

Version Control

Panagiotis Louridas

Software evolves in small steps or versions. Often these versions are results of collaborations among different persons. At times we want to fall back to a previous version or compare different variants. Or, we need to trace changes to change requests. All these tasks relate to version control and show that tool support in this domain is indispensable, as it is with editors and compilers. Panagiotis Louridas describes available tools and shows why the CVS open source tool is so popular among practitioners.

—Christof Ebert

On 1 November 1876, Mark Twain wrote to a friend of his youth that “there was but one solitary thing about the past worth remembering, ... the fact that it is past—can’t be restored.” A developer, however, often needs to both revisit and restore the past, and—pace Twain—both are possible.



Writing programs is fundamentally a problem-solving process. We often follow a path only to find out that we must retrace our steps because yesterday’s version turned out better than today’s. Sometimes we even need to follow more than one path in parallel. And we shouldn’t forget that, with few exceptions, teams, not in-

dividuals, develop software. We must make sure that our work doesn’t trample on a colleague’s work.

Version control tools come to our rescue. They let us retrace and restore our programming past, proceed along two or more development lines in a single project, and coordinate our work with the work of others in the same project.

One of the oldest, most venerable VC systems is CVS (Concurrent Versions System, www.nongnu.org/cvs). CVS began as a bunch of shell scripts written by Dick Grune and posted to the newsgroup comp.sources.unix in 1986. Brian Berliner rewrote it in C in 1988,

and Jeff Polk later contributed important parts. Although CVS is the oldest, it’s still the most widely used VC system because it’s mature, stable, and fielded in a vast corpus of projects. Because CVS is the standard to which all other VC tools are compared, I use it here to illustrate basic VC concepts.

Basic concepts

The basic VC idea is to separate the files that programmers work on, called *working copies*, from the *master copies* of the same files, which are stored in a repository. The programmers never work on the repository. They check out their working copies from the repository, make changes, and then check in (commit) their changes. Every time a programmer commits a file, the VC system creates a new version in the repository. In this way, the repository contains all versions of a file.

This doesn’t waste space: VC systems are frugal, storing only the differences between successive versions. Because “version” also refers to a specific software product release, VC systems often use “revision” to refer to the successive versions of a VC file.

When two or more programmers want to use the same version, the potential for conflicts arises. Two approaches exist to handle conflicts.

In the *lock-modify-unlock model*, developers must obtain a lock on any file they wish to change. For as long as a developer has the file’s lock, no one else can change the file, although it’s possible to check it out (say, for viewing or

compilation). When the developer with the lock finishes working on the file, he or she commits the changes to the repository and unlocks the file.

In the *copy-modify-merge model*, developers are always free to modify their working copies. If conflicting modifications appear, the VC system alerts the developers of the conflict, and they must resolve it by merging the modifications.

The lock-modify-unlock model is simple but restrictive. It's especially cumbersome in distributed development, when different people in different geographic locations and possibly different time zones work on the same project. The copy-modify-merge model is more egalitarian, but it requires developers to work around conflicts amicably. CVS employs the copy-modify-merge model.

Using CVS

When you start working on a project stored in a CVS repository, you must first get a working copy of the project. Suppose the project is called *fib*. You get a copy by using the `checkout` command:

```
cv$ checkout fib
```

This command creates a new directory called *fib* in the current directory and populates it with working copies of all the project files in the repository. It also adds some special files and directories that CVS needs to perform its operations, but programmers normally don't need to concern themselves about them.

All CVS commands start with `cv$`, followed by the CVS operation's name and target, usually a set of files or directories. After you've worked on a file, you can commit the changes to the repository with the `commit` command. Suppose you've been working on the file `foo.c`. The command would be

```
cv$ commit foo.c
```

CVS checks to see whether another programmer has committed a newer version in the meantime. If not, it asks

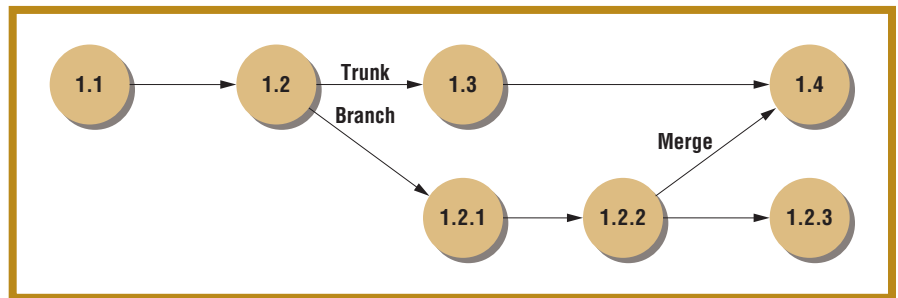


Figure 1. A CVS branching example.

you to enter a log message describing the changes. CVS then adds a new version of the file to the repository; to be more precise, it stores the current version of the file and the differences between the current version and the previous one. It does this at each and every commit, thus maintaining a complete history of the file.

Suppose a colleague has committed a newer version in the meantime. CVS then alerts you that commit isn't possible, and you must reconcile the differences between the latest repository version and your working copy. To bring your working copy up to date with respect to the repository, you use the `update` command:

```
cv$ update foo.c
```

If CVS can reconcile the differences without conflict, it will do so automatically, and then you can commit. Otherwise, CVS will change your working copy, demarcating the changes between it and the repository. It's up to you and your colleague to edit the file, fix the conflicts, and then proceed to commit.

Suppose now that you're working on a version and need to go back to a previous one. You can do this easily by running `cv$ checkout` with an option specifying the previous version's date. You can also request a version by a *tag*—that is, by a name given to a revision through the `cv$ tag` command.

You can find the differences between any two versions with the `cv$ diff` command. The differences show up in diff format—that is, the line difference format that the Unix `diff` command uses. CVS also uses this for-

mat to store version differences in the repository.

At any point in a file's history, you can start a parallel development line. Imagine you've shipped your software and are now actively working on the next version. Reports from the field indicate a bug in the shipped version. You can go back to that version and create a *branch* on which you incorporate the bug fix. Figure 1 shows the evolution of a file that has two branches; the main line is called the *trunk*. As the figure shows, branch changes can be incorporated (merged) with the trunk.

Branches aren't just for bug fixes. For example, you might want to explore a different implementation technique along with the main line of development, or you might want to customize a product in some ways, again without affecting the main development line.

CVS has many more features, including CVS commands to add new project files and directories (`cv$ add`), to remove a file from a project (`cv$ remove`), and to import a directory structure into CVS and start a new project (`cv$ import`). For tracking purposes, CVS keeps a complete log history for each file and offers commands to examine the log for who did what and when at different levels of detail (`cv$ log` and `cv$ annotate`).

Keyword substitution is another useful feature. CVS looks for specific strings in your source and replaces them with the indicated information at commits (for instance, it replaces `ID` with the file name, date and time of the commit, and the committer's user name).

Table 1**Comparison of version control systems**

Feature	CVS	Subversion	BitKeeper	SourceSafe	Perforce	ClearCase	Synergy
Platforms	MS Windows (clients), Unix	MS Windows, Unix	MS Windows, Unix	MS Windows	MS Windows, Unix	MS Windows, Unix	MS Windows, Unix
Atomic commits	No	Yes	Yes	No	Yes	Yes	Yes
File/directory moves, renames	No	Yes	Yes	With workaround	Not directly	Yes	Yes
File/directory copies	No	Yes	Yes	Yes, to a point	Yes	Yes	Yes
Remote repository replication	No	Yes, via tool	Yes	No	Yes, via tool	Yes, via tool	Yes
Repository change propagation	No	Yes, via tool	Yes	No	Not known	Yes, via tool	Yes
Repository permissions	Limited	Yes	Yes	Limited	Yes	Yes	No
Support for treatment of change sets as atomic	No	Partial (no individual cancellation yet)	Yes	No	Yes	Yes, via branching	Yes
Line-wise history tracking	Yes	Yes	Yes	Not directly	Yes	Yes	Via scripting
Work only on part of the repository	Yes	Yes	No	Yes	Yes	Yes	Yes, in projects
User license cost (up to 20 users)	Free	Free	US\$1,750	US\$200	US\$800	US\$4,125	US\$2,900

Open source projects often want to publish CVS repositories on the Web so that others can inspect the code at their ease. CVS offers various ways to do that. Fortunately, CVS is well documented (see the “Version Control System Resources” sidebar).

Alternatives to CVS

CVS is a popular, practical tool that solves many problems arising in cooperative, distributed development. That doesn't mean, however, that it has no warts.

For instance, CVS doesn't handle binary files well. It treats them like all other files. However, computing line-by-line differences of binary file versions doesn't make much sense, and performing keyword substitution in them is catastrophic (CVS does provide an option to turn this function off).

CVS's treatment of directories is stringent. Once you create a directory, deleting or renaming it isn't easy. There's some justification for this. A software project's directories, unlike its files, don't—and shouldn't—change

frequently, because they reflect overall system architecture. Yet sometimes the need arises.

Table 1 compares CVS to several other products. The last row, cost, is only indicative, because different products might come bundled with add-ons. Moreover, prospective buyers might be able to get discounts depending on such factors as the number of users or how far in the quarter they ask for a quote.

Subversion (<http://subversion.tigris.org>) is an open source project that attempts to remain as similar as possible to CVS while improving its capabilities with additional features. For instance, Subversion offers directory versioning. It's also easier to handle file name changes in Subversion than in CVS, which requires a combined copy and deletion to rename a file. Subversion lets you create and store arbitrary properties (called *metadata*) along with any file or directory; it creates versions of the file properties just as it does for the file contents. It also lets you treat a collection of file or directory modifica-

tions as a single unit. In fact, Subversion versions are per project, not per file. Thanks to these and other goodies, Subversion has attracted support and a considerable user base.

BitKeeper (www.bitkeeper.com) is a well-known commercial, proprietary alternative. It was used until recently for the Linux kernel development—using a proprietary product to develop a flagship of the open source movement being a paradox of sorts. (BitKeeper was finally abandoned for political rather than technical reasons.)

Users of the Microsoft Visual Studio suite usually take advantage of Microsoft's Visual SourceSafe tool (<http://msdn.microsoft.com/ssafe>). Perforce (www.perforce.com) is another well-known commercial VC system. IBM Rational offers the comprehensive ClearCase suite (www-306.ibm.com/software/awdtools/clearcase). Telelogic Synergy (www.telelogic.com/corp/Products/synergy) comprises Change Synergy and CM Synergy, which automate change requests and configuration management, respectively.

Version Control System Resources

Table 1 is an abridged version of the material at <http://better-scm.berlios.de/comparison/comparison.html>, which comprehensively compares many version control systems.

"Version Control Systems," by Diomidis Spinellis (*IEEE Software*, vol. 22, no. 5, 2005, pp. 108–109), lucidly expands on the benefits of using VC. See also his "Version Control, Part I" contribution in the Software Engineering Glossary in the same issue (p. 107) and "Version Control, Part II" in the following issue (*IEEE Software*, vol. 22, no. 6, p. 113).

Open Source Development with CVS, 3rd ed., by Karl Fogel and Moshe Bar (Paraglyph Press, 2003, available online at <http://cvsbook.red-bean.com>) and *Version Control with Subversion*, by Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato (O'Reilly, 2004, available online at <http://svnbook.red-bean.com>) thoroughly explain these two systems.

Version Management with CVS, by Per Cederqvist and his colleagues (Network Theory Ltd., 2002), is the official CVS manual. Also known as "the Cederqvist," it's available online at <http://ximbiot.com/cvs/manual>.

Pragmatic Version Control Using CVS, by Andy Hunt and Dave Thomas (Pragmatic Programmers, 2003), is a brief, practical guide to VC.


"RCS—A System for Version Control," by Walter F. Tichy (*Software: Practice & Experience*, vol. 15, no. 7, 1985, pp. 637–654), describes an early VC system that greatly influenced the evolution of these tools.

"A File Comparison Program," by Webb Miller and Eugene W. Myers (*Software: Practice & Experience*, vol. 15, no. 11, 1985, pp. 1025–1040), describes the Unix `diff` utility's workings.

Choosing a VC system is an important decision. As Diomidis Spinellis points out in this issue's Tools of the Trade column (pp. 100–101), source code is only one part of a project's assets. Version history is also an important part—especially considering that the version repository format might not be portable, even if code is. In fact, migrating from one VC system to another is no minor undertaking. Although commercial tools might offer I/O filters and various tools ease porting between open source VC systems, the process is rarely fully automated in real-life, complicated projects. Developers should always be aware of the dangers inherent in vendor lock-in to proprietary formats.

UC systems are available at any cost for any platform. A software organization that doesn't use one is reckless and oblivious to the demands of actual project development, which requires many people to cooperate and, often, to

step back. And yet, many companies around the world keep on developing software with little more VC than simple backups.

Until you gain experience with VC, it's hard to realize that—far from being a drudgery (this is hidden away by the tools, after all)—VC is a source of enjoyment. When we first exercise the capability to revisit and restore our past and to work in parallel with our colleagues without stepping on each other's toes, we wonder how we'd been developing software until then. 

Panagiotis Louridas is a grid software engineer at the Greek Research and Technology Network and a researcher at the Eltron Software Engineering and Security research group of the Athens University of Economics and Business. Contact him at louridas@grnet.aueb.gr.

IEEE Software

How to Reach Us

Writers

For detailed information on submitting articles, write for our Editorial Guidelines (software@computer.org) or access www.computer.org/software/author.htm.

Letters to the Editor

Send letters to

Editor, *IEEE Software*
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
software@computer.org

Please provide an email address or daytime phone number with your letter.

On the Web

Access www.computer.org/software for information about *IEEE Software*.

Subscribe

Visit www.computer.org/subscribe.

Subscription Change of Address

Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Be sure to specify *IEEE Software*.

Membership Change of Address

Send change-of-address requests for IEEE and Computer Society membership to member.services@ieee.org.

Missing or Damaged Copies

If you are missing an issue or you received a damaged copy, contact help@computer.org.

Reprints of Articles

For price information or to order reprints, send email to software@computer.org or fax +1 714 821 4010.

Reprint Permission

To obtain permission to reprint an article, contact the Intellectual Property Rights Office at copyrights@ieee.org.