

Version Control Is For Everyone

Nearly everyone in every organization has to manage multiple versions of one file or another. Contracts, benefit plan documents, router configurations, org charts, web content, accounting system scripts, software source files, even books. All are documents that you (or someone else in your organization) change more than once.

So what is the most recent version of your document? Is it orgchart-100701, orgchart-070110, or was it orgchart.last? Managing file versions with imaginative suffixes is a major step on the path to confusion and potential disaster. File naming is a version control system that doesn't scale, that isn't necessarily as intelligible to others as it is to you, and that leaves your organization with no control over one of its most valuable assets: its data.

Fortunately, version control is an issue that software developers have been managing for decades, and their experience is embodied in open-source and commercial software products that are useful for developers and non-technical people alike. Version control allows you to find the most recent version of a file quickly, review how and why it has changed over time, zoom in on what has changed line by line, and make your changes known and understood to everyone else who may touch the file in the future.

Subversion vs. Git

Two excellent candidates are the topic of this article: Subversion and Git. Both tools are open source, they do the job that you need to get done, and they won't break the bank. If you Google them, you'll find arguments for why one is better than the other. It doesn't matter. What matters is which one is best for your organization and the people needing to manage files.

Both of these systems incorporate the lessons learned from years of experience with such tools in the software development community. Decades ago, programmers worried about how to handle multiple versions of a single file. Then tools evolved to support collections of files that all had to do with the same project. Along came the Internet, and tools evolved once more to support versioning on any machine anywhere, and with more than one user manipulating the same files at once. Today these tools are called distributed version systems.

The basic concepts are the same. You check out a project (a file, collection of files, or complex hierarchy), make changes, test them out, and then check them back in (or commit). You can recognize a coherent set of files by a unique version number, read what has changed over time through logs, and quickly compare

different versions by seeing only the changes between them.

Distributed version systems

Both Subversion and Git are distributed, concurrent version systems; distributed in that they support users over the network wherever they are, and concurrent in the way they can cope with more than one person working on the same file at once (it turns out that they rarely change the same parts of the file).

Subversion is a centralized system with a single repository that's accessed through one of a number of network protocols. You can see projects and their files in shared directories, through a web-based interface, or through a click of your mouse in your system's file browser. When you commit changes, your files are checked into the central repository from which anyone else can update their copies, whether or not they are works in progress (Figure 1).

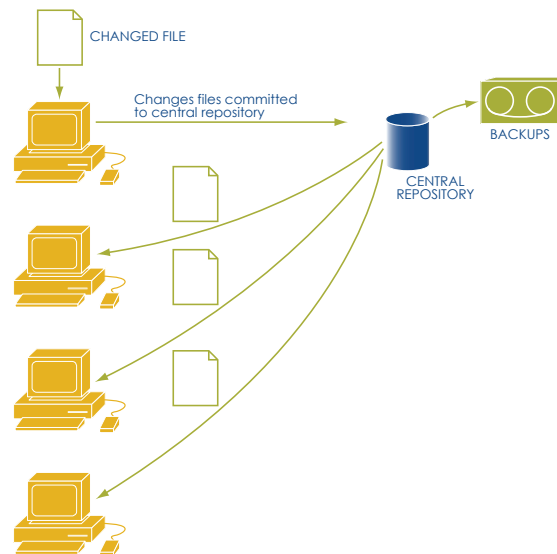


Figure 1: Subversion is a distributed version system with a centralized repository

Git is a decentralized system in which every user has a complete copy of the entire repository. Changes are checked in locally, and then propagated to all of the other copies of the repository in a peer-to-peer fashion (Figure 2). You can probably already see the appeal that Git has for fast-moving development projects, and why programmers love it.

Consider backups. Subversion's centralized repository lends itself to making consistent, point-in-time backups from a centralized

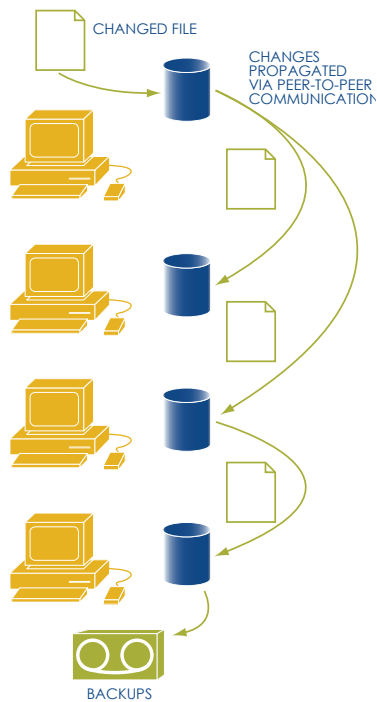


Figure 2: Git is a distributed version control system with a decentralized repository

storage system. This simplifies overlaying your organization's data asset management policies onto a Subversion repository. You can back up and retain data as your company, governmental, or industry regulations require, and users don't need to care.

Backing up a Git repository, however, is like taking a snapshot of a bird in flight. You can make a backup of one client, but it may or may not represent a consistent state of the repository. Git implements data reliability through its multiple distributed copies of the repository. This may or may not fit in with your data asset management policies.

and merges are the norm in the open-source community, so Git makes them simple and fast. Subversion makes these steps a little bit more difficult.

- **Security.** Both tools give you control over who can see the repository and commit changes. Both tools secure data in transit. Neither tool encrypts data at rest.
- **Space and time.** The community argues about which tool is the fastest and has the smallest repository for a given project. We think these arguments are specious, because the value of your data assets is much larger than the cost of disk space or network bandwidth to move repositories around. The space-and-time differences between systems are not enough to dissuade your users from using whichever one you choose.

Rich ecosystem

Subversion inhabits a rich ecosystem of add-on components that can do everything from adding built-in versioning to integrated development tools such as Eclipse to augmenting Windows Explorer so that versioning is a click-of-the-mouse activity. This is important if you're managing revisions (see Figures 3 and 4) of assets such as contracts and benefits plan documents, because the users undertaking these tasks are probably not as technical as your programmers.

While Git also has a graphical user interface, its strengths are at the command line where programmers are happiest. Git's GUI is more of a work of progress than the options with Subversion.

Whatever your digital assets, consider the benefits of managing their versions with something other than ingenious filename suffixes. When making a choice between Subversion and Git, or other open-source or commercial tools, don't get caught up in the "which is better" arguments. The question you need to address is which tool is a better fit for your specific needs, and for the people who will use it. If you're an up-and-coming software development shop and want to give your developers bragging rights along with the utmost credibility in the open-source world, Git is probably the best fit. If you want to use version control across your company and support both technical and non-technical users, think Subversion.

In all cases, once you start

Feature wars

Both Subversion and Git have similar features; they differ in how fast or easy it is to access their features. Consider the following:

- **Revision control.** Both systems do the obvious, giving you tools to check out projects, make changes, compare between versions, synchronize with others, merge, and commit. Both tools can handle all kinds of files, and they understand different line-ending conventions between systems such as Microsoft Windows, Unix, and Linux. Both systems handle binary files, but neither system understands them. For example, if you edit and change a JPEG file, don't expect them to highlight the changes in Aunt Millie's nose before and after you used Photoshop. The same is true for other file formats, such as Word's binary formats. If you want a human to be able to understand changes, save files in a text format.
- **Revision numbers.** Subversion makes it simple with sequential version numbers. You know that revision 1.146 is the one after 1.145. Git identifies versions with a 40-character SHA1 digest, so for all practical purposes you have to consult the log to understand the ordering of versions.
- **Branches and merges.** These occur when someone wants to take the project their own way, for example to make a massive set of changes or implement a fundamental new feature. The branch holds the changes until they are ultimately merged back into the trunk... or not. Branches

VERSIONING THE UNIX AND LINUX SYSTEM ADMINISTRATION HANDBOOK

The new *Unix and Linux System Administration Handbook* was an effort by 10 authors spread across 6 states and 3 continents. We authored the book in Adobe FrameMaker, which allowed us to store each chapter as a readable-text Maker Interchange Format (MIF) file. Our choice of a text format allowed us mere mortals to view and understand differences between versions when we asked Subversion to show us changes.

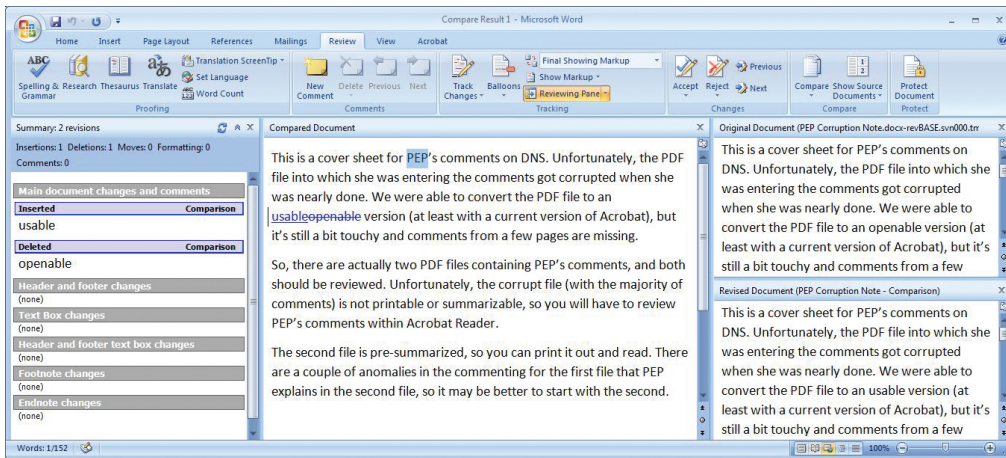


Figure 3: One Subversion add-on component shows differences in Word documents on screen.

using a distributed version system to manage your files, you'll wonder how you ever survived without it.

This article was reprinted from the Q3 2010 issue of *The Barking Seal*, a publication of AppliedTrust. You can subscribe to *The Barking Seal* online at <http://www.appliedtrust.com/barkings seal>. "AppliedTrust" and the AppliedTrust logo are registered service marks of AppliedTrust. All other trademarks are registered to their respective owners.