Version Control Systems

Timothy Daly

February 24, 2004

Abstract

Version Control Systems are software programs that maintain histories of changes to source code. We cover the most widely used system CVS in great detail. We then talk about some of CVS's shortcomings and introduce other systems that address these concerns. This lecture includes a guest presentation of GNU-Arch by James Blackwell.

Contents

1 Weekly News

Wall Street's Secret Affair With Linux [1]

> "There's been a marked change in attitude towards Linux in the last six to nine months," says Mark Hunt, Global Director, Enterprise Product Marketing at Reuters. Reuters has ported its Reuters Market Data Systems (RMDS), which provides real-time market data and financial news, to Linux. Merrill Lynch & Co. is running RMDS on Linux.
>
> "Every major Wall Street firm has at least a pilot going on with Linux," says Vern Brownell, the former CTO of Goldman Sachs. "And more than half of them are probably in production with mission critical applications."

This article is of interest because it hints at the increased acceptance of Linux in mainstream businesses. Because Linux does not have a unified public relations department there is no central source for Linux news. However businesses are much more willing to admit to using Linux which is a major cultural change.

Toronto Conference On Open Source Announced

> Simon Bates writes "The University of Toronto's KMDI [2,3] is hosting a conference to debate the future of open source models of development in software and beyond, addressing how this movement will affect the way we work, learn and stay healthy. Among the 30 speakers will be Eben Moglen, Columbia law school professor and legal counsel to the Free Software Foundation, who has recently described free software as: 'a social movement with specific political goals which will characterize not only the production of software in the twenty-first century, but the production and distribution of culture generally'. The conference will be held from May 9th to 11th and will be webcast."

There are several things of interest about this announcement.

First is that the essence of the conference is to "debate the future of open source models of development" rather than present accomplished work.

Second is that free software is characterized beyond the software realm as: 'a social movement with specific political goals which will characterize not only the production of software in the twenty-first century, but the production and distribution of culture generally'. This recognizes the effect of the gift culture on the Zeitgist.

Third is that the conference will be webcast. Generally you have to pay big bucks to attend conferences. They are giving the conference away for free.

Improvements in kernel development from 2.4 to 2.6 [4]
This is a discussion of source code management as it applies to the Linux kernel. The recent kernel development process was changed to encourage, but not require, the use of BitKeeper which is a commercial product.

There was and is much controversy surrounding the use of a commercial product to develop GPL code. Fights break out about once a month over BitKeeper. Linus Torvolds has stated that he sees no problem with using a commercial product if there is no equivalent free product and the commercial product has needed technology.

It is certainly true that kernel development needs the technology of changesets provided by BitKeeper.

The more religiously correct GPL people have started developing equivalent free technology (such as Gnu-Arch or Subversions). The CEO of BitMover [5] has created a license that allows free software to use BitKeeper for free provided that (a) the project keeps changesets on bkbits.net and (b) users of BitKeeper do not work on a project to develop a free replacement for BitKeeper.

What We Learned In The New Economy [6]
This is a discussion of the Internet boom and bust and the lessons learned. In particular they compare the old ideas with the new:

Then: The Internet changes absolutely everything.
Now: Absolutist statements are absolutely a bad idea

Then: Free Agent Nation is a utopia. The Brand Called You makes you more marketable than ever.
Now: Free Agent Nation is a jungle. The Brand Called You is the only way to survive.

Then: IT spending has fueled an unstoppable productivity boom that has ended the business cycle.
Now: Productivity is still strong – and so is the business cycle.

Then: Move first – or die
Now: Move first without a real business – and die

Then: The Internet gives the customer new, limitless power.
Now: New power, yes. Limitless, no.

Then: Destroy your business, or someone else will.
Now: Incumbent businesses aren't so easily destroyed.

Interest in this article goes back to the first lecture where the third point "The Brand Called You" was discussed. It used to be the case that opportunity was unlimited and you expected people to hire you because you were a computer person. Now the game becomes one of collecting "Whuffie" [9] in order to qualify for a job and the competition is tough. But as the article say, "The Brand Called You" is the only way to survive.

Subversion 1.0 Released [10, 11]

> Phil John writes "Subversion 1.0 has finally been released. The people who maintain CVS have given us a viable replacement for our de-facto (and aged) versioning system. If you're new to Subversion its feature list looks like fixes for everything that is wrong in CVS, renaming, directory structure and metadata version tracking, file deletion, proper management of binary files and it's pretty portable to boot." According to the download page, binaries may take a few days to appear.

Subversion is the follow-on project for CVS. There are a number of flaws in CVS which we will mention. Subversion corrects many of those flaws.

2 The Problem

The problem is simple. You have a job to do that will take longer than one session at the computer. Day after day you modify one or many files. Each day, or several times a day, you save your work.

At some point you make a mistake and want to go back to the way things were yesterday or some earlier day. However saving files generally overwrites older versions and they are lost.

There are many ways to attack this problem. Symbolics Lisp machines create a new copy of the file and never rewrite over old copies. Microsoft Word keeps changes inside the file so you can undo changes.

3 The CVS Solution

CVS will solve this problem. The idea is to keep every copy of the file every time you change it.

Which isn't what happens, of course. (The argument against this is that it requires a lot of disk space. Who cares? Isn't your work worth more than disk space?). There are optimizations. Instead of saving complete copies of the file you save the changes between the last copy and the current copy (the *"diff"*). And you only save copies that you *"check in"*.

3.1 The Basic Idea

Most of this material is a summary of the CVS documentation [7].

CVS is a version control system. You use it to record a history of changes to your source files. CVS will allow you to maintain a site that contains all of your code and a history of changes, called the repository. Repositories can be local or global. If the repository is global then developers worldwide can "check code out". Some are given write access and allowed to "check code in".

The basic cycle of code development using CVS involves:

```
check out an initial copy of the repository into a local directory
for each set of changes
    make the changes in your local directory
    check in the changes to the repository
    commit the changes
```

The concept of "checking code out" involves getting a local copy from the repository, usually of the latest code, but sometimes you might want a pre-change version. CVS can do this. This enables you to back up in time. Most frequently this is useful when you had a previous working version and some change you made has introduced a bug. It is then possible to walk backward thru the changes to find out which change actually caused the bug.

CVS only weakly supports this idea thru the *admin* command. Some versioning systems lock every file you check out. Locking a file means that no-one can modify the file while you have it checked out. Locking files may seem like a good idea as it eliminates the possibility of two people trying to modify the same file at the same time. Locking can cause problems if one developer checks out a set of files and then goes on vacation. This can bring development to a grinding halt. In lieu of locking files CVS will allow you to query the *status* of a file. We will talk about this later.

Changes that you make locally are done using your normal editor. The only special event is when you add or delete a file. If this happens you have to tell cvs using a command like:

```
(create the file)
cvs add (filename or directory)
```

or

```
(delete the file)
cvs delete (filename)
```

Note that you can't delete directories.

The concept of "checking code in" means that you have changed a file and you want to put the changes back into the general pile for others to see. The problem is what to do with the code that is already there. Various systems maintain fairly similar answers to this question but the details are important. When you check the code in you *commit* it. At that time other developers

can see your changes and old copies of the file are hidden unless you explicitly request them.

CVS software is available free online [8].

3.2 CVS is not ...

CVS is not a build system. CVS stores a history of your files in an efficient way. It does not enforce any special build mechanism.

CVS is not a substitute for management. You have to communicate with people, not with machines.

CVS is not a substitute for developer communication. Communication is fundamental. Use mailing lists and IRC channels. Conflicts need to be worked out by the developers. We will cover mailing lists in a later class.

CVS does not have change control. Change control has 3 aspects, none of which are handled by CVS. The first aspect is bug tracking. We will cover bug reporting and tracking in a later class. The second aspect is managing related changes to multiple files. This idea is now called changesets. GNU-Arch handles this and we will discuss it later in this lecture. A third aspect code review which implies that a developer's output is reviewed and possibly changed by a second developer before the changes are accepted into the main line of code.

CVS is not an automated testing program. Testing has two aspects, neither of which CVS handles. The first aspect is testing that a new change works. The second aspect, called regression testing, is to ensure that the change does not break anything.

CVS does not have a builtin process model. Changes go thru stages and some companies set up standard processes to ensure that new code gets reviewed and tested before being applied. Rational Rose, a company recently bought by IBM, has a whole testing model with many required steps.

3.3 A Sample Session

CVS has the notion of a repository which is a directory where it stores all of the files. We can set up a local repository:

```
mkdir /home/daly/cvs
cvs -d /home/daly/cvs init
mkdir /home/daly/cvs/project
```

The first step just creates an ordinary directory to store files.
The second step initializes the directory by creating some CVS files.

The third step creates a directory to hold your project information.
Now you have a local repository.

You can now move to your home directory and start building a project:

```
cd ~
cvs -d/home/daly/cvs co project
cd project
(make a new file, eg. file1.c)
cvs add file1.c
cvs commit -m"add our first file"
```

The first step moves us to our home directory (/home/daly).
The second step "checks out" the project so we can work on it.
The third step moves into the newly created directory called *project*.
The fourth step does local editing on the project.
The fifth step tells CVS we want to add the new file to the project.
The sixth step *commits* our changes to the repository.

Since we finished making changes to the project we no longer need our local project directory. We can erase it completely:

```
cd ~
rm -rf project
```

But since the project is part of the repository we can recover it at any time:

```
cd ~
cvs -d/home/daly/cvs co project
```

and our directory and files appear again.

Most open source projects are already set up. In fact, if you set up your project with one of the major sites (freshmeat, sourceforge, savannah) they will automatically set up a CVS server for you.

3.3.1 Getting the Source

You need to know the location of the repository.

If the repository is local then you only need to know the path to the repository. Most open source projects share a repository with many widely separated developers so the repository is remote. In any case, the general form of the location looks like:

```
[:method:][[user][:pass]@]host[:[port]]/path
```

where:

method (optional) is one of *local*, *ext*, *server*, or *pserver*. The *local* method is assumed if the method isn't specified.

user is your userid on the system. This isn't required for local access.

pass is your password. This isn't required for local access.

host is the name of the machine hosting the repository.

port is the TCP/IP port that the CVS server is listen to.

path is the directory path to the repository.

Note that most of this information is unnecessary to reach our local repository. In fact, we need only specify it as:

```
cvs -d /home/cvs
```

Projects will tell you that the CVS is located some particular site. For instance, we'll assume you are going to work on Axiom. Axiom's CVS is at: **subversions.gnu.org:/cvsroot/axiom**.

We want to get a copy of the files which involves 3 steps.

The first step is to set up a shell variable called **CVSROOT**. You also (usually) need a second shell variable called CVS_RSH which tells CVS which program to use to communicate. Lets assume that your userid on the Axiom site is *"bob"* So you would type:

```
export CVS\_RSH=ssh
export CVSROOT=:pserver:bob@subversions.gnu.org:/cvsroot/axiom
```

The second step is to login to the site thus:

```
cvs login
```

which will prompt you for a password. The third step is to **"checkout"** the code thus:

```
cd ~
cvs co axiom
```

and you end up with the whole axiom development tree of code in a directory called axiom.

If you don't have a userid on the system and just want to fetch the code as an anonymous user you can do:

```
cvs -d:pserver:anoncvs@subversions.gnu.org:/cvsroot/axiom co axiom
```

3.3.2 Committing your changes

Now you edit some files and make changes. Assuming you didn't add or delete files you can save the changes (*"commit the changes"*). No one can see the changes and the repository won't know about the changes until you *commit* them. So you type:

```
cd ~/axiom
cvs commit -m"this is the log message describing the change"
```

3.3.3 Cleaning up

Now all of the changes you made are safely in the repository and you can delete the current files if you wish:

```
cd ~
cvs release -d axiom
```

The release command checks that all of your changes have been committed. The **-d** option will delete the axiom directory if everything has been committed.

Notice that once you commit the changes to the files you can destroy all of the files. They are now safely in the repository and you don't need to keep copies.

3.3.4 Viewing differences

If you have a file on your local disk and you're not sure what you changed you can ask cvs to tell you the difference (*"diff"*) between your local copy and the one in the repository:

```
cvs diff (yourfile)
```

3.4 The Repository

The repository can be on your local disk or on a remote machine.

```
export CVSROOT=:pserver:bob@subversions.gnu.org:/cvsroot/axiom
```

3.4.1 Creating a remote repository

To set up a repository on remote machine create the repository directory and initialize it:

```
mkdir /home/cvs
cvs -d /home/cvs init
```

and add the following line to `/etc/inet/inetd.conf`:

```
cvspserver stream tcp nowait root /usr/local/cvs cvsv --allow-root=/home/cvs pserver
```

Next we need to set up the CVS password file:

```
echo ``remoteuser:cryptpass:localuser'' >>/home/cvs/CVSROOT/passwd
```

You can use *cryptit* to encrypt a password:

```
cryptit nameofdog
 ==> Encrypted version of nameofdog = F3D4D74323E2A1
```

There are various front-end programs for working with CVS. The most popular one is called Cervisia [12].

3.4.2 CVS command output

CVS will output filenames with different kinds of characters that prefix their names. For example, you might see:

```
P: file1.c
U: file2.c
M: file3.c
C: file4.c
?: file5.c
A: file6.c
R: file7.c
```

where:

P means the file was "patched" with a newer version. Patchs are small lists of changes. CVS does this if it is more efficient than downloading a whole copy of the file.

U means the file was "updated" to get a newer version. A copy of the file was downloaded.

M means that you have a modified version in your directory so that it is different from the repository version.

C means that there is a conflict between your changes and changes committed by someone else. This implies that two people are working on the same file at the same time.

? means that you have a file in your local copy that you have not told CVS about. If you want to keep it you need to do a "cvs add".

A means that you explicitly added the file.

R means that you explicitly removed the file.

4 Subversion

4.1 New Features

Subversion [11] is a follow-on project to CVS. CVS suffers from a number of problems. Generally Subversion has tried to remain compatible with CVS unless there is a compelling reason to change. The Subversion site has a list of features which highlight where it differs from CVS.

4.1.1 Directories, renames, and file meta-data

Directories, renames, and file meta-data are versioned. CVS looses certain information like file execute permissions. Renames are done in CVS by:

```
mv oldfilename newfilename
cvs delete oldfilename
cvs add newfilename
```

Subversion directly support renames. In addition, Subversion will allow you to attach any "extra" (meta) data to a file.

4.1.2 Commits are truly atomic

Atomic commits are basic to transaction processing. If you go to an ATM machine and transfer money from account A to account B you want to make sure that either (a) the transfer did occur completely or (b) the transfer failed completely. There are two other "non-atomic" options (c) the bank deducted the transfer from account A and did not credit account B (thus you lost money) or (d) the bank credited account B but never debited account A (thus you gained money).

If you make a set of changes to many files you either want them all to appear or none of them to appear. Since CVS does each file change individually it is possible to get only parts of changes committed. GNU-Arch has the concept of a "changeset" which means that instead of viewing a commit as a series of changes to individual files you group the changes together.

4.1.3 Apache as a network server

CVS uses a separate program to listen for and respond to network requests. Subversion can do this but also allows network requests to be integrated into the Apache web server.

4.1.4 Branching and Tagging are cheap

Branching in CVS is painful. In fact you probably won't use branching until you happen to make a mistake in the commands and end up creating a branch. Branching is quite useful in larger projects. The idea of a branch is that you can work on a feature of a big program without affecting the main line of work. Once you have the new feature working you can *merge* it back onto the main branch and give it to the world.

4.1.5 Natively client/server

CVS is one big program. It does not distinguish between the client and the server except to handle certain commands. For coding purposes this makes it awkward to change only one side without affecting the other side.

Subversion splits the code between client and server. This is a feature for programmers but is unimportant for end-users.

4.1.6 Client/server protocol diffs both ways

The *diff* program will compare two files and write out a set of changes to make the first file change into the second file. Diff files are generally much smaller than the files they change.

CVS will send diff files from the server to the client so the net traffic is small and efficient. However it will send whole files from the client to the server. In many cases this can be very time consuming.

4.1.7 Costs are proportional to change size

If you make a small change you expect a cheap operation to remember the change. This is not always the case with CVS. Some small changes can result in very large file transfers.

4.1.8 Efficiently handling binaries

CVS generally assumes straight ascii text files. This is fine if you work on Unix style systems. But if you work on projects that include binary files (e.g. pictures or Word documents) CVS transfers the file unchanged.

Subversion has a special binary diff program that allows it to compare binaries and only send the changes.

4.1.9 Parseable output

This is more of a debugging and programmer feature than an end-user feature. Basically it means that the output is easier for other programs to read and handle which makes writing tools much easier.

4.2 Missing Features

Like all open source projects there is still more to do.

4.2.1 Symbolic Links

Sometimes projects use symbolic links to either ensure that only one copy of a file exists (so changes can be seen everywhere) and to put a copy of a file into a path without actually making another full copy. Java does this by making all of its commands (java, javac, etc) be links to one executable with different names. Neither CVS nor Subversion supports this.

4.2.2 Better merge support

Merging, like branching, is painful in CVS. The key problem is that there are usually a lot of changes on both the main branch of a project and the sub-branch. Merging is likely to find a lot of conflicts unless the sub-branch has tracked changes in the main tree carefully.

4.2.3 Broader WebDAV compatibility

WebDAV [13] is a standard for distributed authoring. It provides

> a set of methods, headers, request entity body formats and response entity body formats that provide operations for:
>
> Properties: The ability to create, remove, and query information about Web pages, such as their authors, creation dates, etc. Also, the ability to link pages of any media type to related pages.
>
> Collections: The ability to create sets of documents and to retrieve a hierarchical membership listing (like a directory listing in a file system).
>
> Locking: The ability to keep more than one person from working on a document at the same time. This prevents the "lost update problem" in which modifications are lost as first one author then another writes changes without merging the other author's changes.
>
> Namespace Operations: The ability to instruct the server to copy and move Web resources.

CVS does not implement WebDAV at all. Subversion only partially implements it.

4.2.4 Support for plug-in client side diffs

Instead of using the *diff* program on all files it would be useful if you could supply your own diff program. For instance, if you knew the contents of certain log files and knew that the dates inside the logs were not interesting you could write your own diff program to compare two log files. This program could then become a plugin for log files. Neither CVS nor Subversion implements this.

4.2.5 Internationalization

Neither CVS nor Subversion can handle international languages. This is a perfect example of where you can use your foreign language skills to contribute to a project.

4.2.6 Progressive multi-lingual support

CVS handles platform-dependent line endings (Unix uses Carriage-return, Windows uses Carriage-return Line-feed). Subversion would like to expand this special language/platform handling to international languages.

5 The GNU-Arch Solution

A guest presentation by James Blackwell.

6 Appendix A: CVS crib sheet

add add a new file or directory
admin Administration

annotate Show last revision where each line was modified

checkout Check out sources for editing

commit Check files into the repository

diff Show differences between revisions

edit Get ready to edit a watched file

editors See who is editing a watched file

export Export sources from CVS, similar to checkout

history Show status of files and users

import Import sources into CVS, using vendor branches

init Create a CVS repository if it doesn't exist

log Print out log information for files

login Prompts for password for authenticating server

logout Remove stored password for authenticating server

rdiff *patch* format diffs between releases

release Indicate that a Module is no longer in use

remove Remove an entry from the repository

rtag Add a symbolic tag to a module

status Display status information in a working directory

tag Add a symbolic tag to a checked out version of files

unedit Undo an edit command

update Bring work tree in sync with repository

version Display the version of CVS being used

watch Turn on/off/read-only checkouts of files

watchers See who is watching a file

References

[1] Orzech, Dan,
"Wall Street's Secret Affair With Linux",
"http://www.cioupdate.com/article.php/10493_1408471"

[2] Toronto Conference On Open Source Announced,
"http://slashdot.org"

[3] Open Source Conference,
"http://osconf.kmdi.utoronto.ca/"

[4] Improvements in kernel development from 2.4 to 2.6,
"http://www-106.ibm.com/developerworks/linux/library/l-dev26/index.html?_ca=dgr-lnxw01Linux24-26"

[5] BitMover,
"www.bitkeeper.com/index.html"

[6] What We Learned In The New Economy,
"http://www.fastcompany.com/magazine/80/neweconomy.html"

[7] Cederqvist, Per,
"Version Management with CVS",
"http://ftp.cvshome.org/release/archive/cvs-1.11.1/cederqvist-1.11.1p1.pdf"

[8] CVS software,
"http://www.cvshome.org/"

[9] Doctorow, Cory,
"Down and Out in the Magic Kingdom",
"http://www.craphound.com/"

[10] Slashdot,
"Subversion 1.0 Released",
"http://developers.slashdot.org/article.pl?sid=04/02/22/2344228"

[11] Subversion Project Home,
"http://subversion.tigris.org"

[12] Cervisia,
"http://www.kde.org/apps/cervisia"

[13] WebDAV,
"HTTP Extensions for Distributed Authoring – WEBDAV",
"http://www.ietf.org/rfc/rfc2518.txt"