The History of Version Control

Nayan B. Ruparelia
Hewlett Packard Enterprise Services
<nayan.ruparelia@hp.com>

DOI: 10.1145/1668862.1668876
http://doi.acm.org/10.1145/1668862.1668876

A Version Control System (VCS), also known as a Revision Control System or Source Control System, is required when developing projects above a few hundred lines of code or where more than one developer needs to collaborate on a project. It is therefore used extensively on most software projects.

We start this article with a brief introduction to version control, then consider its trajectory from its early days to the current state of the art, and finally conclude with a brief discussion of what the future might hold for software versioning. Along the way, we shall discuss versioning taxonomy, compare a VCS with the broader configuration management capability, and consider the incredible impact that open source software has had in the development of VCS technology.

Keywords: SEN, history column, version control.

Introduction

What Is Version Control?

A VCS enables developers to keep historical versions of source code and project files that are under development and thereafter to retrieve past versions. It stores version information for every file (and the entire project structure) in what is generally called a repository. Inside the repository, several parallel lines of development, normally called branches, may exist. This is particularly useful when you wish to keep a maintenance branch for a stable, released version of your project whilst still working on another version, or branch, that is a newer version. Another use of branches is to work on an experimental or test version of your project that is distinct from the released version; this is known as sandboxing because the test version is decoupled from the release version of your project. A VCS also lets you give labels, often referred to as tags, to a snapshot of a branch to ease extraction in future. This is used for signifying individual releases or the most recent usable development version of the project.

Brief Taxonomy of Version Control

Before we delve into the evolution of various version control systems and their mechanisms, it is important to understand some of the terminology that is used in versioning. Below is a condensed list of terms used commonly by a VCS.

- Branch: declares the predecessor revision, and development continues from there.
- Changeset: a description of the difference between one source tree and another; a changeset can therefore be used to produce a revision of the source tree from another one.
- Checkin: this is when a user updates the repository with changes to the files, so that others may access them.
- Checkout: creates a working copy of the files in the repository on the client machine.
- Commit: same as checkin.
- Head: the latest revision of the repository on the client machine.
- Merge: this is when you incorporate others' changes to the repository into your local copy.
- Tag, or Label: Describes a branch.
- Tree: working directory of files and directories that is created when you perform a checkout.
- Trunk: Highest location within a head of a repository; it can be thought of as a branch without a name.

Common Features of a VCS

A good VCS has the following features:

- Backup and Restore.
  You can save files as they are edited and have the facility to jump to a previous version.
- Synchronization.
  Allows you to share your source code files and update your codebase with the latest version.
- Undo Changes.

Changes you make to the code can be undone by going back to a version that was committed in the past.

- Track Changes.
  As files are updated, you can leave messages explaining why the change happened; this enables you to see how and why the code evolved over time.
- Track Ownership.
  You can tag changes with the name of the person to track who made the changes.
- Sandboxing.
  You can make temporary changes in an isolated area, called a sandbox, to test and try out your code before it is checked in.
- Branching and merging.
  This is akin to a larger sandbox, where you can branch a copy of your code into a separate area and modify it in isolation (tracking changes separately). Later, you can merge your work back into the original codebase.

A comprehensive set of features together with a comparison of current VCS tools is provided by Shlomi Fish [1].

A recent innovation to versioning is the Distributed Version Control System (DVCS); *arch* (started in 2001 by Thomas Lord, and adopted by GNU in 2003) was one of the first DVCS. Using distributed repositories, rather than a single one, files from the repositories are merged on the basis of trust, as defined by the historical quality of changes. The main advantages of a DVCS over a VCS are: a) it allows users to work even when not connected to a network, b) most operations are faster because the local machine effectively acts as the 'central' repository, and c) there is no single point of failure.

VCS And Configuration Management

Now that we have considered what a VCS is, and the terms and features commonly used for versioning, let us step back to look at the broader picture of configuration management and the part that VCS plays in it.

From an IT Service Management (ITSM) perspective, configuration management is a process that ensures that assets, called configuration items (CIs), that are part of the IT infrastructure are identified and their information is maintained and updated in a repository, called a Configuration Management Database (CMDB). In addition to the VCS being used by the CMDB to track CI changes, the use of a VCS is central to many ITSM processes such as Change Management (process for the tracking of change requests).

Because software development is considered as a separate activity from service management, the source-code VCS remains separate and disconnected from the overall ITSM configuration management. This situation is bound to change in future as software development itself begins to be viewed as a service within the enterprise and the development artifacts become integrated into the CMDB as CIs. To some extent, this is currently possible in a roundabout way by the use of a federated CMDB that could interface with a VCS. So, in future, the boundaries between configuration management and VCS are likely to become increasingly blurred.

Historical Perspective of Open Source VCS

Open source VCS software today commands the lion share of users. Also, many commercial VCS tools have borrowed extensively from open source VCS offerings to the extent that it would not be inaccurate to state that open source VCS technology is ubiquitous. Hence, the history of VCS is mostly interwoven by that of open source VCS, which we consider in this section.

SCCS

Developed at Bell Labs in 1972 by Marc J. Rochkind, Source Code Control System (SCCS) was the first VCS. Rochkind wrote SCCS on an IBM S/370 computer running on OS/MVT, and it was later written for UNIX running on a PDP-11. Subsequently, several UNIX distributions included SCCS as part of their standard command set.

The SCCS file format introduced the interleaved delta storage technique and, although SCCS itself is considered obsolete, some present day VCS such as BitKeeper still use the same file format and storage technique.

SCCS remained the dominant VCS until Revision Control System (RCS) was released.

RCS

RCS was developed in the 1980s by Walter F. Tichy while he was at Purdue University. Whereas SCCS consisted of a series of commands for implementing version control, RCS automated the process so that storing, retrieval, logging, identification, and merging of file revisions became easier. Using the UNIX *diff* command, it stored file revisions and so was suited to providing revision control for source code, documentation, procedural graphics, papers, and form letters rather than binary files although these could be handled with reduced efficiency.

Some of the drawbacks of RCS were that it operated only on single files and had no facility for handling an entire project or directory structure. Also, its version command syntax was cumbersome; the result being that teams shunned its branching feature and instead simply used its built-in locking mechanism to work on a single head branch. More significantly, it did not have the facility of concurrency to allow developers based in disparate locations to collaborate on the same project.

This set the scene for the debut of Concurrent Versioning System (CVS), which addressed these drawbacks by itself using RCS.

CVS

Almost every developer will have used CVS at some time because of its strong presence on the software development scene for at least two decades.

Originally called *cmt*, short-form for commit, CVS was born in 1985 when Dick Grune needed to work collaboratively with his students, Erik Baalbergen and Maarten Waage, on the development of the C compiler named ACK (Amsterdam Compiler Kit). Grune wrote CVS using Bourne shell scripts that he committed to comp.sources.unix (implicitly mod.sources [2]) on 23rd November, 1985 followed by a revised version on June 23rd, 1986, describing CVS as a front-end to RCS. The code was later rewritten in C by Brian Berliner in 1988.

CVS introduced new concepts to versioning in a number of ways: it used a client/server model, supported branch imports, used unreserved checkouts and utilized symbolic mapping. Let's discuss these in turn below.

Client/Server Model

This enabled developers at disparate locations to function as a team even when encumbered by slow modems as it stored the version history on a central server whilst the developers worked on a copy of the files on their client machines. However, this meant that network access was required when CVS operations, such as updates or checkins, needed to be performed.

Branch Imports

In cases where developers needed to maintain their own version of files, CVS allowed one team to import branches from another and merge the changes, if required.

Unreserved Checkouts

Unreserved checkouts allow more than one developer to work on the same files at the same time so that there is no lock on files when they are checked out.

Symbolic Mapping

CVS has a database that provides symbolic mapping of names to components of a larger software distribution, thus facilitating the naming of collections of directories and files. The result us that a single command can manipulate an entire collection of directories and files.

Over the years, changing CVS has been popular with developers to the extent that the acronym *YACC* (Yet Another CVS Clone), a pun on the Unix tool *yacc* (yet another compiler compiler), was coined. Noteworthy CVS replacement projects are CVSNT (CVS for Windows NT, released in 1998), OpenCVS (2008), Subversion (2004), and various distributed version control systems. In the early to mid 2000s, many users of CVS began to replace it with Subversion, which was developed to be "a better CVS".

Subversion

CollabNet used CVS in its collaboration software, CollabNet Enterprise Edition, and in order to improve it, they contacted Karl Fogel in February 2000; Karl was the author of a book on the use of CVS on open source projects. Independently, around that time, Carl and his friend, Jim Blandy, had been looking at ways to better manage versioned data; Jim was using the name Subversion to describe the project. Along with Ben Collins-Sussman, CollabNet hired Karl, whilst Jim got his employer, Red Hat, to donate him to the project. Detailed design work began in May 2000 and Subversion quickly attracted a community of active developers.

Released under an open source license, Subversion gained wide usage and replaced CVS as the tool of choice within the open source community. Currently, a large number of extensions and tools exist for Subversion [4]; these have been contributed by a vast number of developers.

Figure 1 shows the internal architecture of Subversion, which may be accessed by a command-line, called the *svn* program, or graphical interface. The *svnserve* daemon is the Subversion server, which can be mirrored to another *svnserve* server incrementally and unidirectionally using svnsync. Using a plug-in module, DAV, for the Apache HTTP server, the repository can also be accessed over the network.

Lately, many open source projects have replaced Subversion or CVS with a distributed version control system such as Git.

Git

BitKeeper, owned by BitMover, is a proprietary VCS; its key feature being the ease with which distributed development teams can keep their own local source repositories and still work with the central repository.

In 2002, BitMover released a version of BitKeeper under a community license that allowed developers them to use the software for free provided they did not participate in the development of a competing tool (whether open source or proprietary) for the duration of its use and an additional year. Despite concerns, most notably from GNU Project founder Richard Stallman, Linux kernel developers decided to use BitKeeper.

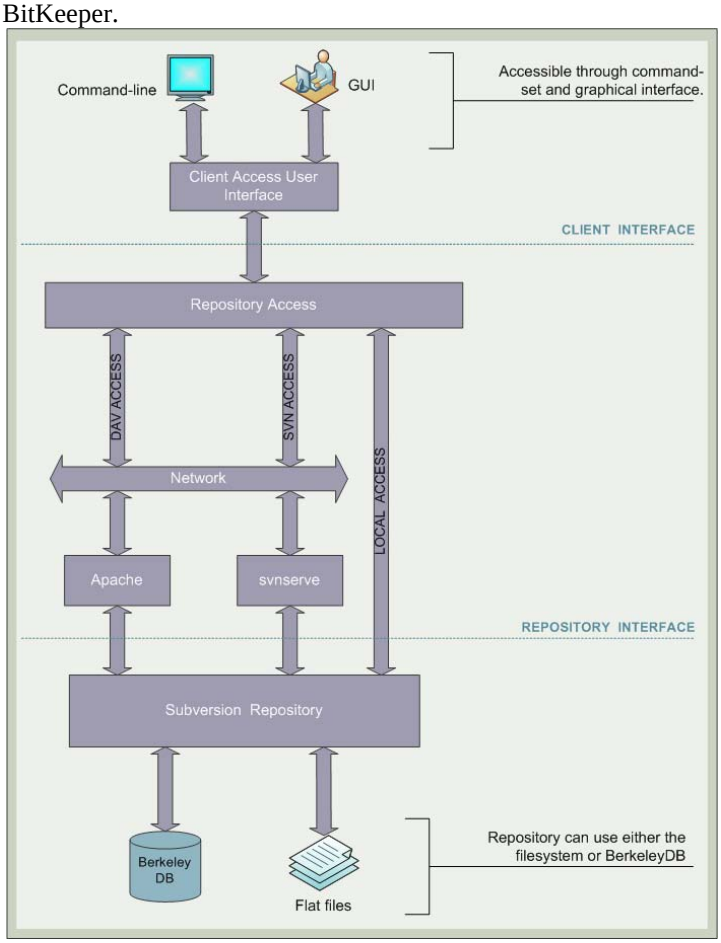

Figure 1: Subversion's Flexible Repository That Uses BerkeleyDB Or The Filesystem

Not being allowed to see metadata and compare past versions was a major drawback of the community version of BitKeeper, and one that significantly inconvenienced most Linux kernel developers. This led Andrew “Tridge” Tridgell, a developer on an unrelated open source project, to develop a BitKeeper client that rectified this inconvenience. Citing this, in April 2005, BitMover withdrew its community version of BitKeeper. In response, the Git project was launched with the intention of it becoming the Linux kernel's source configuration management software, and was eventually adopted by Linux developers.

Historical Perspective of Commercial VCS

Open source VCS tools, especially Subversion, have captured the market and mindshare of developers. However, the main commercial VCS packages of note are ClearCase, PVS, Visual SourceSafe and MKS. I shall consider these and their commercial predecessors in this section.

ClearCase

Written in C++ by IBM and distributed through the mid-1990s, Configuration Management Version Control (CMVC) was derived in part from software purchased from HP. It served as an object repository, and performed software version control, configuration management, and change management functions. It was replaced by Rational ClearCase when IBM acquired Rational Software.

PVCS

Polytron Version Control System (PVCS) was originally published in 1985 by Polytron and is currently sold by Serena Software. PVCS uses the locking of files as a mechanism for concurrency control by creating a parallel branch for the second committer so that modifications to the same project can exist in parallel. This differs from CVS where the second committer needs to first merge the changes via an update command and then resolve conflicts, should they exist, before committing his own changes. In this regard, PVCS has a simpler version control mechanism from a user's perspective.

MKS Integrity

MKS (Mortice Kern Systems) was initially started in 1985 by four students of the University of Waterloo as a consultancy. In 1987, they entered the version control market with MKS RCS, after having released the MKS Toolkit a year earlier; MKS ToolKit was a set of UNIX-like tools that ran on MSDOS. In 2001, MKS started to focus on the enterprise market, expanding from its desktop-based VCS to a multi-tier software change and configuration management system. The new product, now known as MKS Integrity, was written using J2EE and is currently part of MKS's ALM (Application Life-cycle Management) suite.

Visual SourceSafe

In 1994, Microsoft bought a company named One Tree Software, which had created SourceSafe 3.1, and replaced its less powerful VCS called Delta with SourceSafe. SourceSafe was then a 16-bit application that did not use a client/server model. Consequently, it was more suited as a desktop VCS for a single developer. Microsoft modified SourceSafe 3.1 and created a 32-bit version which it released in 1995 as Visual SourceSafe 4.

Like CVS that is built upon RCS, Visual SourceSafe does not support atomic commits of multiple files; this, coupled with a direct, file-based access mechanism that allows any client to modify a file in the repository after locking it, has caused Visual SourceSafe to be unstable at times. Indeed, Microsoft itself has largely preferred to use a version of Perforce named SourceDepot for its own in-house development projects. Partly to mitigate the instability, from version 2005 onwards, Visual SourceSafe began using the client/server model. However, the optimum environment for Visual SourceSafe still remains a small team, comprising at most five developers, that accesses the VCS repository over a local-area-network. This is reflected in Microsoft's Roadmap [3], where the Team Foundation Server, Microsoft's new VCS that is bundled with Visual Studio Team System, is recommended for larger, multi-user, enterprise projects.

Version Control Today

VCS has become integrated in many products ranging from collaboration tools such as Microsoft Sharepoint to test tools such as the HP Quality Centre. In that sense, VCS has become an enabling technology for many unrelated applications.

Today, you would expect to see the following features in a VCS or a distributed VCS:

- Atomic commits: these are all-or-nothing commits; the tree must be uncorrupted before the commit begins, and commits are not visible until they are complete. If the atomic commit is interrupted, it remains invisible and is rolled back.
- Changeset oriented: instead of tracking individual files (as in CVS), a present-day VCS tracks changesets, which are akin to patches.
- Ease of branching: branching can span archives instead of just a source tree.
- Advanced merging: merging can take into account which branch contains which patch, and can do three-way merging between the patch, previous branch and current head.
- Cryptographic signatures: a changeset is stored with a hash to prevent accidental corruption. These hashes can also be signed, using tools such as *gnuPG*, to prevent unauthorized modification.
- Renaming: files and directories are tracked by a unique identifier, instead of by name, and so can be easily renamed whilst still preserving their revision history.
- Metadata tracking: file permissions are tracked, and symbolic links are tracked the same way as files and directories.

Version Control In Future

What the future may hold for VCS is difficult to predict as there are so many different systems and flavors around, especially in the open source world. One trend, however, is certain to persist: DVCS is becoming increasingly popular in the open source community and, over time, will replace centralized systems. At present, the corporate world has increasingly become reliant on open source tools, but its take-up of DVCS will take much longer; some corporations that rely on open source VCS and wish to stay with the centralized systems may be compelled to maintain them.

Another trend that we shall see continue into the future is for VCS to become increasingly integrated with: a) the entire software life-cycle, from requirements capture to defect tracking, and b) the broader configuration and change management tools and processes as defined by ITSM frameworks such as ITIL.

What I should like to see develop in future, however, are the following features:

1) Source code and related artifacts related to the project (emails, documents, test results, etc.) within an atomic commit.
2) Bi-directional replication at VCS level similar to the way that database replication takes place in order to ensure that central repositories have a secondary VCS. Of course, this does not apply to a DVCS.
3) Capture and track automagic dependencies; these are usually run-time, optional dependencies that the software builder enables when invoking a `./configure` script.
4) In a DVCS, it would be nice to have some form of role-based access control, possibly one that is integrated with LDAP or Active Directory. This will accelerate the adoption of DVCS by corporations.

Further Reading

Below is a list of some VCS tools that are available either as open source or on a commercial basis.

- Git (open source) [http://git-scm.com/]
- CVS (open source) [http://www.nongnu.org/cvs/]
- Arch (open source) [http://www.gnu.org/software/gnu-arch/]
- Subversion (open source) [http://subversion.tigris.org/]
- BitKeeper [http://www.bitkeeper.com/]
- Perforce [http://www.perforce.com/]
- ClearCase [http://www-01.ibm.com/software/awdtools/clearcase/]
- Visual Source Safe [http://www.microsoft.com/vstudio/previous/ssafe/]

References

[1] Version Control System Comparison; http://better-scm.berlios.de/comparison/comparison.html#atomic_commits
[2] Original CVS Shell Script Code; mod-sources; http://groups.google.co.uk/group/mod.sources/msg/2ebab72ac0744fb8?hl=en&dmode=source
[3] Visual SourceSafe Roadmap; http://msdn.microsoft.com/en-us/library/aa302175.aspx
[4] Contributions to Subversion; http://subversion.tigris.org/tools_contrib.html