

Comparison:

Perforce and Subversion

Perforce 2011.1 and Subversion 1.7

This document compares Perforce (version 2011.1) with Subversion (version 1.7). Read this comparison to:

- Understand Perforce and Subversion's major feature differences
- See head-to-head metrics for operations like branching, merging, check-ins, and checkouts
- Get a general comparison of the effects of scaling on both systems

Table of Contents

Executive Summary	1
Branching, Merging, and Release Management	3
Subversion	3
Basic Branching and Merging	3
Branching Model, Dependencies, and Guidance	3
Perforce	4
Basic Branching and Merging	4
Branching Model, Dependencies, and Guidance	4
Workspaces and Collaboration	5
Subversion	5
Perforce	6
Visual Tools	6
Subversion	6
Perforce	6
Defect Tracking	7
Subversion	7
Perforce	7
Integration with Related Tools	7
Subversion	7
Perforce	7
Distributed Development	7
Subversion	7
Perforce	7
Scalability and Performance	8
Subversion	8
Perforce	8
Digital Asset Management and Storage	8
Subversion	8
Perforce	9
Support and Maintenance	9
Subversion	9
Perforce	9
Administration and Security	9
Subversion	9
Perforce	9
Auditing, Compliance, and Reporting	9
Subversion	9
Perforce	9
Benchmarks	10
Test Platform	10
Test Data	10
Test Results	10
Appendix A: Detailed Branching and Merging Analysis	12
Results Summary	12
Intended Workflow	12
Subversion's Problems with Base Selection and Merge History	12
Learn More	14
Evaluating Perforce	14
Scheduling a Demo of Perforce	14
Migrating to Perforce	14

Executive Summary

The choice of software version management system has a profound impact on those involved in digital asset management, from software developers to artists to managers. An effective software version management system is one that:

- Provides a full history of the evolution of digital assets
- Enables parallel development and concurrent team activity
- Helps the entire team work more efficiently

- Meets modern development and scalability challenges
- Is fast, flexible, and reliable

This document shows that Perforce has maintained its core strengths in productivity, collaboration, branching and merging, scalability, and administration, while offering innovative solutions for new software version management challenges. Subversion, on the other hand, has focused its efforts on addressing shortcomings in basic functionality. Although the total cost of ownership or return on investment is outside the scope of this document, it becomes clear that Perforce is the stronger solution for modern software version management.

Overview

Attribute	Subversion	Perforce
Branching, Merging, and Release Management	Subversion offers branching, and provides basic merge history tracking in the destination file's attributes. Subversion's merge algorithms do not operate well in all scenarios. Branching framework and release model are designated by convention, with repository-wide hooks available for guidance. Dependencies are managed with externals.	Perforce automatically tracks the history of all branch operations with its advanced and mature merge tracking mechanisms. Streams provide best-practices branching framework. Granular triggers and broker are available for guidance. Dependencies are managed via flexible stream, branch, and workspace views, or via remote depots.
Workspaces and Collaboration	Important metadata stored in the workspace only; corruption of the workspace is difficult to recover from. No shelving facility. Changelists are a client-only concept; no way to see what others are working on or prevent concurrent editing of files that cannot be merged.	Workspace definition stored on the server and can be recreated. Shelving offers simple task switching, task hand off, and code review. Changelists are tracked on the server. The open-edit-submit workflow provides transparency into current work and also helps prevent concurrent work on files that cannot be merged.
Visual Tools	Community supported tools available; usually specific to a single platform or environment.	Full featured multiplatform visual client. Visualization tools include Stream Graph and Merge Quest.
Defect Tracking	Not included with Subversion; users need to use a separate defect tracking solution.	Perforce provides a basic defect tracking system called jobs, and integrates with leading third-party defect tracking systems such as HP Quality Center and Atlassian JIRA.
Integration with Related Tools	Integrations are available via the open source community.	Perforce integrates with leading IDEs, such as Eclipse and Visual Studio, and other tools in many categories. Integration tools include the Defect Tracking Gateway and several fully supported APIs.

Overview (continued)

Attribute	Subversion	Perforce
Distributed Development	Remote slave servers can be synchronized from a centralized master server. Synchronization mechanism requires the coding of event-based scripts.	P4Sandbox offers connection independent versioning and private local branching. The Perforce Proxy offers a file caching solution for remote users, with minimal administrative overhead. Replica servers provide a full copy of server data at remote locations for read-only operations.
Scalability and Performance	Subversion depends heavily on CPU-intensive operations that do not scale well when workloads increase. Subversion usually runs as an Apache module. Performance benchmarks indicate that Subversion does not scale well in demanding environments.	Perforce's architecture is proven and continues to scale well up to and beyond many terabytes and tens of millions of versioned files. Architected for speed, Perforce scales linearly and is currently deployed in environments with 10,000+ users and heavy automation. Proxy, Broker, replica servers, and P4Sandbox provide a powerful and flexible deployment architecture. Perforce supports aggressive continuous integration and automation with no impact on the central server.
Digital Asset Management and Storage	Inefficient at storing large digital (binary) assets. Shared storage for metadata and versioned files.	Effectively stores any kind of digital asset. Storage for metadata is separate from versioned file storage, which can use different storage systems for different products or file types.
Support and Maintenance	Available at additional cost from third parties. Older releases unsupported after two major increments.	World-class technical support, training, and professional services provided by Perforce. Products supported for a long time with a defined sunset policy.
Administration and Security	Several deployment options available, often requiring an Apache web server. Security often done at the repository or directory level. Inconsistent tools across platforms. Upgrades often must be done simultaneously for server and client, and sometimes require making new workspaces.	Simple, consistent deployment. Granular access control. Excellent cross-platform support. Simple upgrade procedures with good interoperability across versions.
Auditing, Compliance, and Reporting	Utilizes Apache logging for auditing. Reporting done via command-line scripts.	Built-in audit log. Reporting can be done via command line, scripting APIs, custom SQL queries, or a report engine.

Branching, Merging, and Release Management

Subversion

Basic Branching and Merging

Subversion offers codeline branching using a simple copy command to create a branch, and keeps a basic history of the operations attached to each file individually in file-keyed attribute lists. Subversion merges are slow and historically have been problematic and prone to merge errors. Bidirectional and divergent merges are supported with newer servers and clients, but with extra merge steps involved.¹ Other complex merging cases, particularly involving renames and tree conflicts, can be difficult to resolve.² Merging renamed files may overwrite newer revisions on the target branch or cause a tree conflict.³

As one simple example of a case where Subversion merging does not work properly, consider the following scenario illustrated in Figure 1.

In this scenario, a file was branched immediately after creation, several edits were made on both branches, and

some of the changes from the child branch were merged back. In Step 7, we try to merge the latest changes from the parent to the child. Subversion selects the first parent revision as the merge base, which leads to an unnecessary merge conflict.

This scenario, and a detailed branching and merging analysis in Appendix A, show that Subversion cannot handle more complex merge scenarios well, and offers less granular resolution in many situations.

Branching Model, Dependencies, and Guidance

Development and release management models are built by convention, usually using a known directory structure for different branch types. Hooks can be used to guide these models, although Subversion hooks affect the entire repository and must be coded appropriately.

Dependencies between modules or projects can be managed using Subversion externals, which require careful configuration and usage, and are disjoint in some sense from the rest of the workspace. Many Subversion operations, for example, will not automatically recurse into external directories.

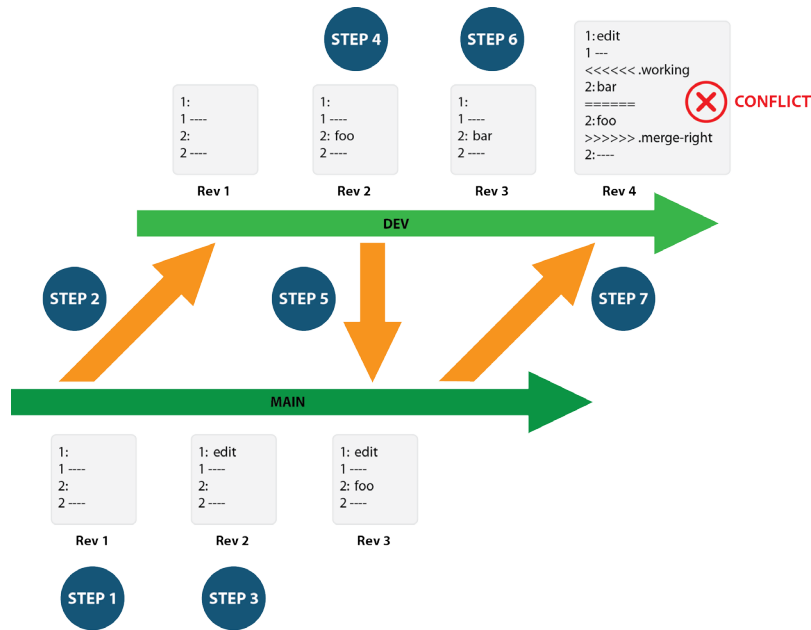


Figure 1: A simple merge scenario in which Subversion selects the wrong merge base

1 For example, after promoting a feature branch into the trunk with a reintegrate merge, the feature branch cannot be used unless the trunk is merged in and that change ignored. Subversion does not properly track merge history in these cases.
2 Subversion treats a rename as a combination of a copy and delete.
3 <http://svnbook.red-bean.com/en/1.6/svn.branchmerge.advanced.html#svn.branchmerge.advanced.moves>

Multiple products can be housed in a single repository if the directory structure is planned in advance, in which case server storage is shared amongst the products. Alternatively, each product can use its own independent repository and file storage, but at the cost of communication and code sharing between products.

Perforce

Basic Branching and Merging

The Perforce Inter-File Branching model is powerful and flexible, capable of branching thousands of files rapidly while retaining a complete branch and merge history (including the method of resolving merges). Instead of manually tracking all changes across branches, users can rely upon Perforce to merge file changes across multiple branches automatically and with fewer conflicts to resolve. This enables a variety of development scenarios, such as client-specific versions, experimental branches, personal or task branches, and the classic release branching patterns. A built-in graphical tool, Revision Graph, can be used to display the detailed branching history of each file for easy visualization of code propagation (see Figure 2).

Perforce inherently understands the ancestral relationship files share between branches. Once a change has been integrated from an originating branch into a target branch, there is no ambiguity that the change exists in both branches. Therefore, on subsequent integrations from the target back to the originating branch, users will only be prompted to integrate changes that have occurred directly in the target branch. This kind of integration pattern is referred to as bidirectional or ladder merging.

Perforce's merge engine provides both robustness and flexibility. The most complex merge scenarios are supported, including refactoring, indirect merges, and

handling non-content changes. A detailed branching and merging analysis is provided in Appendix A.

In the simple merge scenario described in Figure 1, Perforce correctly selects the merge base on the target branch, giving a clean merge with no conflicts (see Figure 3).

Branching Model, Dependencies, and Guidance

Perforce Streams provide a lightweight but powerful branching model. Using streams, a product architect can define the relationship between streams, between the modules that compose a product, and also direct the flow of change (merges) between streams. This information is used to simplify and automate many routine operations for users.

When using either streams or regular branching, Perforce offers several tools to guide and enforce policy. Triggers and access control are granular, while the Perforce Broker provides command filtering.

Dependencies between modules or projects are managed via flexible stream, branch, or workspace views. A logical working view of a product can thus be easily assembled from several components developed independently. Remote depots support code drops between independent repositories.

Multiple products can be housed in a single depot or each in their own depot, depending on access control and usage policies. Housing products in multiple depots allows for distribution of physical file storage on the server, an important consideration for projects with large data volumes. The use of multiple depots is effectively transparent to the end user.

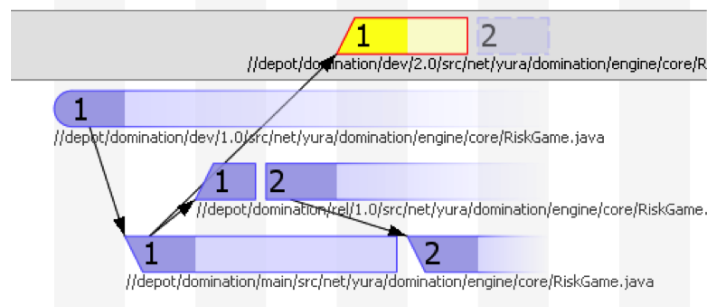


Figure 2: Perforce Revision Graph

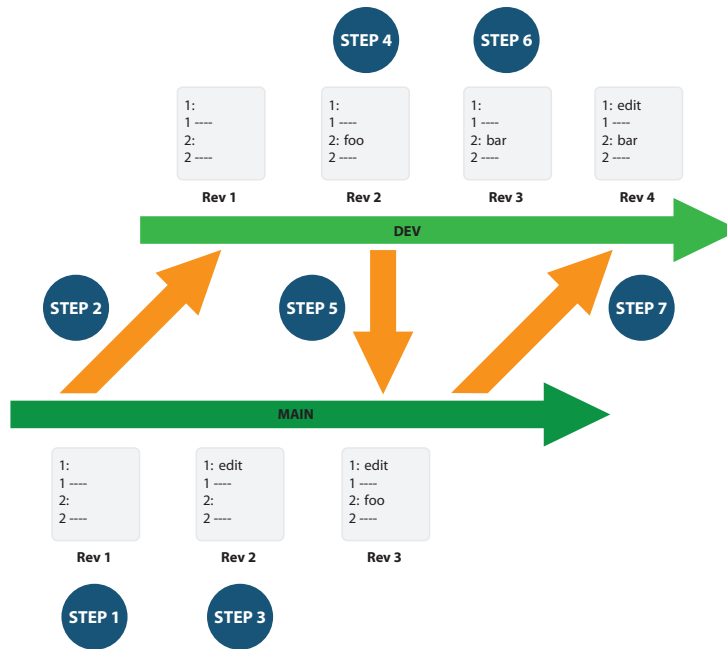


Figure 3: Perforce selects the correct merge base

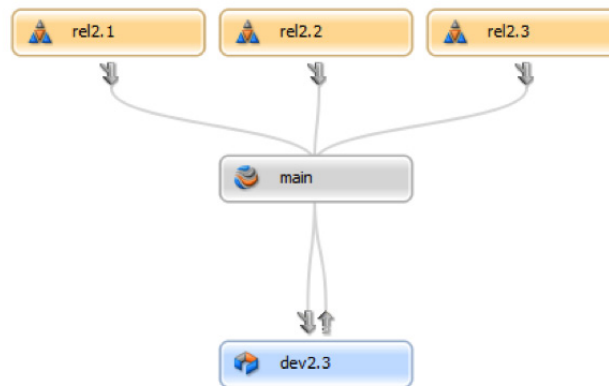


Figure 4: Perforce Stream Graph

Workspaces and Collaboration

Subversion

Subversion stores important metadata and state information in the workspace. Recovery from workspace corruption is often difficult, and requires creating a new workspace. Prior to Subversion 1.7, each directory in the workspace had a `.svn` folder; accidentally deleting a subdirectory in the workspace would result in that subdirectory being unknown to the workspace, even while the parent directory was intact. In Subversion

1.7 all metadata is stored in the root of the workspace, but the new metadata storage format is susceptible to corruption from concurrent access.

Subversion offers no shelving facility; storing and sharing work in progress requires using a branch.

Subversion's changelists are client-only concepts. There is no way in Subversion to know what others are working on or prevent concurrent checkouts of files that are difficult to merge, as Subversion does not offer an open-edit-submit workflow.⁴

⁴ Subversion does offer ad-hoc file locking, which requires the locking user to issue a special command. This lock prevents another user from committing the locked file, but does not prevent them from working on it in the first place unless a special file property is set.

Subversion workspaces usually offer a simple, monolithic view of the repository starting at the point of checkout. Sparse checkouts allow pruning or inclusion of some parts of the repository, but could require many steps in order to assemble the proper set of data in the workspace.

Perforce

All important metadata in Perforce is stored on the server. Recreating a lost workspace (apart from actual unsaved file content) is simple.

Perforce shelves provide a simple way to store work in progress, switch tasks, and share work in progress for code review.

Perforce changelists are server-side metadata, allowing all users to see what files are checked out. Perforce can also prevent concurrent checkouts of files that are difficult to merge, which helps prevent wasted time and effort. Perforce supports both an open-edit-submit workflow and an edit-open-submit workflow.

Perforce workspace views provide a layer of abstraction between the server's view of the data and the view on a workstation. Using a workspace view, a user can pick and choose different pieces of a repository to view. The view is captured and versioned as important metadata.

Visual Tools

Subversion

Community supplied tools are available for most platforms, but there tends to be a distinction between Windows and other platforms.

Perforce

Consistent visual tools are available for all major platforms. Innovative visualizations like Stream Graph, Time-Lapse View, Merge Quest, and HTML5 applets, greatly enhance user productivity (see Figures 4-6).

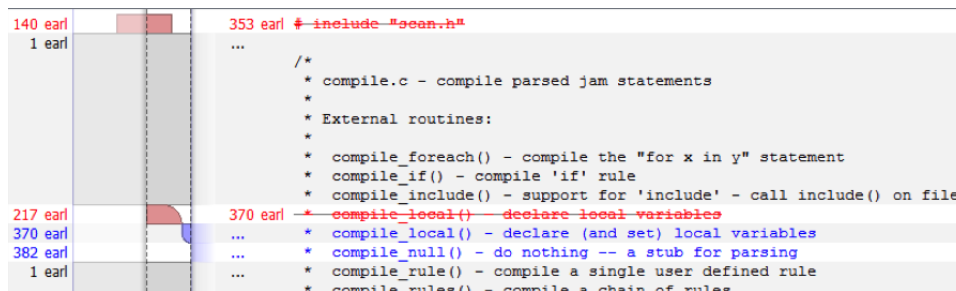


Figure 5: Perforce Time Lapse View

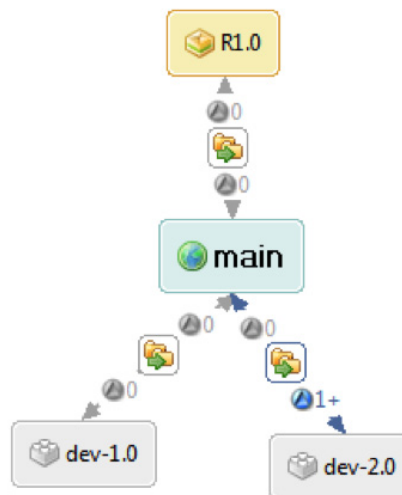


Figure 6: Perforce Merge Quest

Defect Tracking

Subversion

Subversion does not include defect tracking, but a limited number of third-party integrations are available. Defect tracking integration typically relies on fragile commit message parsing.

Perforce

Perforce provides a basic defect tracking system called *jobs*. A job typically represents an enhancement request or a bug to be fixed. Job definitions are customizable to support many workflows. Jobs can also work with several third-party defect tracking systems, such as HP Quality Center and Atlassian JIRA. Data entered into a Perforce job is automatically replicated in the defect tracking tool, and vice versa. Perforce can tell you whether or not a bug fix is present in any particular codeline, regardless of the codeline in which the bug fix originated.

Using Perforce jobs, users can easily link submitted changes to defect records without relying on commit messages.

Integration with Related Tools

Subversion

The open source community supports different Subversion GUIs, as well as plug-ins for working within popular IDEs.

Perforce

Perforce has a mature multiplatform GUI, and there are plug-ins for most of the popular IDEs. Integration tools include the Perforce Defect Tracking Gateway and several fully-supported APIs. Perforce also has integrations for popular applications including:

- IDEs
- Web and graphical tools
- Software build tools
- Microsoft Office
- Merge and diff tools

Distributed Development

Subversion

With Subversion, remote developers work directly against local slave servers. All slaves are configured to synchronize their data stores from a centralized master. Event-based scripts triggered by updates in the master notify each registered remote slave to synchronize its archive. Scripts need to be created and then deployed to both slave and master servers. Each slave server must be configured to either deny all write access or proxy write requests through to the master server. If local commits are done directly on a slave server, there is no easy correction.

Switching workspaces between master and slave servers requires that each user run a special command, making it difficult to automate.

Third-party commercial tools are necessary to achieve additional levels of performance at remote locations at an additional cost.

Perforce

Distributed development with Perforce does not require any additional process overhead or additional cost. Several tools support distributed development with Perforce and are typically transparent to the end user.

P4Sandbox supports connection independent versioning, private local branching, and fast local operations, in addition to its other workflow benefits.

Perforce proxies at remote locations support Perforce's distributed architecture (see Figure 7). The Perforce Proxy caches and serves files to users at remote locations, thereby reducing traffic across slower WAN links. All users, local or remote, connect to the same central depot, and look at the same project files. The Perforce Proxy requires minimal administrative attention.

Replicated Perforce servers provide completely local read-only operations for remote users. As a large percentage of Perforce operations are read-only, using a local replica offers a significant performance benefit.

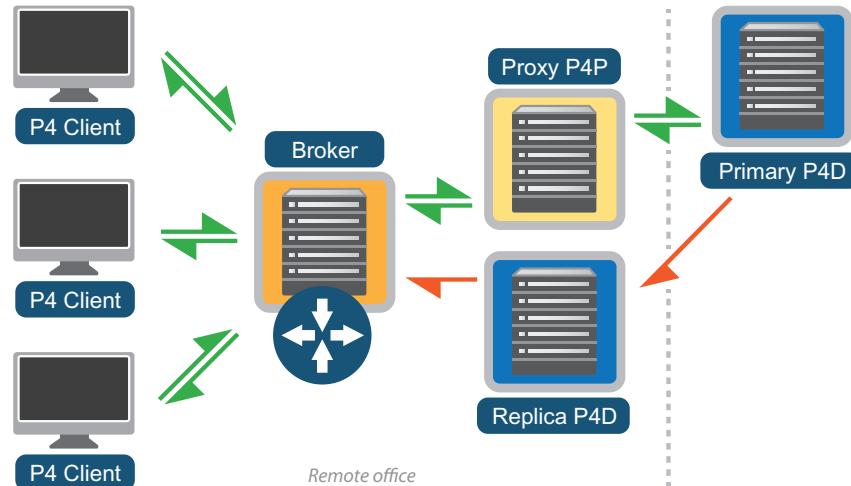


Figure 7: Perforce's built-in distributed architecture

Scalability and Performance

Subversion

With Subversion, the binary delta differencing scheme used to version both binary and text files consumes a large amount of CPU time on the client and server. This is especially problematic when the files are large and there are many concurrent users competing for the same resources. Perforce easily outperforms Subversion in all equivalent versioning operations during comparative benchmarking tests.

On a qualitative note, some visual tools provided by popular Subversion GUIs can take an excessively long time to present information. On a repository with 13,000 revisions, for instance, the TortoiseSVN revision graph takes over three minutes to load a revision graph for a single file with only three revisions.

Perforce

Perforce is architected first and foremost for speed. The reverse delta RCS format for text and GZip compression on binary files means users do not wait for check-ins and other standard operations. Perforce scales linearly, so there is no performance penalty imposed by the number of revisions or the size of any given file(s). The Perforce Server has been deployed successfully in environments with several thousand users, terabytes of versioned content, and millions of revisions. Perforce's deployment architecture now includes proxies, brokers, replicas, and P4Sandbox; these tools can be deployed in many combinations to satisfy demanding environments. Replicated servers are particularly useful for supporting

automated processes such as aggressive continuous integration; the performance burden of such processes is shifted entirely away from the production server.

Digital Asset Management and Storage

A modern version control system must be able to effectively store and manage any kind of digital asset, from source code to graphics to chip designs. Otherwise a second system will be necessary for storing these non-text assets, which requires extra overhead, administration, and process.

Subversion

Subversion cannot effectively store and manage large digital assets. Its binary differencing algorithm imposes a severe CPU burden on the client workstation, with unacceptable performance degradation. Additionally, a Subversion workspace maintains a pristine copy of all file content in addition to the actual working copies. With large data volumes, this could impose a burden on the typical workstation.

On the server, storage for a single repository is on a single file system for the metadata and all versioned files, giving the administrators little flexibility to deal with large data volumes. Subversion slave servers can be used to offload file transfer work, but switching workspaces from the master server to the slave requires a manual step by each user.

Perforce

Perforce is used by customers to store and manage terabytes of digital assets of all types. Perforce's efficient workspace management imposes little overhead on clients. On the server, storage can be partitioned between the metadata and several depots for versioned files, giving administrators great flexibility in planning storage solutions. If more granularity is necessary, particular file types can be managed in separate storage, and unnecessary data can be archived to offline storage. Perforce also offers several easy-to-deploy tools that are transparent to both end users and the process for offloading file transfer work from the primary server to support continuous builds, automation, and remote users.

Support and Maintenance

Subversion

Subversion offers full accessibility to the source code, but the administrator must invest the time and effort to become familiar with the codebase. Commercial third-party support, training, and services are also available. Subversion stops supporting older releases after two major increments. For instance, when Subversion 1.6 was released, support for Subversion 1.4 was discontinued.

Perforce

Expert and responsive technical support is a hallmark of Perforce and full technical support is included during an evaluation. Perforce believes that when you buy a software version management system, you are not only putting your faith in the software, but also in the technical support you expect to receive. Because the quality of a technical support organization is better experienced than described, Perforce encourages prospective customers to judge for themselves during a typical 45-day trial evaluation. Perforce also offers a full range of training and professional services. Perforce provides technical support for past releases for an extended duration, and has a defined policy that provides 12 months' minimum notice before an older product is retired.

Administration and Security

Subversion

Subversion requires administrators to make several decisions before implementation. For example, administrators must decide whether to configure a web server or use a standalone Subversion server; which authentication and access control mechanisms to use; and whether to use a file-based system or a database for storing versioned files. Subversion access control is not granular, and is often done at the repository level, although directory-level access control is possible.

Subversion imposes extra administrative overhead. Its toolset is inconsistent across platforms, forcing users in multi-platform environments to adapt accordingly. More importantly, the Subversion server and client software must be upgraded simultaneously and often, with manual upgrade steps required for clients. In some cases, Subversion recommends making new workspaces rather than upgrading existing ones.

Perforce

Perforce functions the same regardless of platform, and provides a built-in security mechanism with many levels of access that can be applied to any subset of the archived files—from the entire depot to the individual file. Versioned files are stored the same on all platforms.

Perforce imposes minimal administrative overhead. Perforce deployment is simple and consistent. Working in a multiplatform environment poses no problem, and Perforce has excellent support for mixed server and client versions. Upgrade procedures are simple and fast.

Auditing, Compliance, and Reporting

Subversion

Subversion relies on Apache facilities for audit logging. Reporting is most often done using the Subversion command-line client.

Perforce

Perforce has a built-in audit log. Reporting is possible via the Perforce command line, scripting APIs, SQL queries, or report engines. The latter two options make sophisticated data mining possible, and are available because Perforce's database can be replicated into a relational database.

Benchmarks

Perforce easily outperforms Subversion in all equivalent operations during comparative benchmarking tests. Notably, Subversion performance degrades considerably as the number and size of files increases. The test results are detailed below.

Test Platform

The combined client/server machine on which the tests were run had the following specifications:

- Amazon EC2 high memory quadruple extra large instance
- Linux x86_64 64-bit Red Hat Enterprise Linux
- 26 ECUs, 8 cores, 68GB RAM
- P4D/LINUX26X86_64/2011.1.BETA/347706
- svn, version 1.7.0 (r1176462):
CollabNetSubversionEdge-2.1.0_linux-x86_64

Test Data

The test data for the small run consisted of the Apache web server and Tomcat application server source trees, totaling 46MB and 4,480 files. The medium dataset was four copies of the small dataset. The large dataset comprised 207 large binary files, totaling 632MB in size.

Test Results

Performance results for the Perforce and Subversion comparative benchmarks are presented in the following tables. Response times were measured in seconds.

Table 1: Results for equivalent versioning operations on a small dataset (all time in seconds).

Operation	Subversion (1.7) – time in seconds	Perforce (2011.1) – time in seconds
Add tree	20.9	34.0
Lightweight branch	0.3	1.1
Delete tree	1.5	1.0
Label/tag tree	0.3	0.3
Merge edits to entire tree	73.1	16.4
Preview merge (no merge necessary)	4.3	0.2
Sync entire workspace	14.0	1.4
Preview sync (no updates necessary)	1.0	0.6
Restore deleted tree	32.0	36.1

Table 2: Results for equivalent versioning operations on a medium dataset (all time in seconds).

Operation	Subversion (1.7) – time in seconds	Perforce (2011.1) – time in seconds
Add tree	79.5	139.7
Lightweight branch	0.7	4.4
Delete tree	4.0	4.0
Label/tag tree	1.0	0.5
Merge edits to entire tree	283.4	62.5
Preview merge (no merge necessary)	11.2	0.4
Sync entire workspace	41.8	5.9
Preview sync (no updates necessary)	1.5	2.5
Restore deleted tree	161.8	160.9

Table 3: Results for equivalent versioning operations on a large (binary) dataset (all time in seconds).

Operation	Subversion (1.7) – time in seconds	Perforce (2011.1) – time in seconds
Add tree	44.2	51.8
Lightweight branch	0.1	0.8
Delete tree	1.5	1.0
Label/tag tree	0.1	<0.1
Merge edits to entire tree	62.5	10.7
Preview merge (no merge necessary)	3.5	<0.1
Sync entire workspace	62.7	9.4
Preview sync (no updates necessary)	1.0	<0.1
Restore deleted tree	63.0	48.9

Appendix A: Detailed Branching and Merging Analysis

This appendix analyzes several common problems with Subversion branching and merging, with specific examples. The analysis is grounded in a set of branching and merging tests that Perforce runs internally as a regression suite for its own product. Perforce successfully passes all of the tests described here.

Results Summary

Out of 141 tests, Subversion failed 20. Six others are not applicable because Subversion does not support as many options during the merge and resolve process.

Intended Workflow

Subversion's merging commands were intended to support a basic branch-merge-promote workflow. A child branch is created for new work. Changes from the parent are periodically merged down to keep the child up to date. When work on the child branch is complete, it is promoted to the parent and retired.

Whenever you begin working outside the bounds of that basic workflow, complexities appear:

- Reusing a child branch requires additional steps, as described in the Subversion online book.⁵
- Working at the file rather than the directory level is much more difficult. For instance, it is difficult to reference a deleted head revision in Subversion, so propagating a file delete needs to happen by merging the entire directory (and possibly dealing with a tree conflict).
- Merge base selection is inefficient when dealing with ladder merges or indirect merge history, resulting in more merge complexity and conflicts.
- Indirect merge history is sometimes ignored. Previously merged or ignored deltas may be reintroduced.
- Subversion's method of reaching branch convergence, the `reintegrate` merge option, is intended for a very specific use case and cannot be applied elsewhere. In general situations Subversion lacks a command that forces convergence during a merge.⁶

Subversion's Problems with Base Selection and Merge History

Several failures were due to poor or inefficient base selection, while others were due to failure to consider prior merge history. Subversion seems to favor picking a merge base on the source branch, even if selecting a base on the target (or an indirectly related) branch would yield a more efficient merge.

In some cases picking a less desirable merge base has little or no effect on the end result. But in some cases, as discussed prior in *Branching, Merging, and Release Management*, picking a less desirable merge base causes merge conflicts unnecessarily. The base should be chosen to reflect, as much as possible, any existing merge credit—revisions already accounted for in the merge target. Generally speaking, picking a merge base as close as possible to the source and target revisions yields less deltas to consider, reducing the chances for conflicts and other problems.

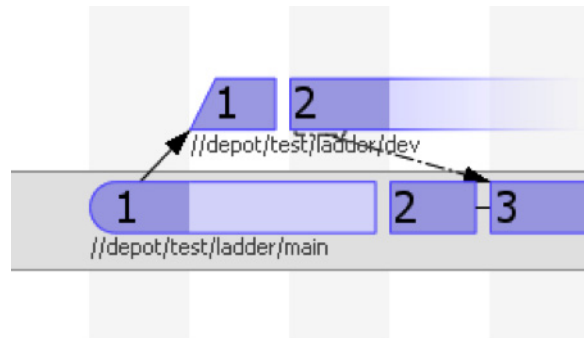
When a merge gives conflicts and other problems without a valid reason, or reintroduces previously considered deltas, users lose confidence in the merge engine. Besides the time and effort to fix the immediate problem, users become more likely to manually verify merges, which can severely impact productivity.

On a related note, Subversion quite easily allows merging between files or directories with no ancestry. Subversion's merge command originally had to assume no information about merge ancestry, as Subversion did not offer merge tracking until the 1.5 release. Although baseless merges are occasionally useful, they should not be available without specific user acknowledgement, in order to prevent accidental merges.

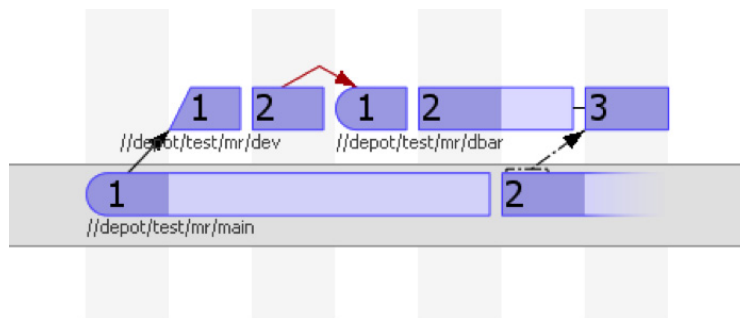
Further examples of improper base selection and merge history handling are detailed next.

⁵ <http://svnbook.red-bean.com/en/1.6/svn.branchmerge.advanced.html#svn.branchmerge.advanced.reintegrateatonce>

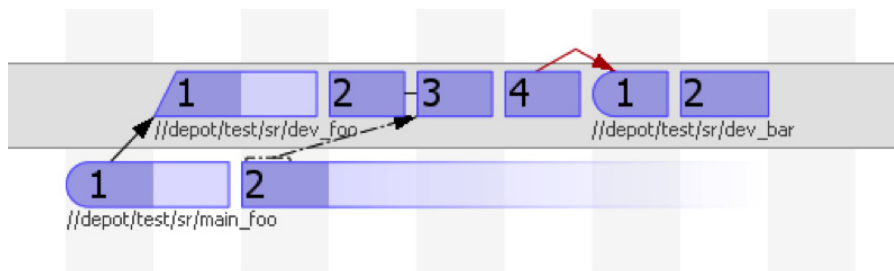
⁶ If there are conflicts during a merge, Subversion allows either the source or target to be accepted in full as the merge result. But oddly enough if there are no conflicts the contents will always be automatically merged.



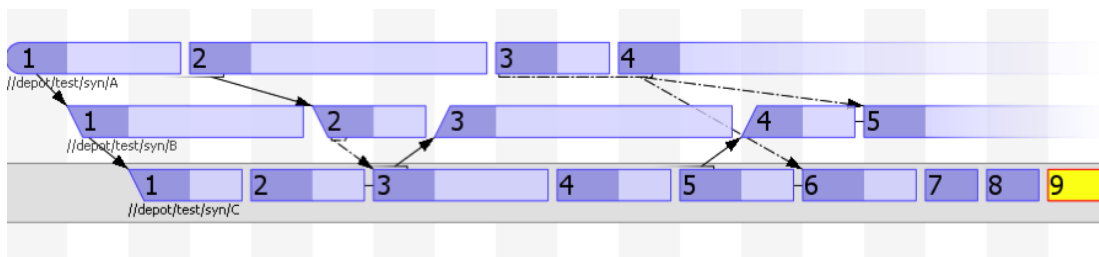
Subversion picks main#1 as the base, rather than dev#2. Subversion favors a base on the source branch even though dev#2 yields fewer deltas during the 3-way merge.



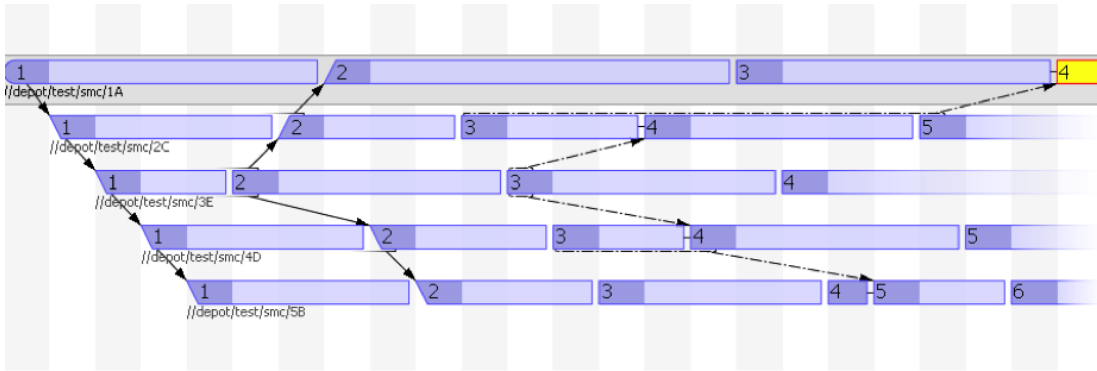
Subversion picks dev#1 as the base, rather than main#2, again favoring a less efficient base on the source branch.



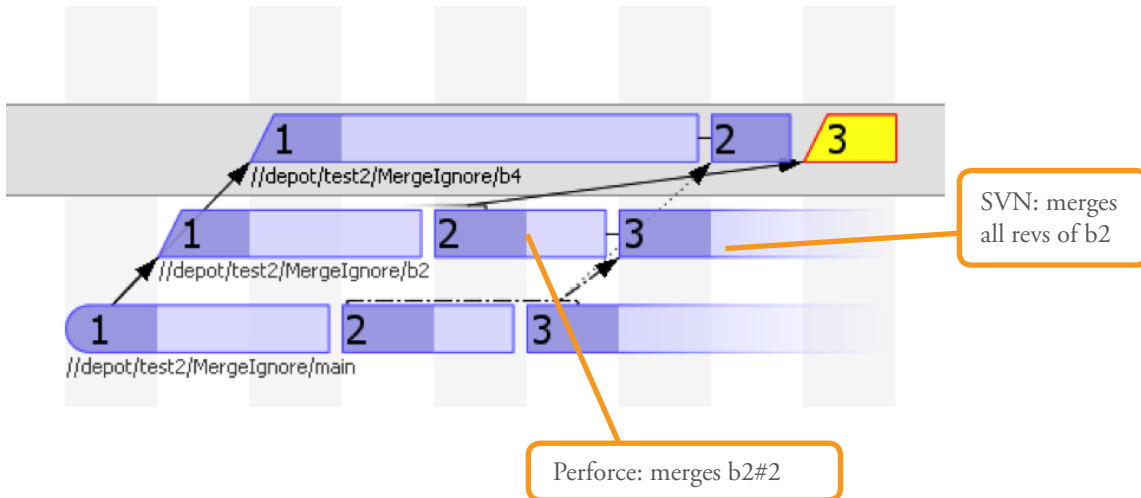
Subversion selects dev_foo#1 as the base, again favoring a less efficient base on the source branch. main_foo#2 would yield a simpler merge.



Subversion selects B#4 as the base, rather than C#6. It does not properly consider the merge credits granted indirectly from a previous merge from the A branch.



Subversion picks 1A#1 as the base, rather than 3E#3. By ignoring merge credit for much of the indirect merge history, Subversion forces us to consider nine diffs instead of seven.



Subversion does a naive merge of all contents of b2 into b4. In effect b4 receives the diffs from main#2,3 that had previously been ignored in the merge from main#3 to b4#2. If there are no conflicts, the merge would succeed silently, and the previously ignored changes from main would be reintroduced into b4, possibly leaving b4 in a bad condition.

The correct approach, as the Perforce merge arrows indicate, is to effectively merge only b2#2, as the contents of b2#3 had already been ignored in b4.

Learn More

Evaluating Perforce

More than 400,000 users at 5,500 companies rely on Perforce for enterprise version management. Perforce encourages prospective customers to judge for themselves during a typical 45-day trial evaluation. Free technical support is included with your evaluation. Get started: perforce.com/trial

Scheduling a Demo of Perforce

To learn more about Perforce, schedule an interactive demo tailored to your requirements: perforce.com/product/demos

Migrating to Perforce

Perforce Consulting Services has experience assisting customers with migrations from various legacy software version management systems. For more information, visit: perforce.com/consulting

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