Perforce 2008.2
Introducing Perforce

December 2008

Table of Contents

Introducing Perforce

How Perforce Works

Perforce is a Software Configuration Management (SCM) system based on a client/server architecture. Users of Perforce client programs connect to a Perforce server and use Perforce client programs to transfer files between the server's file repository and individual client workstations.
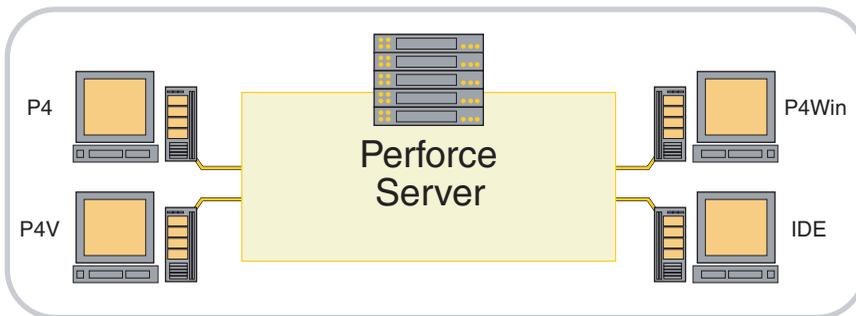
This document assumes that your Perforce server is already installed, configured and running. To set up a Perforce server, see the *System Administrator's Guide*.

The Perforce Server

The Perforce server manages the master file repository, or *depot*. There can be more than one depot per server. The depots contain every revision of every file under Perforce control. Perforce organizes files in depots into directory trees, like a large hard drive. Files in a depot are referred to as *depot files* or *versioned files*. The server maintains a database to track change logs, user permissions, and which users have which files checked out at any time. The information stored in this database is referred to as *metadata*.
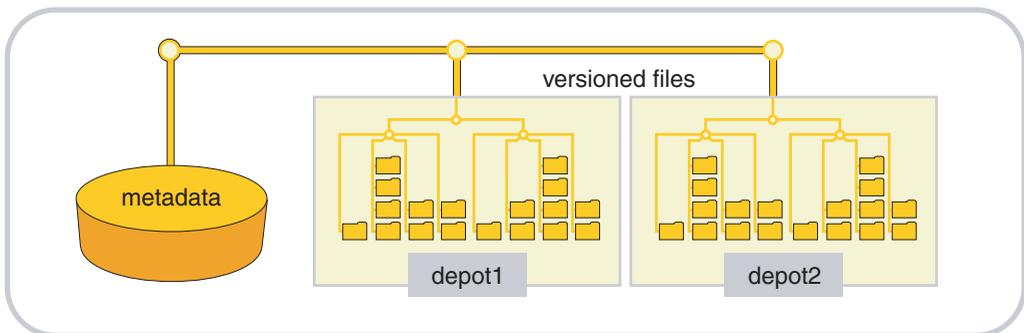
Perforce servers use native operating system capabilities to manage the database and the versioned files, and require no dedicated filesystems or volumes.

Perforce client programs

You use Perforce client programs to communicate with the Perforce server. Perforce client programs enable you to check files in and out, manage conflicts, create development branches, track bugs and change requests, and more. Perforce client programs include:

- P4, the Perforce Command-Line Client, for all platforms

- P4V, the Perforce Visual Client, for Mac OS X, UNIX, Linux, and Windows

- P4Win, the Perforce Windows Client

- P4Web, the Perforce Web Client, a browser-based client

- Integrations, or *plug-ins*, that work with commercial IDEs and productivity software

Under Perforce, you never work directly on files in the depot. Instead, you use a Perforce client program to manage a specially-designated area of your local workstation called a *client workspace.* A client workspace contains a local copy of a portion of the depot.
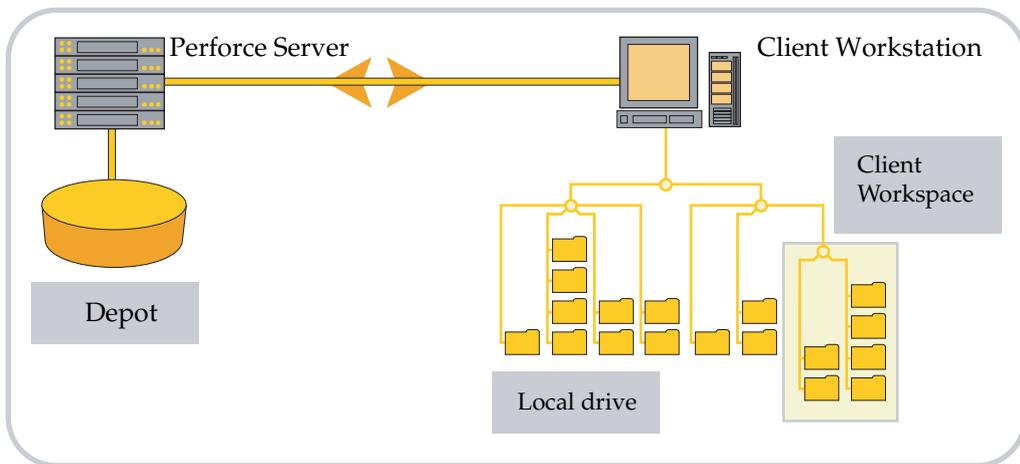
When you retrieve files into a client workspace, your Perforce client program requests the files from the Perforce server. To keep network traffic to a minimum, the Perforce server keeps track of what files you (and other users) have retrieved. Perforce client programs do not require a persistent connection to the Perforce server.

To use Perforce, you must set up your Perforce client program to connect to your organization's Perforce server, tell your client program where your client workspace is located on your local hard drive, and tell your client program what files from the depot you plan to work with.

Connecting to a server

To work with Perforce, you must connect to a Perforce server. Your Perforce client program communicates with Perforce servers over a TCP/IP connection. Each Perforce client program needs to know the address and port of the Perforce server with which it communicates.

The address and port are stored in the P4PORT environment variable; depending on the Perforce client you are using, the process of configuring this variable is referred to as "*setting your port*", or "*connecting to the server*".


```
Perforce Server                    Port Setting
IP address: 192.168.0.10           P4PORT=192.168.0.10:1666
machine name: p4dserv              P4PORT=p4dserv:1666
listening on: port 1666
```
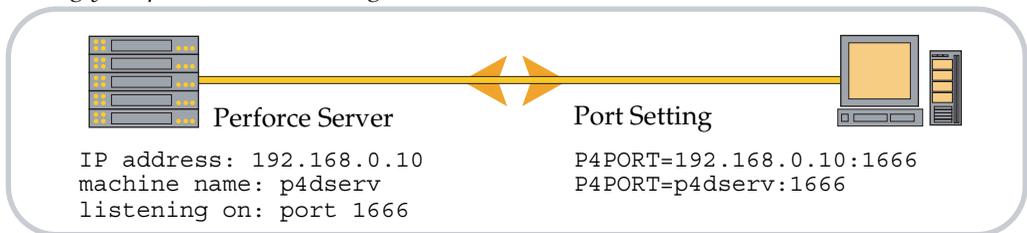
The documentation and the online help for your client program contain information on how to set your port. If you don't know the port setting used for connecting to your organization's Perforce server, ask your Perforce administrator.

Mapping files in the depot to your client workspace

Perforce client programs manage files in a designated area of your local disk, called your *client workspace*. Your client workspace is where you will be doing most of your work. You can have more than one client workspace, even on the same computer. The top level directory of any client workspace is called the *client workspace root*.

To control where the depot files appear under your client workspace root, you must map the files and directories on the Perforce server to corresponding areas of your local hard drive. These mappings constitute your *client workspace view*.

Client workspace views:

• Determine which files in the depot can appear in a client workspace.

• Map files in the depot to files in the client workspace.

Client workspace views consist of one or more lines, or *mappings*. Each line in your workspace view has two sides: a "*depot side*" that designates a subset of files within the depot and a "*client side*" that controls where the files specified on the depot side are located under your client workspace root.

Client side

Depot side
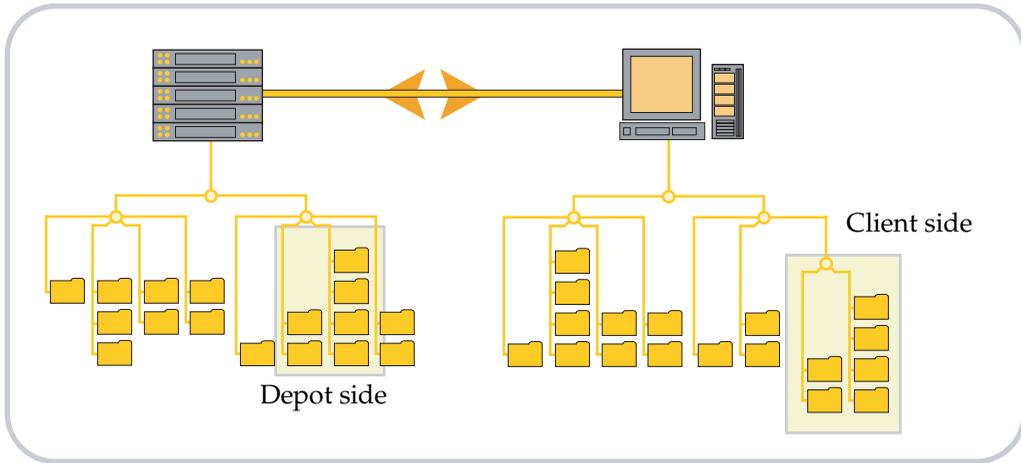
Creating a client workspace view doesn't transfer any files from the server to your computer. The view only sets up the mapping that controls the relationship between the depot and your client workspace when files are transferred.

To learn more about how to set up the mappings that define your client workspace, see the documentation and the online help for your Perforce client program.

Other configuration options

Other options for your Perforce client enable you to control the default behavior of various operations within Perforce. For instance, you can control carriage return/linefeed translation for cross-platform development, or select a preferred text editor or merge utility for use within Perforce. To learn more about these and other options, see the documentation for your particular Perforce client program.

Working in Perforce

Getting files from the server

The Perforce server manages the depot, an archive of every version of every file in the system. Your client workspace has a client workspace view that maps a subset of the depot's files to an area of your computer.

To populate your client workspace with the depot files, you must retrieve them from the server. In Perforce, updating your workspace with files from the server is often referred to as *syncing your workspace*. Other systems call this "refreshing", "getting the tip revision", or just "getting files".

Syncing your workspace

When you sync your workspace, your Perforce client program uses your client workspace view to map files in the depot to files in your client workspace, compares the result against the contents of your client workspace, and then adds, updates, or deletes files in your workspace as needed to bring your workspace contents in sync with the depot.

Syncing your workspace retrieves a copy of the latest ("head") revision of each file. (Other SCM systems might refer to this as the "tip" revision.)
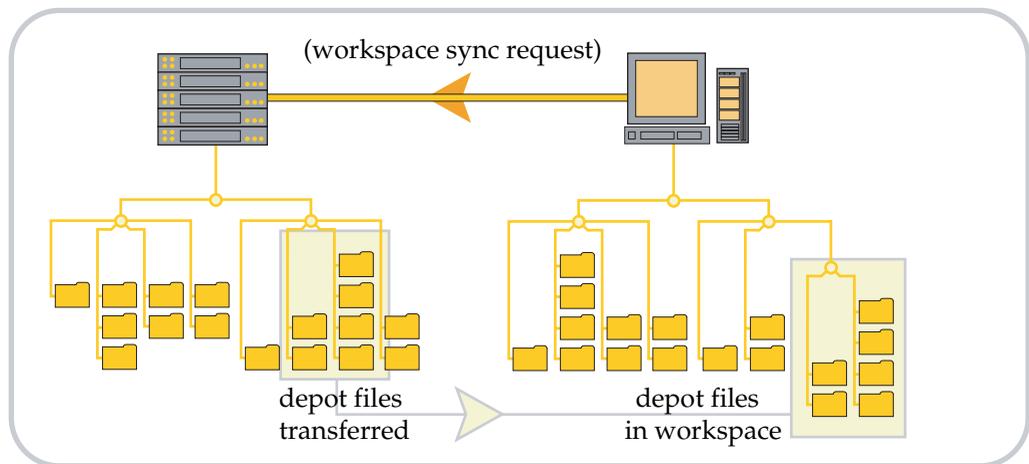
Perforce client programs manage file permissions in your workspace. By default, files synced to your workspace are read-only, and become writable when you check them out for editing.

Perforce client programs also support options that enable you to retrieve earlier revisions of files, or the revisions of files stored as of specified points in time, or sets of revisions that other users have tagged, or *labeled* with a user-defined identifying label.

Referring to files in Perforce

Perforce organizes files in the depot using directory hierarchies, like a large hard drive. Perforce client programs use a set of rules that define the specification of files in the depot and in your workspace. Whether you are setting up the mapping (the client workspace view) between the depot and your workspace, loading your workspace with files from the depot, or checking files in or out, these rules for referring to files are common across all operating systems.

Perforce syntax

When you refer to files in Perforce, you can specify files relative to a client workspace root ("client syntax"), or to the top of the depot tree ("depot syntax"), or by absolute and or relative paths on your local file system ("local syntax").

Files specified in client syntax or depot syntax always begin with two slashes (//), followed by the client workspace or depot name, and the full pathname of the file, relative to the client workspace root or top of the depot tree. Path components in client and depot syntax are always separated by forward slashes (/), regardless of the component separator used by the local operating system.

Syntax	Example
Depot syntax	`//depot/main/src/file.c`
Client syntax	`//myworkspace/module/file.c`
Local syntax	`C:\Projects\working\module\file.c`

When mapping depot files to the local hard drive, the client workspace name is an alias for the client workspace root.

```
Workspace:   myworkspace

      Root:   C:\Projects\working\

      View:   //depot/main/src/...   //myworkspace/module/...

  Location:   C:\Projects\working\module\file.c
```

For example, if the client workspace is named `myworkspace`, and the workspace root is `C:\Projects\working`, then the mapping specified by the view

```
    //depot/main/src/... //myworkspace/module/...
```

maps the depot file `//depot/main/src/file.c` into the client workspace as `C:\Projects\working\module\file.c`.

Using wildcards in views

You can use these wildcards when configuring workspace views in Perforce.

Wildcard	Meaning	Example
`*`	Matches all characters except slashes within one directory.	`/src/*.c` matches `/src/file.c` and `/src/file2.c`, but not `/src/lib/file.c`
`...`	Matches all files under the current working directory and all subdirectories.	`/src/...` matches all files and all subdirectories in and under `/src`
`%%1 - %%9`	Positional specifiers that replace portions of filenames in views.	Mapping `/%%1/%%2/...` to `/%%2/%%1/...` maps `/web/images/file.gif` to `/images/web/file.gif`

These wildcards are also used when specifying files in the Command-Line Client. For more about Perforce syntax and wildcards, see the *Command Reference*.

Referring to specific revisions of files

Perforce uses the # character to denote file revisions. File revisions in Perforce are sequentially-increasing integers, beginning from #1 for the first revision, and so on. The most recent revision of a file is the highest-numbered revision of that file on the server, and is called the *head revision*. The revision you last synced to your workspace is called the *have revision*. The zeroth revision of a file is called the *null revision*, and contains no data.

When you work in Perforce, development branches (or *codelines*) are represented as directory paths. Files in different codelines have their own set of revision numbers, starting at revision #1 and increasing upwards. The ancestry of files in different codelines is preserved in *integration records*. To learn more about branching in Perforce, see "Branching and integration" on page 19.

The indicator `file.c#3/4` shows that you currently have revision #3 of `file.c` synced to your workspace, and that the most recent revision of `file.c` is #4. In contrast to other SCM systems, Perforce does not use perturbed version numbers (for instance, "revision `1.2.3` of `file.c`") to denote revisions of files in different development branches.

Syntax	Refers to	Remarks
`file.c#3`	The third revision of `file.c`	"Sync to the third revision of `file.c`".
`file.c#head`	The most recent revision of *file.c* stored on the server; this is the *head revision*.	To get the latest versions of files from the depot, you *sync your workspace to the head revision*.

Syntax	Refers to	Remarks
`file.c#have`	The revision of `file.c` last synced to your workspace; this is the *have revision*.	When you discard changes to a file, your Perforce client program reverts the copy of the file in your workspace to the *have revision*.
`file.c#none` `file.c#0`	The nonexistent, or *null revision*, of `file.c`.	When you use a Perforce client option to *remove files from your workspace*, you are actually syncing the revision of the file in your workspace to the *null revision*.

Perforce syntax and the status bar

The syntax for the head, have, and null revisions (`#head`, `#have`, and `#none`) is used in the Command Line Client and in the status window of graphical client programs. See the *P4 User's Guide* for details.

What file types are supported?

Perforce file types include six base file types.

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

By default, when anyone adds a file to the depot, Perforce attempts to determine the type of the file automatically. You can change a file's type by opening it for edit as the new file type. If the file is already open for edit, you can reopen it as the different file type.

The six base file types can have modifiers (such as `+w`, `+x`, `+k`, and others) applied that control such things as locking behavior, file permissions within a client workspace, or how revisions are stored on the server. The *Command Reference* contains complete list of file types and applicable modifiers.

Working with files

The changelist is the basic unit of work in Perforce. The basic file editing operations common to all SCM systems (such as editing files, adding files, deleting files, backing out changes, and checking in files) are performed in changelists.

Using changelists

After you have set up your workspace view and synced your workspace to the depot, you can start working in Perforce. Before you can work on a file in your workspace, you must use your Perforce client program to open the file in a changelist. A changelist consists of a list of files, their revision numbers, the changes you have made to the files, and a description that you supply that describes the work you performed.

Changelists serve two purposes:

- to organize your work into logical units by grouping related changes to files together

- to guarantee the integrity of your work by ensuring that related changes to files are checked in together

If you are working on a change to some software that requires changes to three files, open all three files in one changelist. When you check the changelist back into the depot, other users will see your changelist linked to the changes made to all three files.

Perforce changelists are *atomic change transactions*; if a changelist affects three files, then the changes for all three files are committed to the depot, or none of the changes are. Even if the network connection between your Perforce client program and the Perforce server is interrupted during changelist submission, the entire submit fails.

How changelist numbers work

A changelist with changes not yet submitted to the depot is a *pending changelist*. A changelist containing changes that have been committed to the depot is a *submitted changelist*. Each changelist has a *changelist number* (generated by Perforce), and a *changelist description* (supplied by the user who performed the changes).
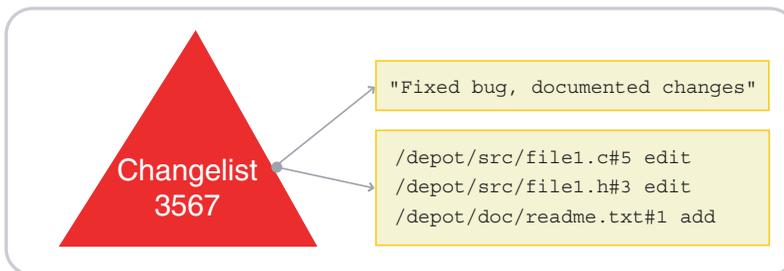
When you open a file in Perforce, the file is opened in a *default changelist*. The default changelist is assigned a changelist number when you check its files back into the depot.

You can partition your work in progress into multiple pending changelists. Pending changelists other than the default changelist are assigned numbers when you create the changelist. (A new number may be assigned to a pending changelist when you submit the changelist to the depot.)

Editing files

To edit a file, you check out the file in a changelist. Your Perforce client program makes the copy of the file in your client workspace writable, and informs the Perforce server that you have opened the file for editing.

For your changes to be available to other users, you must submit the changelist back to the depot. After your changelist has been submitted, other users can sync their workspaces and obtain their own copies of your changes.

Adding new files

To add a file, you create a file in your client workspace and mark the file for add in a changelist. Your Perforce client program determines the file's type (you can override this file type), and informs the Perforce server that you intend to add a file.

For your new file to be available to other users, you must submit the changelist with the added file back to the depot. After the changelist has been submitted to the depot, other users can sync their workspaces and obtain their own copies of the new file.

Deleting files

To delete a file, you mark the file for delete in a changelist. The file is deleted from your workspace immediately. Your Perforce client informs the server that you intend to delete a file, but the file is not marked as deleted in the depot until you submit the changelist.

After you have submitted the changelist to the server, other users see your file marked as deleted. Local copies of the deleted file remain in other users' workspaces until those users sync their workspaces to the depot.

Deleted file revisions are never actually removed from the depot. You can always recover older revisions of deleted files by syncing revisions that predate the file's deletion into your client workspace.

Discarding unwanted changes

You can discard any changes you made to a file in a changelist by *reverting* the file. Reverting a file removes the file from its changelist and restores the copy of the file in your client workspace to the revision last synced to your workspace.

If you revert a file opened for edit or marked for delete, whatever version of the file you last synced is restored to your workspace. If you revert a file marked for add, the file is removed from your changelist, but your local copy of the file remains in your client workspace.

Checking in files

When you are satisfied with the changes you have made to the files you opened and want your work to be available to others, check your work back in to the depot by *submitting* the changelist.
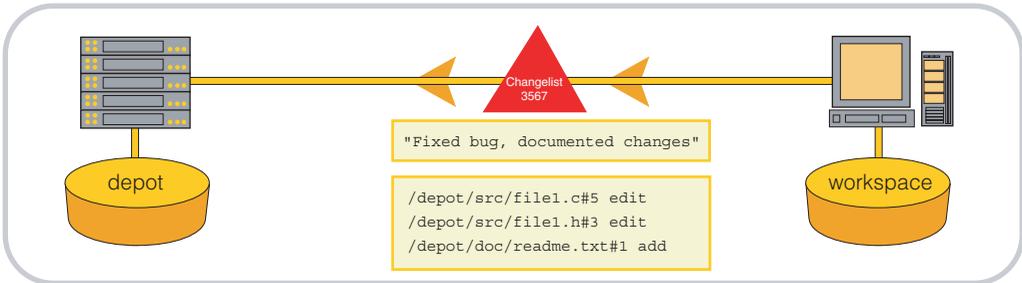

There is no such thing as a partially-submitted changelist. Changelist submission is an atomic transaction; either all of the files in a changelist are submitted successfully, or none are.

Resolving conflicts

When two users edit the same file at the same time, their changes can conflict. If your changes conflict with earlier changes submitted by another user, Perforce requires that you *resolve* the conflicting files and *re-submit the changelist*. Because changelists are atomic transactions, until you resolve the conflict, none of the changes to any of the files in your changelist can appear in the depot.

The resolve process enables you to decide what needs to be done: should your file overwrite the other user's? Should your own file be thrown away in favor of the other user's changes? Or should the two conflicting files be merged into one file? At your request, Perforce can perform a three-way merge between the two conflicting text files and the file from which the two conflicting files were derived.
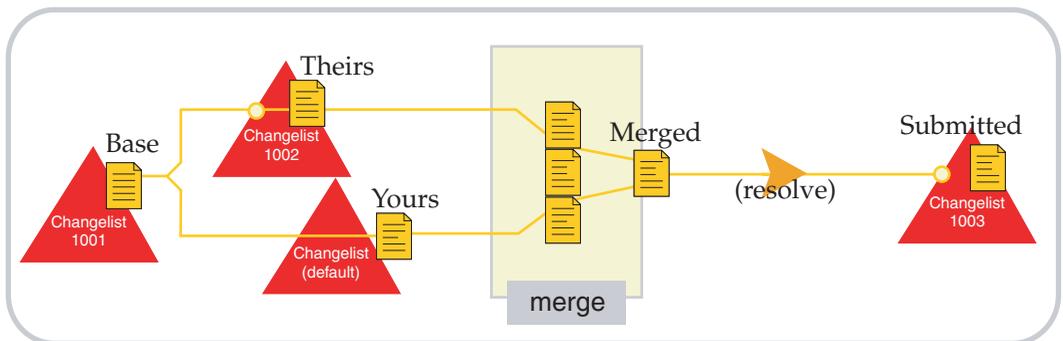
Working concurrently

Perforce helps teams to work concurrently. The conflict resolution and three-way merge process enables multiple users to work on the same files at the same time without interfering with each other's work.

The three-way merge process for resolving file conflicts helps you to resolve conflicting changes to text files, but is not necessarily meaningful for `binary` files such as graphics or compiled code. If you are working on files where merges are not meaningful, you can lock such files to prevent others from making changes that conflict with your work.

Perforce supports two types of file locking. You can prevent files from being checked in with *file locking* and you can prevent file checkout with *exclusive-open*:

- To prevent other users from checking in changes to a file you are working on, *lock the file*. Other users can still check out your locked file, but are unable to submit changelists that affect your locked file until you submit your changes. (To allow users to submit changelists that affect your locked file before you submit your work, unlock the file.)

- To prevent a file from being checked out by more than one user at a time, use the `+l` *exclusive-open filetype modifier*. Files that have the `+l` filetype modifier can only be opened by one user at a time. Your Perforce administrator can use a special table called the *typemap table* to automatically specify certain file types as exclusive-open.

For example, users working within an IDE that does not permit change resolution might also want to lock the files they're working on so they don't have to switch to a Perforce client program to submit their work, and users working on digital assets might want to automatically classify all `.gif` or `.mpg` files as exclusive-open.

If you are editing	Locked?	Meaning
`file` (*type*)	unlocked	Anyone can check out `file` and submit their changes.
		If a user submits changes to `file` while you have `file` open, you must resolve your changes against their changes when you submit your changelist.
`file` (*type*)	locked	Anyone can check out `file`, but no users can submit changes to `file` until you submit your changes to `file`, or until you remove the lock on `file`.
`file` (*type*+l)	unlocked or locked	Only one user at a time can have `file` open in a changelist. The status of the lock is irrelevant; no other users can submit changelists involving the file because no other users can check out the file.

For more about locking files, the exclusive-open filetype modifier, and the typemap table, see the *Command Reference* and the *System Administrator's Guide*.

Comparing files

You can use Perforce to compare any two revisions of the same file, of any two files in the depot, or of files in the depot and their corresponding copies in your workspace.

The `p4 diff` and `p4 diff2` commands produce output similar to that of the standard diff program included in UNIX and Linux systems. Other Perforce client programs (including P4V and P4Win) include P4Merge, which provides a graphical view of file differences. For example:

Reviewing change histories of individual files

The history of a file is represented by a series of file revisions, one per file. Each revision to the file is associated with a changelist. You can compare files against the revision in your workspace or against any of the revisions stored in the depot.

This P4V screenshot shows that the depot holds three revisions of the file `//depot/dev/main/jamgraph/gparticle.cpp`. The most recent revision, #3, was submitted in changelist 720.

Reviewing change histories of groups of files

The history of a directory is represented by a series of changelists. Directories do not have individual revision numbers; rather, every changelist that includes at least one file is considered to be part of a directory's history.
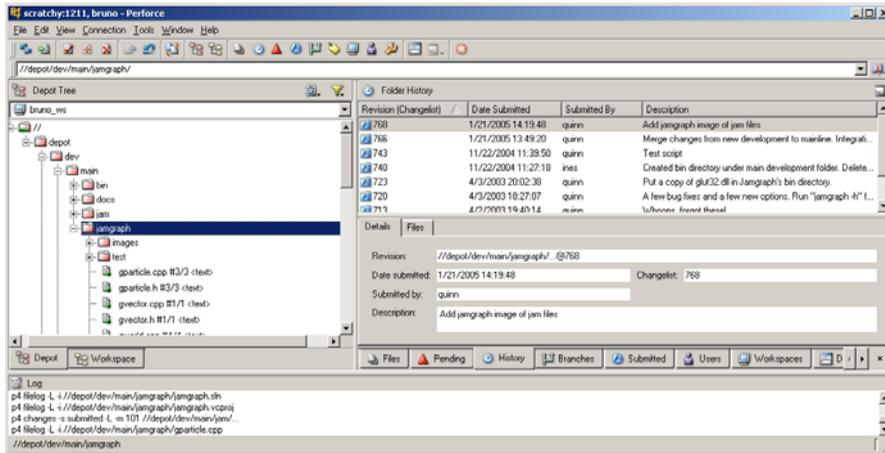
This P4V screenshot shows that the most recent changelist that affected at least one file in `//depot/dev/main/jamgraph` was changelist #768.

Perforce syntax and the status bar

Perforce has forms of syntax for referring to a file as it exists in the depot upon submission of a numbered changelist, as tagged by a mnemonic label, or as of certain dates and times. These forms of syntax (`@changelist`, `@labelname`, or `@date`, or `#start,end`) are typically used only with the command line client, but they also appear in the status window of graphical client programs. See the *P4 User's Guide* for more details.

Branching and integration

To structure groups of related files by purpose, such as a new product or release, you use branching. Branching is a method of managing changes between two or more sets of related files. Perforce's *Inter-File Branching* mechanism enables you to copy any set of files to a new location in the depot by allowing changes made to one set of files to be copied, or *integrated*, to the other. The new file set (or *codeline*) evolves separately from the original files, but changes in either codeline can be propagated to the other by means of *integration*.

Most software configuration management systems support some form of branching; Perforce's mechanism is unique because it mimics the style in which users create their own file copies when no branching mechanism is available.

Suppose for a moment that you're writing a program and are not using an SCM system. You're ready to release your program: what do you do with your code? Chances are that you'd copy all your files to a new location. One of your file sets becomes your release codeline, and bug fixes to the release are made to that file set; your other files are your development file set, and new functionality to the code is added to these files.

In Perforce's terminology, each set of files constitutes a *codeline*, and copying a change from one set of files to another is called *integration*. The entire process is called *branching*.

Perforce organizes files in the depot using directory hierarchies, like a large hard drive. When you make a new codeline, it appears in the depot as a subdirectory, such as `//depot/dev/main/jam` for ongoing development work, `//depot/release/jam/2.1` for release 2.1, and `//depot/release/jam/2.2` for release 2.2.
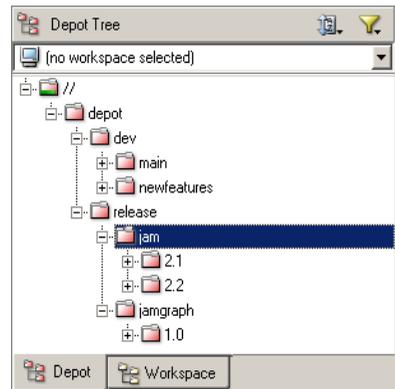
Creating a codeline

To create a codeline or development branch, decide which files belong in the branch (the *source files*), and *integrate* those files into the new codeline to create the *target files*. The Perforce server "opens the target files for `branch/sync`" in a changelist.

Opening files for `branch/sync` is just like opening them for `add`, `edit`, or `delete`; the files are opened in a changelist, and your client workspace view must include the target files. Similarly, no changes are made to the depot until you submit the changelist. The atomic nature of changelists ensures that when you create a codeline, it contains all of the files you branched.

Without an SCM system, you might create a branch by copying the files from one directory into another directory. The advantage of integration over copying the files and

adding the copies to the depot in a new directory is that when you integrate files from one codeline to another, Perforce can track the connections between related files in an *integration record*, facilitating easy tracking and propagation of changes between the two sets of files.

Integration also enables Perforce to perform a "lazy copy" of the files. When you branch files, the server does not actually hold two copies of the files - it merely holds the source file and a pointer in the database records the fact that the branch to the target file has occurred. Lazy copies make branching a low-overhead operation; the server doesn't have to keep track of duplicate copies of files.
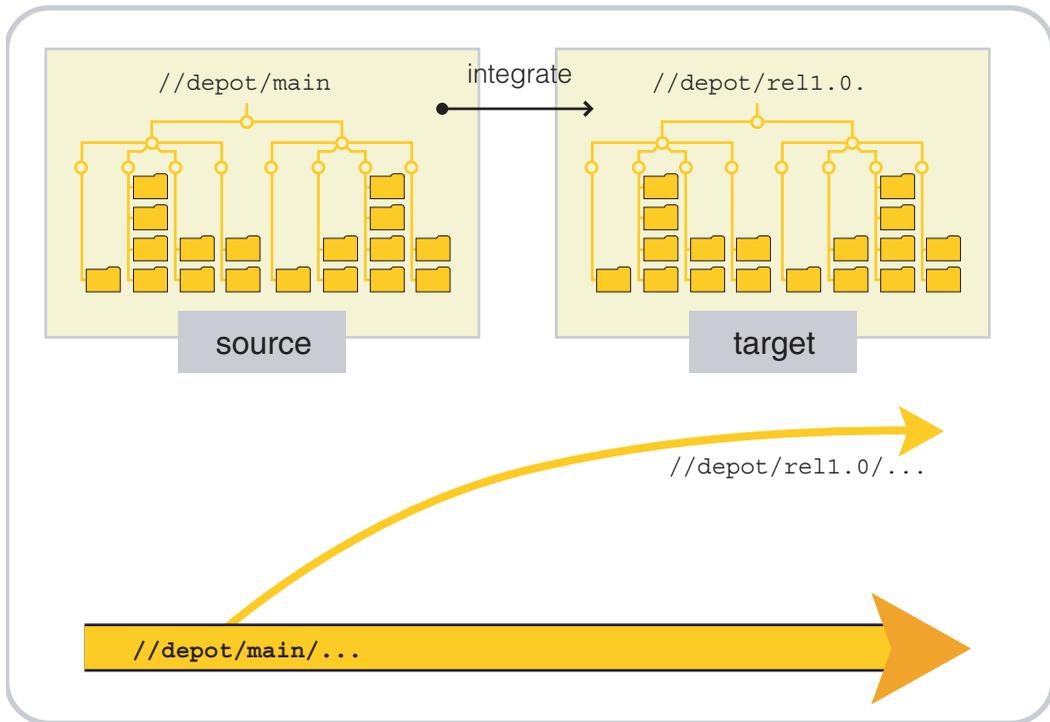
To integrate files from a source codeline to a target codeline:

- the target must be in your client workspace view

- the source doesn't have to be in your client workspace view (although you must have permission to read the source files)

- you open files for branch in a new changelist by integrating them

- you create files in the new codeline by submitting the changelist

- when you submit the changelist with the target files, the target files in the new codeline are at revision #1

- integration records enable you to examine the history of files in the new codeline, including the fact that they were created by means of integration from the source files.

Propagating changes between codelines

You can use integration to propagate changes between related codelines in much the same way you create codelines. (Creating a codeline is equivalent to propagating a set of changes that make up the entirety of the source files into an empty set of target files.)

When you create a codeline, the target files are by definition empty; there is no possibility your changes can conflict. When you propagate changes between existing codelines, conflicts can arise because conflicting changes may have been made in both the source and the target codelines.
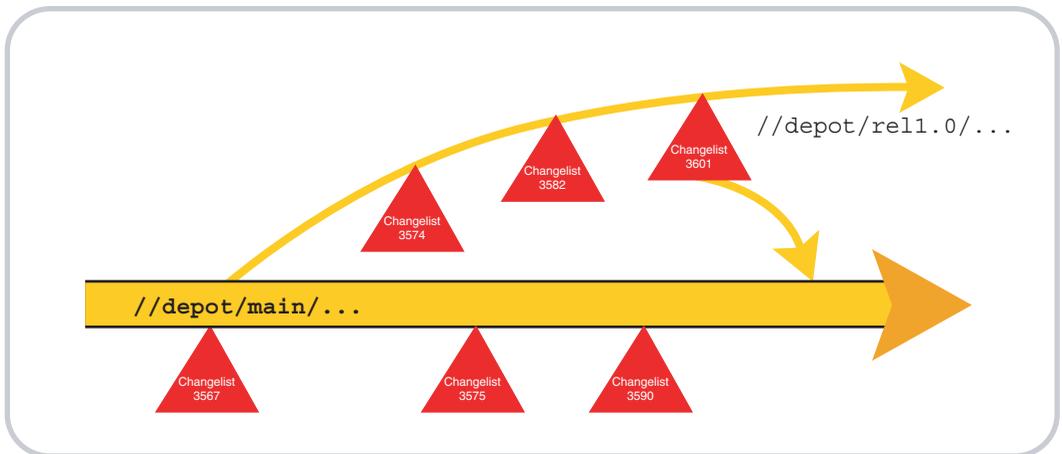
In the example shown, the `rel1.0` codeline was created by branching source files from `//depot/main` into a target of `//depot/rel1.0` in changelist 3567. Changelists 3574, 3582 and 3601 represent work performed in the release branch, and changelists 3575 and 3590 represent work performed in the main line.

In order to propagate work done in the release branch back into the main line, you integrate from source files in `//depot/rel1.0` into `//depot/main`, resolving any conflicting changes between work done in the release branch and work done in the main line.

Resolving differences between codelines

When you integrate changelists from a source codeline to an existing target codeline, Perforce *schedules a resolve*, or a *three-way merge*, between the files.

The revisions in the target files in your client workspace are referred to as *yours*. The revisions of the source files in the depot are referred to as *theirs*. Where changes between these files do not conflict, the changes can be merged automatically. Where changes conflict, you must choose which changes are to be accepted into the file.

Because changelists are atomic, you must resolve every file in a changelist before the submit can succeed. You can resolve these situations in one of three ways:

- Automatically: In many cases, you will know whether you want to accept the changes that are "yours" (that is, the target revisions in your client workspace) or "theirs" (that is, the source revisions in the depot). Whether you accept "yours" or "theirs", this type of resolve is referred to as an *automatic resolve without merging*.

- Accept merged: Sometimes, there are changes made to the files that are "theirs" and "yours" do not conflict. In these cases, Perforce merges the two files and provides you with an option to accept the merged result. Such a resolve is referred to as a "safe" *automatic resolve with merging*.

- Manual merge: Finally, there may be cases where the same lines in "theirs" and "yours" have been changed. Such lines are said to conflict. When changes conflict, Perforce resolves as many differences as possible and produces a merged file containing conflict markers for manual resolution. You must either edit the merged file manually before submitting it, or accept the merged file with the conflict markers included, and fix the conflict in a subsequent changelist.

Duplicating complex branch structures

Perforce provides two mechanisms for branching: *integrating using a file specification*, and *integrating using a branch specification*.

For simple branch structures, you can manually specify the paths of the source and target files, and integrate your source files to the target branch using file specifications. You must manually specify the source and target codelines every time you branch with a file specification.

For more complex branch structures, you can set up *branch specifications* that enable you to reliably duplicate even the most complex branch structures. A branch specification holds a set of mapping rules (a *branch view*) that controls how files in the source branch are integrated in the target branch. After you have set up a branch specification, you can perform the integration by using the branch specification to perform all of the integrations specified in the branch view.

For example, the branch specification in the screenshot shows three mappings:.

1.   a mapping of all files in
     `//depot/dev/main/jamgraph`
     into
     `//depot/release/jamgraph/1.0`
     directory

2.   an exclusionary mapping to
     ensure that test work in
     `/jamgraph/test` is not copied
     from the main line

3.   a mapping to include a DLL
     (`glut32.dll`) deliverable located
     in a common `/bin` build directory
     unrelated to the `jamgraph` project.

Tracking change history between codelines

The Revision Graph feature of P4V (also available in P4Win) is a convenient way of visualizing a file's history across (and between) branches.
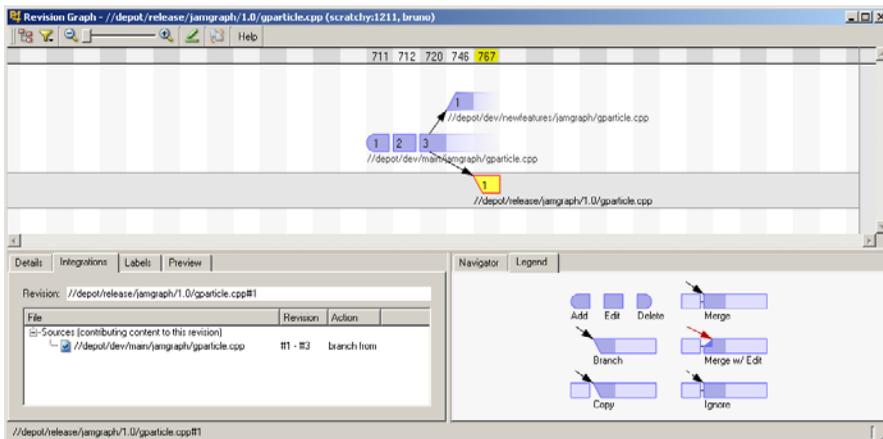
The example screenshot shows a simple revision graph. The changes to the file represented by revision #1 through revision #3 were integrated from the main codeline (`//depot/dev/main/...`) into a release branch (`//depot/release/jamgraph/1.0/...`) and into a development branch (`//depot/dev/newfeatures...`).

To learn more about branching

Although Perforce's branching mechanism is relatively simple, the theory of branching can be very complex. When should a branch be created? At what point should code changes be propagated from one codeline to another? Who is responsible for performing merges? These questions are common to every SCM system, and the answers are not simple.

A white paper on *Inter-File Branching* that describes best practices for the use of Perforce's branching mechanism, as well as technical details, is available from:

```
http://www.perforce.com/perforce/branch.html
```

Next steps

Work and defect tracking

Perforce includes a basic defect tracking system called *jobs*. A Perforce job is a description of work to be done, such as a bug fix or a change request. Perforce's job tracking mechanism enables you to link one or more jobs to the changelists that implement the work specified in the jobs. Associating jobs with changelists helps teams know if and when work was completed, who performed the work, and what file revisions were affected by the work. Jobs linked to changelists are marked as closed when the changelist is submitted.

The types of information tracked by the jobs system can be customized; Perforce administrators can add, change, or delete fields used by Perforce jobs. See the *System Administrator's Guide* for details.

Perforce currently offers two independent platforms to integrate Perforce with third-party defect tracking and workflow systems. Both platforms allow information to be shared between Perforce's job system and external defect tracking systems.

P4DTG, the Perforce Defect Tracking Gateway, is a closed-source integrated platform that includes both a graphical configuration editor and a replication engine. For more information, see:

        http://www.perforce.com/perforce/products/p4dtg.html

P4DTI, Perforce Defect Tracking and Integration, is an open-source product available under a FreeBSD-like license. For more information, see:

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

Tagging files with labels

Perforce *labels* are sets of tagged file revisions, enabling you to reproduce specific groups of files within client workspaces. Labels differ from changelists in that a changelist number represents the state of all files in the depot at the time the changelist was submitted. Labels are used to tag arbitrary groups of files, even when those file revisions represent work that was submitted in more than one changelist.

Another difference between changelists and labels is that changelists are referred to by Perforce-assigned numbers, while labels take names chosen by users. For example, you might want to tag the file revisions that compose a particular release with the label `rel2.1`. At a later time, you can update the revisions tagged with `rel2.1` to reflect fixes performed in subsequent changelists, and retrieve all the tagged revisions into a client workspace by syncing the workspace to the label.

For more about labels, see the *P4 User's Guide*.

Editors and merge tools

Most Perforce client programs have options that enable you to specify a preferred text editor or merge tool. For example, the Command Line Client uses the environment variables P4EDITOR and P4MERGE to invoke a preferred editor or merge tool.

See the documentation for your particular Perforce client program for details.

Protections and permissions

Perforce provides a protection scheme to prevent unauthorized or inadvertent access to the depot. The protections determine which Perforce commands can be run, on which files, by whom, and from which client workstations. Perforce administrators can set protections by using the p4 protect command in the Perforce Command-Line Client, or by using the Administration Tool in P4V.

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

Users and licenses

Perforce servers are licensed according to how many users they support. This licensing information resides in a file called license in the server root directory. The license file is a plain text file supplied by Perforce Software. Without the license file, the Perforce server limits itself to two users and five client workspaces.

For further information, see the *System Administrator's Guide*, or contact technical support.

Where to learn more about Perforce

To obtain online help from within all Perforce client programs:

- Use the help menu from within graphical Perforce client programs

- Type p4 help from the command line for help with the Command-Line Client

Documentation for Perforce is available on the web at:

    http://www.perforce.com/perforce/technical.html

A list of technical notes and FAQs is available at:

    http://www.perforce.com/perforce/technotes.html

Perforce also offers training classes. For details, see:

    http://www.perforce.com/perforce/training.html

For an active mailing list where you can hear from other Perforce users and ask questions:

    http://maillist.perforce.com/mailman/listinfo/perforce-user

Perforce support personnel are also available for email and telephone support.

    http://www.perforce.com/perforce/support.html