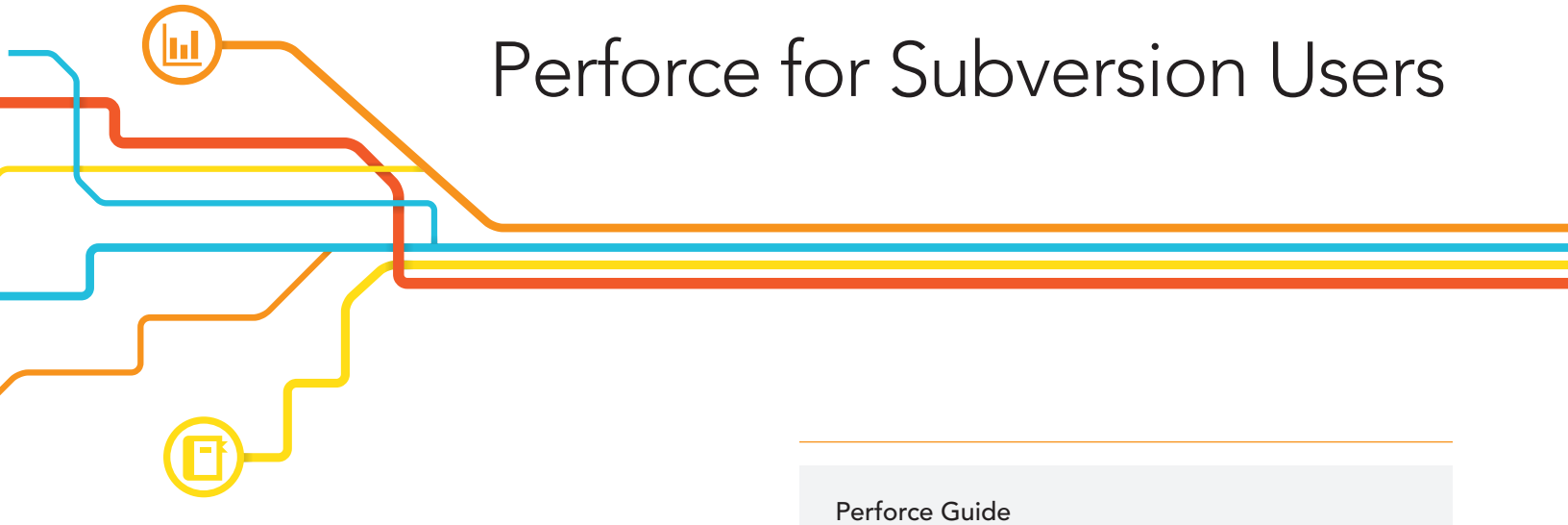
PERFORCE
Version everything.



Perforce for Subversion Users

Perforce Guide

This guide is designed to help Subversion users more quickly adopt Perforce version management. Use this guide to:

- Understand the main differences between Subversion and Perforce
- Adapt your development processes to use Perforce to its full potential

Table of Contents

INTRODUCTION

The purpose of this guide is to help you adopt Perforce version management if you are already familiar with Subversion. It explains the main differences between Subversion and Perforce so you can understand how to adapt your development processes to use Perforce to its full potential.

It is not meant as a replacement for the existing documentation but rather as a narrative. It should get you started quickly and highlight which parts of the documentation are initially most important to you.

Subversion and Perforce share many similarities, such as the central server and the local workspace or working copy, but there are distinct differences that you need to understand to become successful in using Perforce.

This guide is based on Subversion 1.8 and Perforce 2014.1

CONCEPT DIFFERENCES BETWEEN SUBVERSION AND PERFORCE

Connecting to a repository

Subversion uses standard URLs to indicate the location of a server on the Internet. Typically, the URL prefix is "http:" or "https:" (for a Subversion server embedded in an Apache Web server) or "svn:" (for a standalone Subversion server, "svnserve").

Perforce uses the scheme:

```
prefix:hostname:port
```

where the prefix is typically "tcp" and can be omitted. Other prefixes include "tcp6" for IPv6 and "ssl" for encrypted connections.

Although the Perforce server can use any port permitted by the OS, the typical port is "1666".

Repositories vs. depots

A Subversion server contains a set of independent repositories. A client can only connect to one repository at a time, but can reference other repositories through svn-externals.

Perforce depots are not independent. Their main purpose is to group projects under a common path and define a backend storage location. Files can be stored in two kinds of depots: standard (classical) depots and stream depots. Files can be branched and integrated (merged) from one depot to another, and users can access files from several depots at the same time.

Additionally, there are specialized depots such as *spec*, *unload*, and *archive* depots that developers do not use directly.

Authentication

Both Subversion and Perforce employ named users for all operations.

In Perforce, users first log in to the server and are issued a ticket that is valid for a certain amount of time, 12 hours by default.

Passwords in Perforce are either stored within the server itself or in an authentication agent such as LDAP or Active Directory.

Workspaces

To make a change to a file under source control, Subversion users check out a working copy onto their local machine. In Subversion, this working copy contains an administrative directory called ".svn" that holds the connection information and state.

Perforce uses a client workspace, which is a named entity of the server. A client workspace is often called a *client* or a *workspace* in Perforce. The command-line client, P4, allows both names; "p4 client" and "p4 workspace" are aliases.

There are no local hidden files; instead, the state of each workspace is kept on the server. A client workspace consists of a unique name, a root directory, a set of options, and a view.

The workspace view specifies the set of files in the depot that should be mapped to the local machine because you probably do not want all of the files that are available on the server. A workspace view lets you select just the set that you want.

Note that you can map content from multiple depots into a workspace, but can only map content from one server.

Many Perforce shops use streams, which can automatically generate a workspace view, or they generate the view using scripts or template workspaces. Many others let their users generate their own workspaces.

Why workspaces?

Workspaces are an amazingly powerful concept. They are flexible enough to allow users to define the layout of files in the depots to suit their needs. Different development platforms, builds, and deployment environments might have divergent needs in how files are organized locally.

Workspaces in Perforce are not only used to map the set of files a user wants to work with; the server can also track exactly which revisions of each file the user has synced. This approach allows the system to send the correct set of files to the user when syncing without having to scan the file system first to see which files need to be updated. With a large number of files, this can be a huge performance win. This is also very popular in industries that have very strict auditing rules—Perforce admins can easily track and log who has synced which files.

An advantage of being able to map a number of modules to one workspace is that you can easily modify multiple code modules in one check-in, guaranteeing that anyone with a similar client view who syncs to your check-in will have all the code in the correct state.

A client workspace first needs to be created on the server before it can be used. Most users will have more than one workspace on the same or separate machines.

Files under source control

Subversion stores text and binary files as well as empty directories.

The Perforce server does not version empty directories directly. If you require empty directories in your repository, place an empty hidden file into it (e.g., ".dir" or ".p4ignore").

Perforce distinguishes among text, binary, and Unicode-encoded files as well as symbolic links. Additionally, each version of a file can have extra attributes such as *executable* or *exclusive checkout*.

Pending changes and shelves

Subversion (since version 1.5) has the concept of a local changelist attached to a working copy that allows users to group changes under an arbitrarily named change.

Perforce stores its changelists on the server. It has the concept of a default pending change to which all changes in a workspace are attached. Each file operation such as adding, editing, or deleting a file is automatically attached to the pending changelist.

A pending change can also be explicitly saved; it is then given a unique change number. Each workspace can have unlimited numbered pending changes.

Similar to Subversion, a file can be in only one changelist at a time on the client. But Perforce allows the change to be shelved, which transfers the changed content to a central server where it can be inspected, compared, and reviewed by all users.

Storing a change

A change in Subversion is committed with a change comment and is then visible to all other users. It is possible to only commit a single named changelist. Once committed, the change is given a repository-unique, strictly increasing number called a *revision*.

The Perforce operation to commit a change is called a *submit*. In most cases, users will submit all changes in their current pending change. As in Subversion, each submitted change is assigned a unique, strictly increasing number called a *change*.

A change contains a description and, optionally, a list of jobs that the change fixes. Jobs are Perforce's bug tracking system and usually mirror an external bug tracking system via a plugin.

Subversion's revision and Perforce's change both identify not just a particular change but also the state of all files in the repository at a given time.

Labels

Labels in Subversion are called *tags*. They are represented as a file path, and by convention only, stored in a subdirectory called "tags".

Perforce stores labels as metadata on the server. There are three kinds of labels:

- *Static* labels resemble Subversion's tags in flexibility
- *Automatic* labels are aliases for changes and fixes
- *Unloaded* labels act like static labels but store their data in a single file

To access files at a label, a Perforce user syncs "files@labelname".

Identifying file revisions

Subversion identifies file revisions only through its global revision number, or HEAD. A single file might contain changes only in revisions 1, 15, 31, and 73.

Files in Subversion are only identified through their local path. Recursion can be defined through a specified depth.

Perforce users specify a file in three ways:

1. Depot syntax—an absolute path including the depot root: //depot/dir/file
2. Client syntax—an absolute path including the client root: //client/dir/file
3. Local syntax—a relative or absolute path of the operating system

Perforce has several ways to identify a particular file revision. A concept similar to a Subversion revision is a *change*, an integer number that is unique to the entire server. Each individual file has its own revision number that starts with 1 when the file is added to the repository. Additionally, a label may also be used to identify a file revision. The following table provides examples. All specifiers in the table could refer to the same file revision.

Identifier	Symbol Used	Examples
Change	@	README.txt@1775
Revision	#	README.txt#14
Label	@	README.txt@rel2.2
Head Revision	#head	README.txt#head

Branching and merging

Subversion and Perforce have a similar architecture for branching: branched files are represented in the file system as copied files, with additional metadata that links the branched files to their source.

Where both products differ is in their implementation.

Subversion by convention stores its branches in a directory called "branches" parallel to the "trunk" directory that contains the mainline. Subversion does not keep track of the relationships among branches; it's up to you to do so, with a naming scheme or with external documentation.

Perforce has no fixed naming scheme, although there is a Perforce Directory Standard that suggests certain naming conventions. Branches are typically created with the help of a *branch spec*, which is a document stored on the server that defines the source and target paths and allows customizations such as exclusions and renames.

The real difference between Subversion and Perforce is the way subsequent merges are treated. Perforce uses a separate database table to keep track of every merge and the choice a user made when resolving a conflict on the server. This feature allows Perforce to make an accurate choice of which file revision still requires merging and the common base of a merge, thus minimizing any merge conflicts, even if the merge across branches is only indirectly related.

Streams

Subversion has no concept of a relationship between branches beyond a naming convention. In contrast, Perforce streams have several benefits to the user:

- Streams define the purpose of each branch: mainline, development, task, or release. Streams follow the mainline model—all changes flow toward the mainline, similar to a Subversion trunk.
- Each stream (except the mainline, which is the root) has a parent stream that defines a clear hierarchy along which changes flow.
- The stream graph identifies changes that still need to be propagated.
- Client workspaces are locked to a stream, eliminating the need to set up the view manually. Client workspaces can be switched from one related stream to another; only the files that differ between these streams will be updated.

Streams replace the branch spec with a stream spec that defines the stream's relationship with other streams as well as its location in the depot and its view.

Access permissions

Subversion sets its access permissions through an Apache module at the repository or directory level with either full or no access.

Perforce has a more fine-grained approach. Access permissions are stored in the Perforce server's protections table and define list, read, open, and write access. Access permissions are typically defined for a group of users and can be restricted to an individual

file, although they are usually defined for a stream or a branch.

Without list access, a user cannot see any files under a specified path, rendering this part of the depot invisible.[1]

ADAPTING YOUR DEVELOPMENT PROCESSES

Choosing a client

Subversion itself does not provide an official GUI, but TortoiseSVN, which adds Subversion commands to Windows Explorer and is maintained by the community, is a popular choice. There are also add-ons for IDEs such as Visual Studio and Eclipse.

Perforce supplies an official GUI client called P4V that is available on Windows, Mac OS X, Linux, and Solaris, as well as official Eclipse and Visual Studio clients. Additionally, many third-party integrations and clients exist.

Both Subversion and Perforce provide rich command-line interfaces ("svn" and "p4", respectively) that offer access to all functions.

Setting up the environment

Subversion stores all its connection information in its hidden ".svn" file. This directory is created when a user checks out files from a Subversion server the first time, and it contains the connection parameters needed to commit changes back.

Perforce users specify the connection parameters directly in any tool such as P4V or an IDE. From the command line, the connection information is typically defined in the environment or, on Windows and Mac OS X, in the registry. The important variables are:

P4PORT=my-perforce-server:1666

P4USER=my-user-name

P4CLIENT=my-client-workspace-name

If the client machine has only one client workspace, then these three variables are all that a user needs.

If you have more than one workspace on your workstation, you can define the environment variable P4CONFIG to point to a file usually named ".p4" or "p4config.txt". Place a text file with this name containing the connection parameters into your workspace root. This technique is closest to how Subversion works.

1 See more details in the Perforce System Administrator's Guide: http://www.perforce.com/perforce/doc.current/manuals/p4sag/chapter.protections.html

Getting help

If you are stuck with a command, Perforce provides a comprehensive help system, invoked via "p4 help". The help pages do require a working connection.

Creating and using a client workspace

After connecting to the Perforce server, the first step, before you can upload and submit any files, is to create a client workspace.

A client workspace has a name, a root directory, a view, and a set of options.

Client workspace name

The client workspace name is unique on the whole Perforce Server. By default, the name matches the hostname of the user's workstation because each workspace is locked to a host.

Most organizations have an established naming convention for client workspaces (sometimes enforced through triggers). A useful convention could be user.host.project (e.g., myname.macbook.p4python).

Set your P4CLIENT value to whatever name you choose.

Creating a client

To create a client, run the command

```
p4 client
```

from the command line *after* you set P4CLIENT. An editor will open up that presents the client workspace information for you to edit. Make your changes (see 3.4.4 below) and save and close the editor, and the Perforce Server will create or update the client workspace for you.

Alternatively, create the client workspace through a GUI tool such as P4V.

Root

The root of your client workspace is the directory under which you will place all files under Perforce control. This is similar to the root of your Subversion working copy.

View

The view maps files from the Perforce Server to your local drive (and the other way around for new files). Usually, the layout of the files on the server and your client workspace match, but this does not always have to be the case.

In general, a workspace view maps a depot path to a client workspace path. If you created a fresh Perforce Server, your first workspace path will most likely look like this

```
//depot/... //workspace_name/...
```

Here "workspace_name" is replaced with your client workspace name. The three dots match all files and directories below the specified path.

Because a Perforce Server often stores many projects, this wide open view that maps every single file on the server can cause you to download many more files than you want. To avoid this problem, you can change your mapping to something like the following, depending on your depot layout

```
//depot/project/branch/...  //workspace_name/...
```

where "project" is your project name and "branch" is the current branch you are working on.

The client workspace mapping is somewhat similar to specifying a subdirectory in the URL when checking out a working copy from a Subversion server.

Options

There are several options for configuring a client workspace. Coming from Subversion, the most important option is noallwrite. More details on workspace options can be found in the *P4 User's Guide* (see Learn More).

Working with files

Subversion and Perforce work in similar ways with files. You retrieve a local copy of the files, then edit, build, and test in this environment, and ultimately commit your changes back to the server.

In Subversion, all files are always writable in the workspace. You edit the files you need and when you commit your changes, Subversion determines which files have changed and offers to include them in the commit.

In Perforce, you have a choice. By default, all the files in your workspace are read-only to begin with. You explicitly check out files using the command "p4 edit". This has several effects:

- The file is made writable in your workspace.
- The file is marked for edit on the Perforce Server; this step makes it clear to your team members that you are editing the file.
- If the file has the file type "exclusive checkout", no other user can edit the file until you have committed or reverted. This is useful for files that cannot be merged (e.g., most binary files).
- When you submit your changes, Perforce does not have to search your entire workspace for changed files. Instead, it simply looks up changes in the database.

If you prefer the Subversion mode of working, you can set your workspace to *allwrite* by modifying its options. With this setting, all files synced to your workspace are now writable by default and you can modify them without checking them out explicitly first. If you switch an existing workspace to *allwrite*, keep in mind that you need to resync all previously synced files again or use OS methods to make them writable.

Perforce offers two commands to identify which files you have changed: "p4 status" shows you which files have been edited, added, or deleted, and "p4 reconcile" adds the changed files to the pending changelist ("p4 status" is an alias for "p4 reconcile n"). The typical workflow is then

```
p4 sync            # update your workspace
vi hello.c         # change your file locally
p4 reconcile       # update the pending changelist
p4 submit          # submit the changes
```

"p4 reconcile" can also discover renamed and moved files even if their content has slightly changed. This is particularly important when refactoring Java code because the class name inside the file and the file name itself are linked.

To prevent "p4 reconcile" from adding files you do not want to submit (such as generated object or class files), you can add these files to an ignore list in a file specified by the environment variable P4IGNORE (e.g., "p4ignore").

Before switching your workspace options, keep in mind that most IDEs and editors integrated with Perforce will check out the file automatically for you when you start typing.

Why Explicit Checkouts?

One reason for using explicit checkouts is that it removes the need to scan files for content changes. While with smaller projects calculating hashes for each file is fairly cheap, many Perforce users have millions of files in a workspace and/or have files in excess of 100MB. Calculating all the hashes in those cases is extremely time consuming. Explicit checkouts let Perforce know exactly which files it needs to work with. This behavior is one of the reasons Perforce is so popular in industries that use large files like game studios, movie producers, and hardware.

Another benefit: Explicit checkouts provide a form of asynchronous communication that lets you know which files your peers are working on. It can let you know that you may want to avoid working in a certain area to prevent a needless conflict, or it can alert you to the fact that a new developer on the team has wandered into code that perhaps doesn't need to be edited.

Explicit checkout also plays nicely with the Perforce concept of pending changelists. Pending changelists are buckets that you can put your open files into to organize your work. Branches are great, but sometimes it is nice to be able to organize your work into multiple named changes before actually submitting to the server. With the Perforce model of potentially mapping multiple branches or multiple projects into one workspace, pending changelists make it easy to keep separate changes organized.

Changelists

In Subversion, you can group your changes into local changelists and commit these individually, although in most cases you probably will have simply committed all changed files in your working copy together.

In Perforce, a changelist is a global object on the server that has several states described next.

Default pending changelist

Each workspace always has a default changelist associated with it; you do not need to create it. When you check out a file without specifying an explicit change, it is automatically in the associated default changelist.

Default changelists are not numbered, but they can be accessed with the name "default".

Numbered pending changelist

You can create a numbered pending changelist by creating a *change* object. The change object needs a description and can have zero or one or more open files associated with it. The server sets the number when you create the object.

The Perforce Server uses a single atomic counter for all changes: submitted, pending, and shelved. When a new numbered pending change is created, it receives the next available number, just like a submitted change.

Perforce asserts that submitted changes are ordered by time—a higher change number implies the change was submitted later. It is likely that other users will have submitted changes after you created your pending change and before you submitted this change, so your pending change will probably be renumbered when it is submitted. This is expected behavior. The pending change number is simply there for convenience to be able to update the pending change.

You can have an arbitrary number of pending changes associated with your workspace and you can move open files between numbered pending changes and from and to the default changelist. Note that each file can only be opened once in a pending change for your workspace because the file content is still stored on your local disk. If you need to keep several states of a file in your workspace at the same time, you need to shelve the file.

Shelved changes

Numbered pending changes can be shelved, that is, the content of the open files can be saved on the Perforce Server for safekeeping and sharing. This feature can be used to:

- Back up a change if it is not ready for a submit.
- Stash changes temporarily to work on something else.
- Share changes for review or transport between workspaces.

Shelved changes are visible to other team members and can be unshelved in different workspaces, by different users, and even different branches or streams. Shelves are temporary—they need to be deleted when the change they are associated with is submitted.

Submitted changes

When submitting a default pending change, you need to provide a description. For numbered pending changes, the description already stored will be suggested. Once a change is submitted, its contents cannot be modified further. It is possible, however, to update its description and any associated fixes for jobs.

MIGRATING YOUR DATA

All migrations from one version control system to another offer two basic choices:

1. Keep the old repository in read-only mode for reference and only import the head revision into the new tool. This is certainly the fastest and easiest way to migrate your data, but it is not applicable if you are migrating in the middle of an existing project or if you have long-running releases you need to support.
2. Migrate some or all of the history into the new tool. There will always be an impedance mismatch between different source control systems because storage and usage are often very different. Subversion and Perforce are sufficiently similar to make a full migration possible and the outcome acceptable.

Perforce provides a conversion tool for your Subversion repository, which you can find here:
ftp://ftp.perforce.com/perforce/tools/p4convert-svn

This tool supports two modes: Full import into a new Perforce Server and incremental import into an existing Perforce Server. Please refer to the documentation for more information.

For assistance with your migration requirements, please contact Perforce Support or Professional Services.

LEARN MORE

Perforce has a whole host of documentation and guides available on its Website. The following documents will help you to use Perforce more effectively.

Perforce Resources:
http://www.perforce.com/resources

Introduction to Perforce:
http://www.perforce.com/perforce/doc.current/manuals/intro/index.html

P4 User's Guide:
http://www.perforce.com/perforce/doc.current/manuals/p4guide/index.html

Perforce System Administrator's Guide:
http://www.perforce.com/perforce/doc.current/manuals/p4sag/index.html

Perforce Directory Standard:
http://info.perforce.com/PDS.html

Agile Flow of Change:
http://info.perforce.com/whitepaper-agile-flow-change.html

Contact Perforce:
http://www.perforce.com/contact

perforce.com

North America
Perforce Software Inc.
2320 Blanding Ave
Alameda, CA 94501
USA
Phone: +1 510.864.7400
info@perforce.com

Europe
Perforce Software UK Ltd.
West Forest Gate
Wellington Road
Wokingham
Berkshire RG40 2AT
UK
Phone: +44 (0) 845 345 0116
uk@perforce.com

Australia
Perforce Software Pty. Ltd.
Suite 3, Level 10
221 Miller Street
North Sydney
NSW 2060
AUSTRALIA
Phone: +61 (0)2 8912-4600
au@perforce.com