisions specified in <i>labelname</i>. This includes the nonexistent revision of the unlisted files.)</p>
<code>p4 sync @labelname,@labelname</code>	<p>Bring the workspace into sync with a label as with <code>p4 sync @labelname</code>, but preserve unlabeled files in the workspace.</p> <p>(The revision range <i>@labelname,@labelname</i> applies only to the revisions specified in the label name itself, and excludes the nonexistent revision of the unlisted files.)</p>
<code>p4 sync @2011/06/24</code>	Bring the workspace into sync with the depot as of midnight, June 24, 2011. (That is, include all changes made during June 23.)
<code>p4 sync status%40june1st.txt</code>	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 <code>status@june1st.txt</code> .

For details, see [“Limitations on characters in filenames and entities” on page 482](#).

<code>p4 sync file.c#none</code>	Sync to the nonexistent revision of <code>file.c</code> ; the file is deleted from the workspace.
<code>p4 sync ...#none</code>	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

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 flag 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.
-n	Display what p4 tag would do without actually performing the operation.
-l <i>labelname</i>	Specify the label to be applied to file revisions
-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 the Global Options section.

Usage Notes

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

- By default, [p4 tag](#) operates on the head revision of files in the depot. To preserve the state of a client workspace, use [p4 labelsync](#), which operates on the revision of files last synced to your workspace.

- With a distributed Perforce service, [p4 labelsync](#) works with a label local to the edge server. The **-g** flag 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** flag 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** flag is inverted to allow updating of local labels.

Examples

<code>p4 tag -l rel1 //depot/1.0/...</code>	Tag the head revisions of files in <code>//depot/1.0/...</code> with label <code>rel1</code> . If the label <code>rel1</code> does not exist, create it.
<code>p4 tag -l build //depot/1.0/...@1234</code>	Tag the most recent revisions as of the submission of changelist <code>1234</code> of files in <code>//depot/1.0/...</code> with label <code>build</code> . If the label <code>build</code> does not exist, create it.
<code>p4 files @labelname</code>	List the file revisions tagged by <i>labelname</i> .

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

Synopsis

Display all tickets granted to a user by [p4 login](#).

Syntax

`p4 [g-opts] tickets`

Description

The [p4 tickets](#) command lists all tickets stored in the user's ticket file.

Options

g-opts See the [Global Options](#) section.

Usage Notes

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

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

Examples

`p4 tickets` Display all tickets stored in a user's local ticket file.

Related Commands

To start a login session (to obtain a ticket) [p4 login](#)

To end a login session (to delete a ticket) [p4 logout](#)

Synopsis

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

Syntax

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

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 flags specified, the command invokes the default editor to allow the user to specify one or more trigger definitions.
- The **-i** flag specifies that the user use standard input to specify one or more trigger definitions.
- The **-o** flag 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 the chapter "Triggers and Daemons" in the *Perforce Administrator's Guide*.

Options

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

Usage Notes

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

Examples

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 the *System Administrator's Guide*.

Related Commands

To obtain information about the changelist being submitted

[p4 describe](#)
[p4 opened](#)

To aid daemon creation

[p4 review](#)
[p4 reviews](#)
[p4 counter](#)
[p4 counters](#)
[p4 user](#)

Synopsis

Establish trust of an SSL connection to a Perforce service.

Syntax

```
p4 [-p port] 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.

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

<code>-l</code>	List all known fingerprints on this client workstation.
<code>-y</code>	Automatically accept any prompts.
<code>-n</code>	Automatically refuse any prompts.
<code>-d</code>	Delete an existing trusted fingerprint.
<code>-f</code>	Force the replacement of a mismatched fingerprint.
<code>-i <i>fingerprint</i></code>	Install the specified <i>fingerprint</i> .
<code>-r</code>	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 flag may be combined with the <code>-l</code> , <code>-i</code> , or <code>-d</code> flags.
<code>-p</code>	The P4PORT setting of the service from which to check or retrieve the fingerprint.

Usage Notes

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

- Your system administrator can help you confirm the accuracy of any fingerprint (or change to a fingerprint) provided to you by a Perforce service.

Synopsis

Modify the file name-to-type mapping table.

Syntax

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

Description

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

By default, Perforce automatically determines if a file is of type **text** or **binary** based on an analysis of the first 65536 bytes of a file. If the high bit is clear in each of the first 65536 bytes, Perforce assumes it to be **text**; otherwise, it's **binary**. Files compressed in the **.zip** format (including **.jar** files) are also automatically detected and assigned the type **ubinary**.

Although this default behavior can be overridden by the use of the **-t filetype** flag, it's easy to overlook this, particularly in cases where files' types were usually (but not always) detected correctly. This situation occasionally appears with PDF files (which sometimes begin with over 65536 bytes of ASCII comments) and RTF files, which usually contain embedded formatting codes.

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

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

Form Fields

The [p4 typemap](#) form contains a single **TypeMap:** field, consisting of pairs of values linking file types to file patterns specified in depot syntax:

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

Usage Notes

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

- To specify all files with a given extension at or below a desired subdirectory, use four periods after the directory name, followed by the extension. (for instance, `//path/....ext`) The first three periods specify "all files below this level". The fourth period and accompanying file extension are parsed as "ending in these characters".
- File type modifiers can be used in the typemap table. Useful applications include forcing keyword expansion on or off across directory trees, enforcing the preservation of original file modification times (the **+m** file type modifier) in directories of third-party DLLs, or implementing pessimistic locking policies.
- Specify multiple file type modifiers consecutively. For example, **binary+1F510** refers to a **binary** file with exclusive-open (1), stored in full (F) rather than compressed, and for which only the most recent ten revisions are stored (510). For more information on syntax, see the [File Types](#) section.
- If you use the **-t** flag and file type modifiers to specify a file type on the command line, and the file to which you are referring falls under a [p4 typemap](#) mapping, the file type specified on the command line overrides the file type specified by the typemap table.

Examples

To tell the Perforce 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 the [File Types](#) section.

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

```
Typemap:
  +l //depot/...
```

Related Commands

To add a new file with a specific type, overriding the typemap table

p4 add -t *type* *file*

To change the filetype of an opened file, overriding any settings in the typemap table

[p4 reopen](#) -t *type* *file*

Synopsis

Unloads a workspace, label, or task stream to the unload depot or to a flat file.

Syntax

```
p4 [g-opts] unload [-f -L -z] [-c workspace | -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** flag 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 postprocessing 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** flags to unload a specific client workspace or label. By default, users can only unload their own workspaces or labels; administrators can use the **-f** flag to unload workspaces and labels owned by other users.

Use the **-a**, **-al**, or **-ac** flags 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** flag.

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** flag 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

-c workspace	Unload the specified client workspace's metadata from db.have (and related tables) and store it in the unload depot.
-l label	Unload the specified label from db.label (and related tables) and store it in the unload depot.
-s stream	Unload the specified task stream. (the <i>stream</i> must be of type task)
-d date	Unload metadata older than the specified date.
-u user	Unload metadata owned by the specified user.
-o outputFile	<p>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.</p> <p>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>
-f	Force flag; administrators can unload workspaces, labels, and task streams owned by other users.
-L	Unload a locked workspace, label, or task stream.
-z	Store the unloaded workspace, label, or task stream in compressed format.
-a	Unload all applicable client workspaces and labels; requires -d , -u , or both -d and -u flags. This flag does not affect task streams.
-ac	Unload client workspaces; requires -d , -u , or both -d and -u flags.
-al	Unload labels; requires -d , -u , or both -d and -u flags.
g-opts	See the Global Options section.

Usage Notes

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

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

		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](#)

Synopsis

Release the lock on a file.

Syntax

```
p4 [g-opts] unlock [-c changelist | -s shelvedchange | -x ] [-f] file...
```

Description

The [p4 unlock](#) command releases locks created by [p4 lock](#).

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

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

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

Options

-c <i>changelist</i>	Unlock files in pending changelist <i>changelist</i> . This flag applies to opened files in a pending changelist that were locked by p4 lock or a failed submit operation of an unshelved changelist.
-s <i>shelvedchange</i>	If a file is locked in a pending shelved changelist, unlock it and keep it within the <i>shelvedchange</i> . This can typically only happen if a p4 submit -e command is aborted.
-f	Superuser or administrator force flag; allows unlocking of files opened by other users.
-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 the Global Options section.

Usage Notes

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

Related Commands

To lock files so other users can't submit them	p4 lock
To display all your open, locked files (UNIX)	p4 opened <code>grep "*locked"</code>

Synopsis

Restore shelved files from a pending change into a workspace

Syntax

```
p4 [g-opts] unshelve -s changelist [-f] [-n] [-c changelist] [-b branch | -S 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.

You can limit the files to be unshelved by specifying a file pattern.

Unshelving copies the shelved files into the user's workspace as they existed when they were shelved. (For example, a file open for edit when shelved will also be open for edit in the unshelving user's workspace.)

Options

-s <i>changelist</i>	Specify the pending changelist number that contains the originally-shelved files.
-c <i>changelist</i>	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.
-b <i>branch</i>	Specifies a branch spec through which the shelved files will be mapped prior to unshelving. This flag enables you to shelve files in one branch and unshelve them in another.
-S <i>stream</i>	Specifies the use of a stream-derived branch view to map the shelved files between the specified stream and its parent stream; you can shelve files in one stream and unshelve them in another related stream.
-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.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- 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](#)

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

<code>-L</code>	For scripting purposes, perform the update on a list of valid file arguments in full depot syntax with a valid revision number.
<code>-n</code>	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.
<code>-q</code>	Quiet operation: suppress normal output messages. Messages regarding errors or exceptional conditions are not suppressed.
<code>g-opts</code>	See the Global Options section.

Usage Notes

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

- [p4 update](#) is an alias for `p4 sync -s`.

Related Commands

p4 update is an alias for <code>p4 sync -s</code>	p4 sync -s
To copy files from the depot to the client workspace	p4 sync

Synopsis

Create or edit Perforce user specifications and preferences.

Syntax

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

Description

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 multiserver environments, and are restricted to an even smaller subset of Perforce commands. Neither **operators** nor **service** users consume Perforce licenses.

When called without a *username*, [p4 user](#) edits specification of the current user. When called with a *username*, the user specification is displayed, but cannot be changed. The form appears in the editor defined by the [P4EDITOR](#) environment variable.

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

The user who gives a Perforce command is not necessarily the user under whose name the command runs. The user for any particular command is determined by the following:

- If the user running the command is a Perforce superuser, and uses the syntax **p4 user -f username**, user *username* is edited.
- If the *-u username* flag is used on the command line (for instance, **p4 -u joe submit**), the command runs as that user (a password may be required);
- If the above hasn't been done, but the file pointed to by the [P4CONFIG](#) environment 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:	Writable	Type of user: standard , operator , or service .
Email:	Writable	The user's email address. By default, this is <i>user@client</i> .
Update:	Read-only	The date and time this specification was last updated.
Access:	Read-only	The date and time this user last ran a Perforce command.
FullName:	Writable	The user's full name.
JobView:	Writable	A description of the jobs to appear automatically on all new changelists (described in the <i>Usage Notes</i> below).
Password:	Writable	The user's password (described in the <i>Usage Notes</i> below).
PasswordChange:	Read-only	The date and time of the user's last password change. If the user has no password, this field is blank.
Reviews:	Writable List	A list of files the user would like to review (see the <i>Usage Notes</i> below).

Options

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

Usage Notes

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

- The `-d` flag can be used by non-superusers only to delete the user specification that invoked the [p4 user](#) command. Perforce superusers can delete any Perforce user.
- User deletion fails if the specified user has any open files. Submit or revert these files before deleting users.
- By default, user records are created without passwords, and any Perforce user can impersonate another by setting [P4USER](#) or by using the *globally available* `-u` flag. To prevent another user from impersonating you, set a password with the [p4 passwd](#) command.

Passwords can be created, edited, or changed in the [p4 user](#) form or by using the [p4 passwd](#) command. Setting your password in the [p4 user](#) form is only supported at security levels 0 or 1. You can [p4 passwd](#) to set passwords at any server security level, and you *must* use [p4 passwd](#) to set passwords at higher security levels. For more about how the various security levels work, see the *System Administrator's Guide*.

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

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 be 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 server](#)
 - [p4 serverid](#)
 - [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 doesn't already exist as a Perforce user. If user joe2 already exists, allow a Perforce superuser to modify the user's settings.

Related Commands

To view a list of all Perforce users

[p4 users](#)

To change a user's password	p4_passwd
To view a list of users who have subscribed to review particular files	p4_reviews
To control how new users are created by changing the <code>dm.user.noautocreate</code> configurable	p4_configure

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

<i>-m max</i>	List only the first <i>max</i> users.
<i>-l</i>	Login information: includes time of last password change and login ticket expiry, if applicable. You must be a Perforce superuser to use this option.
<i>-a</i>	Include service users in list.
<i>-r</i>	On replica servers, only users who have used this replica server are reported.
<i>-c</i>	On replica servers, only user information from the master server are reported.
<i>g-opts</i>	See the Global Options section.

Usage Notes

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

- You 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](#)

Synopsis

Verify that the Perforce versioning service's archives are intact.

Syntax

```
p4 [g-opts] verify [-m maxRevs] [-q] [-s] [-t|-u|-v|-z] [-X] [-b batch]
    file[revRange]...
p4 [g-opts] verify -U unloadfile...
```

Description

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

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

In replicated environments, [p4 verify -t](#) reports **BAD!** or **MISSING!** files with **(transfer scheduled)** at the end of the line.

Options

-q	Run quietly; verify the integrity of files for which MD5 digests have previously been generated, and only display output if there are errors.
-s	Verify file size as well as digest. The -v implies the -s flag.
-t	When run on a replica, p4 verify -t causes the replica to schedule a transfer of the contents of any damaged or missing revisions. The -t option cannot be used with the -v or -u options.
-u	Store the filesize and MD5 digest of each file in the Perforce database if and only if no filesize and/or digest has been previously stored. Subsequent uses of p4 verify will compare the computed version against this stored version.
-v	Store the MD5 digest of each file in the Perforce database, even if there's already a digest stored for that file, overwriting the existing digest. (The -v flag is used only to update the saved digests of archive files that have been deliberately altered outside of Perforce control by a Perforce system administrator.) The -v and -u options are mutually exclusive.

-z	Skip revisions that have already been computed in the current pass; this option speeds verifications in the cases of revisions which exist via lazy copies. This option cannot be used with the -v or -u options.
-b batchsize	By default, p4 verify processes files in batches of 10000 files at a time. You can change this batch size with the -b batchsize flag. To disable batching, specify -b 0 . If the -z flag is specified, the -b flag is ignored and all flags are processed in a single batch.
-X	Skip files of filetype +X (for which the service runs an archive trigger.)
-U unloadfile	Verify files in the unload depot. See p4 unload for details.
-m maxRevs	Limit p4 verify to <i>maxRevs</i> revisions.
g-opts	See the Global Options section.

Usage Notes

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

- If [p4 verify](#) returns errors, contact Perforce technical support.
- It is good administrative practice to regularly verify the integrity of your depot files with **p4 verify -qz //...**

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

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

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

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 the [Global Options](#) section.

Usage Notes

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

- The mappings are derived from the client view: a simple 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

<code>p4 where file.c</code>	Show depot, client workspace, and local filesystem locations of <code>file.c</code> (or where <code>file.c</code> would appear if it existed in the depot.)
<code>p4 where 100%25.txt</code>	Use ASCII expansion of "%" character to locations for file <code>100%.txt</code> . 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](#)

Synopsis

Create or edit a client workspace specification and its view.

Syntax

```
p4 [g-opts] workspace [-f] [-t template] [workspacename]  
p4 [g-opts] workspace -o [-t template] [workspacename]  
p4 [g-opts] workspace -d [-f] workspacename  
p4 [g-opts] workspace -s [-S stream | -t workspacename] workspacename  
p4 [g-opts] workspace -S stream [-c change -o] workspacename  
p4 [g-opts] workspace -i [-f]
```

Description

The command [p4 workspace](#) is an alias for [p4 client](#).

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](#).

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	<code>P4CLIENT=<i>value</i> ; export P4CLIENT</code>
UNIX: csh	<code>setenv P4CLIENT <i>value</i></code>
VMS	<code>def/j P4CLIENT "<i>value</i>"</code>
Mac OS X (bash)	<code>P4CLIENT=<i>value</i> ; export P4CLIENT</code>
Max OS X (Settings)	<p><code>p4 set P4CLIENT=<i>value</i></code></p> <p>On OS X, you can also set the variables in the <code>com.perforce.environment</code> property list in your <code>~/Library/Preferences</code> folder.</p> <p>If you have administrative privileges, you can set the variables in the system <code>/Library/Preferences</code> folder with <code>p4 set -s <i>var=value</i></code>.</p> <p>These locations are reflected in the output of p4 set on Windows and OS X.</p>
Windows	<p><code>p4 set P4CLIENT=<i>value</i></code></p> <p>Windows administrators running Perforce as a service can set variables for use by a specific service with <code>p4 set -S <i>svcname var=value</i></code>, or set variables for all users on the local machine with <code>p4 set -s <i>var=value</i></code>.</p> <p>(See the p4 set command for more details on setting Perforce variables in Windows and OS X).</p>

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
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	

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	<code>p4d -A auditlog</code>	N/A

Value if not Explicitly Set

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

Notes

[P4AUDIT](#) specifies the location of the audit log file.

When auditing is enabled, 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 *Distributing Perforce*.

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.

For more information about central authorization servers, see *Distributing Perforce*.

Description

Set Perforce Broker options for a Windows service.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	p4broker %P4BROKEROPTIONS%	N/A

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

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

```
p4broker p4broker -c c:\p4broker\broker.conf
```

you can set the [P4BROKEROPTIONS](#) variable for the Windows service to run with

```
p4 set -S "Broker" P4BROKEROPTIONS= "-c c:\p4broker\broker.conf"
```

When you run P4Broker under the "Broker" service, the Broker will configure itself using the specified `broker.conf` file. Use [P4BROKEROPTIONS](#) when you need to call `p4broker` with flags for which there are no corresponding environment variables, or when you are doing so within the context of a Windows service.

For more information on the Perforce Broker, see *Distributing Perforce*.

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](#) on 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 *Distributing Perforce*.

Description

Character set used for translation of unicode files.

Usage Notes

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

Value if not Explicitly Set

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

Notes

[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 be set 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/doc.current/user/i18nnotes.txt>

For a complete list of valid [P4CHARSET](#) values, issue the command `p4 help charset`.

Description

Name of current client workspace.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
Windows	Value of <code>COMPUTERNAME</code> environment variable
All others	Name of host machine

Examples

`cinnamon`

`computer1`

`WORKSTATION`

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
```


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 <i>commandcharset cmd</i>	Yes

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

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

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

For a complete list of valid [P4COMMANDCHARSET](#) values, issue the command `p4 help charset`.

Description

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

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

Usage Notes

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

Value if not Explicitly Set

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

Examples

A sample [P4CONFIG](#) file might contain the following lines:

```
P4CLIENT=joes_client
P4USER=joe
P4PORT=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 variables to set within a [P4CONFIG](#) file include the following:

- [P4CHARSET](#)
- [P4CLIENT](#)
- [P4DIFF](#)
- [P4EDITOR](#)
- [P4HOST](#)

- [P4LANGUAGE](#)
- [P4MERGE](#)
- [P4PASSWD](#)
- [P4PORT](#)
- [P4TICKETS](#)
- [P4USER](#)

Description

Set Perforce server or proxy trace flags.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	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 flags are useful only to administrators working with Perforce Technical Support to diagnose or investigate a problem.

The preferred way to set trace flags for the Perforce server (or proxy) is to set them on the **p4d** (or **p4p**) command line. For technical reasons, this does not work for sites running Perforce servers or proxies as services under Windows. Administrators at such sites can use [p4_set](#) to set the trace flags within [P4DEBUG](#), allowing the service to run with the flags enabled.

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

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	<code>p4d -Id <i>description</i></code>	N/A

Value if not Explicitly Set

Operating System	Value
All	None

Examples

"Commit server"

"Replica server"

"Build farm"

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

Examples

```
diff
```

```
diff -b
```

```
windiff.exe
```

Notes

The value of [P4DIFF](#) can contain flags to the called program, for example, *diff -u*.

The commands [p4 describe](#), [p4 diff2](#), and [p4 submit](#) all use a diff program built into **p4d**. This cannot be changed.

Description

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

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

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

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

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, <code>vi</code> .
Windows	If SHELL is set to any value, then <code>vi</code> ; otherwise, <code>notepad</code>
VMS	If POSIX\$SHELL is set, then <code>vi</code> ; otherwise, <code>edit</code> .
Mac	If EDITOR_SIGNATURE is set, then the program with that four-character creator; otherwise, <code>SimpleText</code> .

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](#).

Description

Name of host computer to impersonate.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
Yes	No	<code>p4 -H <i>hostname</i> command</code>	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.

Description

Specify a file that contains a list of files to ignore 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

```
/users/edk/.p4ignore
```

Notes

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: files are specified in local syntax, a # character at the beginning of a line denotes a comment, a ! character at the beginning of a line excludes the file specification, and the * wildcard matches substrings. The Perforce wildcard of "... " 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
```


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	<code>p4d -J file</code>	N/A

Value if not Explicitly Set

Operating System	Value
All	P4ROOT /journal

Examples

journal

off

/disk2/perforce/journal

Notes

If a relative path is provided, it should be specified relative to the Perforce server root.

Setting [P4JOURNAL](#) to `off` will disable journaling. This is not recommended.

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

Description

This environment variable is reserved for system integrators.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	N/A

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	<code>p4d -L <i>file</i></code> <code>p4p -L <i>file</i></code>	N/A

Value if not Explicitly Set

Operating System	Value
All	Standard error

Examples

`log`

`/disk2/perforce/log`

Notes

If a relative path is provided, it is specified relative to the Perforce server root.

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

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 *Perforce System Administrator's Guide*.

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%
```


Description

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

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None.

Notes

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

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

Description

A unique identifiable name for a Perforce server.

[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](#).

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	None

Examples

masterserver

failoverserver

buildserver

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.

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	<code>p4 -P passwd command</code>	Yes

Value if not Explicitly Set

Operating System	Value
All	None

Notes

Perforce passwords are set via [p4 passwd](#), or in the form invoked by [p4 user](#). The setting of [P4PASSWD](#) is used to verify the user's identity. If a password has not been set, the value [P4PASSWD](#) is not used, even if set.

While it is possible to manually set the [P4PASSWD](#) environment variable to your plaintext password, the more secure way is to use the [p4 passwd](#) command. On UNIX, this will invoke a challenge/response mechanism which securely 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).

Description

For the Perforce Proxy, the directory in which the proxy stores its files and subdirectories.

Usage Notes

Used by Client?	Used by Server?	Command-Line Alternative	Can be set in P4CONFIG file?
No	Yes	<code>p4p -r <i>directory</i></code>	N/A

Value if not Explicitly Set

Operating System	Value
All	<p>p4p's directory.</p> <p>Windows administrators running the Perforce Proxy process as a service should use <code>p4_set -S <i>svcname</i> P4PCACHE=<i>directory</i></code> to set the value of P4PCACHE for the named service.</p>

Notes

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

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

For more information on setting up a Perforce Proxy, see *Distributing Perforce*.

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	<code>p4p -e size</code>	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 *Distributing Perforce*.

Description

Set Perforce Proxy options for a Windows service.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
All	Null

Notes

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

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

you can set the [P4OPTIONS](#) variable for the Windows `proxysvc` to run with

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

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

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

For more information on setting up a Perforce Proxy, see *Distributing Perforce*.

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.

P4PORT protocol value	Behavior in IPv4/IPv6 or mixed networks
ssl:	Use ssl4: behavior, but if the address is numeric and contains two or more colons, assume ssl6: If the <code>net.rfc3484</code> configurable is set, allow the OS to determine which transport is used.
ssl4:	Listen on/connect to an IPv4 address/port only, using SSL encryption.
ssl6:	Listen on/connect to an IPv6 address/port only, using SSL encryption.
ssl46:	Listen on/connect to an IPv4 address/port. If that fails, try IPv6. After connecting, require SSL encryption.
ssl64:	Listen on/connect to an IPv6 address/port. If that fails, try IPv4. After connecting, require SSL encryption.

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	<code>p4 -p protocol:host:port cmd</code>	Yes

Value if not Explicitly Set

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

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.

Description

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

Usage Notes

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

Value if not Explicitly Set

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

Notes

Create this directory before starting the Perforce 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 *Perforce System Administrator's Guide* and *Distributing Perforce*.

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 . 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 *System Administrator's Guide*.

- To generate a fingerprint from your server's private key and certificate, run `p4d -Gf`. ([P4SSLDIR](#) must be configured with the correct file names and permissions, and the current date must be valid for the certificate.)

After you have communicated this fingerprint to your end users, your end users can then compare the fingerprint the server offers with the fingerprint you have provided. If the two fingerprints match, users can use [p4 trust](#) to add the fingerprint to their [P4TRUST](#) files.

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	<code>p4p -t host:port</code> <code>p4d -t host:port</code>	N/A

Value if not Explicitly Set

Program	Value
Perforce Proxy	<code>perforce:1666</code>
Replicated environments	None

Examples

Perforce server examples
<code>1818</code>
<code>master:11111</code>
<code>perforce.example.com:1234</code>
<code>192.168.0.123:1818</code>

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 *Distributing Perforce*.

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.

Description

The trust file contains the fingerprints of the keys received 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?
No	Yes	None	N/A

Value if not Explicitly Set

Program	Value
All	\$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.

Description

Current Perforce username.

Usage Notes

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

Value if not Explicitly Set

Operating System	Value
Windows	The value of the <code>USERNAME</code> environment variable.
All Others	The value of the <code>USER</code> environment variable.

Examples

```
edk
```

```
lisag
```

Notes

By default, the Perforce username is the same as the OS username.

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

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

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

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	<code>p4 -d <i>directory</i> command</code>	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, <code>getcwd()</code> is used.
All Others	The actual current working directory.

Notes

Sometimes the **PWD** variable isn't inherited properly across shells. For instance, if you're running **ksh** or **sh** on top of **csh**, **PWD** will be inherited from your **csh** environment but not updated properly, causing possible confusion in subsequent Perforce commands.

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

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.

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

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

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

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 flag 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 marshalled Python dictionary objects. This is most often used when scripting.
-H host	Overrides any P4HOST setting and replaces it with the specified hostname.
-p port	Overrides any P4PORT setting with the specified 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** 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--%d--' % num
        dict = marshal.load(sys.stdin, 'rb')
        for key in dict.keys(): print "%s: %s" % (key,dict[key])

except EOFError: pass
```

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

- 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

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

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

Using revision specifiers

File specifiers can be modified by appending `#` or `@` to them.

The `#` and `@` specifiers refer to specific revisions of files as stored in the depot:

Modifier	Meaning
<code>file#n</code>	Revision specifier: The <i>n</i> th revision of <i>file</i> .
<code>file#none</code> <code>file#0</code>	The nonexistent revision: If a revision of <i>file</i> exists in the depot, it is ignored. This is useful when you want to remove a file from the client workspace while leaving it intact in the depot, as in p4_sync file#none . The filespec <code>#0</code> can be used as a synonym for <code>#none</code> - the nonexistent revision can be thought of as the one that "existed" before the first revision was submitted to the depot.
<code>file#head</code>	The head revision (latest version) of <i>file</i> . Except where explicitly noted, this is equivalent to referring to the file without a revision specifier.
<code>file#have</code>	The revision on the current client: the revision of file last p4_sync ed into the client workspace.
<code>file@n</code>	Change number: The revision of <i>file</i> immediately after changelist <i>n</i> was submitted.
<code>file@=n</code>	Change number: The revision of <i>file</i> at pending changelist number <i>n</i> .
<code>file@labelname</code>	Label name: The revision of <i>file</i> in the label <i>labelname</i> .
<code>file@clientname</code>	Client name: The revision of <i>file</i> last taken into client workspace <i>clientname</i> .

Modifier	Meaning
	Note that deleted files (that is, files marked for delete at their latest revision) are not considered to be part of a workspace.
<i>file@datespec</i>	<p>Date and time: The revision of <i>file</i> at the date and time specified.</p> <p>If no time is specified, the head revision at 00:00:00 on the morning of the date specified is returned.</p> <p>Dates are specified <i>yyyy/mm/dd:hh:mm:ss</i> or <i>yyyy/mm/dd hh:mm:ss</i> (with either a space or a colon between the date and the time).</p> <p>The datespec @now can be used as a synonym for the current date and time.</p>

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

If specifying files by date and time (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.txtp4 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.

Synopsis

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

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

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

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

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

Usage Notes

Views are set up through the [p4 client](#), [p4 branch](#), or [p4 label](#) commands as part of the process of creating a client workspace, label view, or branch view respectively.

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

```
//depot/...    //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/...
Some files in the depot are mapped to a different part of the client workspace	//depot/... //ws/... //depot/rel1/... //ws/release1/...

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

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	<code>//depot/main/... //depot/1.1dev/...</code>
Rearranging directories in the new release	<code>//depot/main/... //depot/1.1dev/... //depot/main/*.c //depot/1.1dev/src/*.c //depot/main/*.txt //depot/1.1dev/doc/*.txt</code>

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 doesn't copy any files; label views are used to limit the set of files that are taggable by a label.

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

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

Synopsis

Perforce supports six base file types:

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

File type modifiers are then applied to the base types allowing for support of RCS keyword expansion, file compression, 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.	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, please 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, please 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 <i>Internationalization Notes</i> .	RCS deltas in UTF-8 format
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. For details, see the <i>Internationalization Notes</i> .	RCS deltas in UTF-8 format

File type modifiers

The file type modifiers are:

Modifier	Description	Comments
+w	File is always writable on client	

Modifier	Description	Comments
+x	Execute bit set on client	Used for executable files.
+ko	Old-style keyword expansion	<p>Expands only the \$Id\$ and \$Header\$ keywords:</p> <p>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.</p>
+k	RCS keyword expansion	<p>Expands RCS (Revision Control System) keywords.</p> <p>RCS keywords are case-sensitive.</p> <p>When using keywords in files, a colon after the keyword (for instance, \$Id:\$) is optional.</p> <p>UTC keywords are better suited to describe events in globally distributed installations</p> <p>Supported keywords are as follows:</p> <ul style="list-style-type: none"> • \$Id\$ • \$Header\$ • \$Date\$ Date of submission • \$DateUTC\$ Date of submission in UTC time zone • \$DateTime\$ Date and time of submission • \$DateTimeUTC\$ Date and time of submission in UTC time zone. • \$DateTimeTZ\$ Date and time of submission in the server's time zone, but including the actual time zone in the result. • \$Change\$ • \$File\$ • \$Revision\$ • \$Author\$
+l	Exclusive open (locking)	If set, only one user at a time will be able to open a file for editing.

Modifier	Description	Comments
		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 aren't 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.
+Sn	Only the most recent <i>n</i> revisions are stored, where <i>n</i> is a number from 1 to 10, or 16, 32, 64, 128, 256, or 512.	Older revisions are purged from the depot upon submission of more than <i>n</i> new revisions, or if you change an existing +Sn file's <i>n</i> to a number less than its current value. Earlier revisions unaffected; see the Usage Notes for details.
+m	Preserve original modtime	The file's timestamp on the local filesystem is preserved upon submission and restored upon sync. Useful for third-party DLLs in Windows environments.
+X	Archive trigger required	The Perforce service runs an archive trigger to access the file. See the <i>System Administrator's Guide</i> for details.

A file's type is normally preserved between revisions, but can be overridden or changed with the **-t** flag during **add**, **edit**, or **reopen** operations:

- [p4 add](#) **-t filetype filespec** adds the files as the specified type.
- [p4 edit](#) **-t filetype filespec** opens the file for **edit** as the specified type. The file's type is changed to the specified **filetype** only after it is submitted to the depot.
- [p4 reopen](#) **-t filetype filespec** changes the type of a file already open for **add** or **edit**.

The **filetype** argument is specified as [**basetype**] **+modifiers**. For example, to change **script.sh**'s type to executable text with RCS keyword expansion, use [p4 edit](#) **-t text+kx script.sh**.

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

Perforce file types for common file extensions

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

File Type	Perforce file type	Description
.asp	text	Active Server Page file
.avi	binary+F	Video for Windows file
.bmp	binary	Windows bitmap file
.btr	binary	Btrieve database file
.cnf	text	Conference link file
.css	text	Cascading style sheet file
.doc	binary	Microsoft Word document
.dot	binary	Microsoft Word template
.exp	binary+w	Export file (Microsoft Visual C++)
.gif	binary+F	GIF graphic file
.gz	binary+F	Gzip compressed file
.htm	text	HTML file
.html	text	HTML file
.ico	binary	Icon file
.inc	text	Active Server Include file
.ini	text+w	Initial application settings file
.jpg	binary	JPEG graphic file
.js	text	JavaScript language source code file
.lib	binary+w	Library file (several programming languages)
.log	text+w	Log file
.mpg	binary+F	MPEG video file
.pdf	binary	Adobe PDF file
.pdm	text+w	Sybase Power Designer file
.ppt	binary	Microsoft Powerpoint file

File Type	Perforce file type	Description
.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 <i>YYYY/MM/DD</i>	\$Date: 2010/08/18 \$
\$DateTime\$	Date and time of last submission in format <i>YYYY/MM/DD hh:mm:ss</i> Date and time are as of the local time on the Perforce service at time of submission.	\$DateTime: 2010/08/18 23:17:02 \$
\$Change\$	Perforce changelist number under which file was submitted	\$Change: 439 \$
\$File\$	File name only, in depot syntax (without revision number)	\$File: //depot/path/file.txt \$
\$Revision\$	Perforce revision number	\$Revision: #3 \$
\$Author\$	Perforce user submitting the file	\$Author: edk \$

Usage Notes

- The type of an existing file can be determined with [p4 opened](#) or [p4 files](#).
- *Delta storage* (the default mode with **text** files) is a method whereby only the differences (or *deltas*) between revisions of files are stored. *Full file storage* (the default mode with **binary** files) involves the storage of the entire file. The file's type determines whether full file or delta storage is used. Perforce uses RCS format for delta storage.
- Some of the file types are compressed to **gzip** format for storage in the depot. The compression occurs during the submission process, and decompression happens while syncing. The process is transparent to the user; the client workspace always contains the file as it was submitted.
- Symbolic links in non-UNIX client workspaces appear as small text files containing a relative path to the linked file. Editing these files on a non-UNIX client should be done with caution,

as submitting them to the depot may result in a symbolic link pointing to a nonexistent file on the UNIX workspace.

- Changing a file's type does not affect earlier revisions stored in the depot.

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

- Running [p4 integrate](#) on temporary object files (**+S** and **+Sn**) does not produce a lazy copy; the integrated **tempobj** file consumes additional disk space 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

Old Keyword	Description	Base Filetype	Modifiers
xbinary	Executable binary file	binary	+x
ctext	Compressed text file	text	+C
ctxtext	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
ubinary	Uncompressed executable binary file	binary	+Fx
tempobj	Temporary object	binary	+FSw
ctempobj	Temporary object (compressed)	binary	+Sw
xtempobj	Temporary executable object	binary	+FSwx
xunicode	Executable unicode	unicode	+x
xutf16	Executable UTF-16	utf16	+x

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/>)

