Version Control With CVS

By icarus

Table of Contents

The High−Wire Act

Working on a complex software development project is often a lot like being a high−wire trapeze artist. There's the brain−busting tension, the nervous audience hoping that you won't fall, and the split−second coordination needed to achieve the stated goal. And, just to make matters worse, there's usually no safety net...

That's where CVS comes in. It's your safety net when you're doing the trapeze act...and it can save your neck if you fall.

An Elephant's Memory

CVS, or the Concurrent Versions System, is a very powerful tool in the arsenal of any developer or software programmer. It is a system which allows you to keep track of the software code you write, to maintain it in a logical manner, and to easily backtrack to previous versions or releases of the software. By storing your code in a CVS repository, you can easily mark specific points in development, log changes made over time, and extract a snapshot of a specific file as it looked six or eight months in the past.

In addition to keeping track of different software versions, CVS also helps to manage large, distributed software development projects (common to many open−source projects), in which developers located in different geographical locations collaborate to build a piece of software. In such situations, keeping track of changes made by individual developers to the overall body of code is almost impossible, and CVS provides an elegant solution to the problem by allowing each developer to work on copies of the original source and then merging the changes into the main code tree.

Finally, CVS also supports "code branches", which are essentially offshoots of the main code tree, usually initiated to fix bugs in older versions of the code. Since CVS comes with in−built mechanisms for version numbering and tagging, it provides developers with a simple way to spin off sections of the main code tree, yet still keep track of different versions and even merge the code branches back into the tree at a later date.

It should be noted at the outset itself, though, that CVS is merely a mechanism to manage different versions of your code. It does not help you write better code, provide you with deep and meaningful insights on software architecture, or assist you in building you a better thingamajig. The stability of your software code is entirely outside the boundaries of CVS' responsibility.

Think of CVS as an elephant, one with a long memory − if you're a good coder, it'll help you reminisce about the bee−yoo−tiful software you write in your younger days...and if you're a bad one, it'll make it hard for you to forget your mistakes.

If you don't already have CVS installed on your system, you can download it from http://www.cvshome.org/ Installation consists of the usual configure−make−make install cycle, or you could download a pre−compiled binary for your specific platform. The remainder of this tutorial assumes that you have CVS installed and functioning on your system.

Understanding The Repository

The first thing you need to do once you've got your copy of CVS installed is to set up a "repository". The repository is a central pool which contains the entire source code tree, and it is the fundamental building block of the CVS system. Once source code is committed to the repository, it comes under the aegis of the CVS revision control mechanism and can be tracked, released and otherwise manipulated via the command line.

In order to set up your repository, you need to identify a location that can store your files. A good choice (and the one recommended in the CVS manual) is

```
/usr/local/cvsroot
```

although you are free to select any location you wish.

A point to keep in mind here is that the location you select should have sufficient disk space to store your complete source tree, with a little left over besides. Also remember that if you have a bunch of developers all working on the same machine, each developer will need sufficient disk space to store his or her own working copy of the source tree (the option here is to use a CVS server and have your developers connect to it remotely to download the latest source tree).

Once you've decided on the location of the repository, you should tell the system about it by exporting it as an environment variable.

```
$ export CVSROOT=/usr/local/cvsroot
```

You should add this line to your startup scripts so that the $CVSROOT variable is set each time a user logs in to the system.

Finally, have CVS prepare the repository via the "cvs init" command, which looks like this:

```
$ cvs init
```

This will create the directory $CVSROOT/CVSROOT, which contains administrative information for the CVS system.

Checking It Out

Once the repository is prepared, it's time to begin adding source code to it. For example, let's suppose that you're doing some development on your Web site and have the following working directories set up in your home area:

```
|-- public_html
|   |-- community
|   |   |-- columns
|   |   |-- downloads
|   |   `-- images
|   |-- company
|   |   `-- images
|   `-- customers
```

In this case, it would be a good idea to have CVS track the entire "community" area of the site, since you expect to be doing quite a bit of work on it. You can import the entire section, together with all the files and sub−directories under it, into CVS with the "import" command.

```
$ cvs import −m "community" community mfre base N
community/index.php3 N community/functions.php3 N
community/post.php3 N community/read.php3 N
community/search.php3 No conflicts created by this import
```

Note that you need to run this command from the directory containing the files you wish to import. The −m tag provides a description for the imported files; this is followed by the name of the directory in the repository, a "vendor tag" containing information about the software vendor ["mfre"] and a "release tag" containing information about the current release of the software ["base"].

If you peek into your $CVSROOT, you'll see that it now contains a copy of the entire "community" directory tree. At this stage, it's a good idea to back up the original source code from your home area and then delete it from the system so that you do not accidentally work on the non−CVS version of your code.

The first time you import files into your source tree, CVS assigns them all a version number of 1.1. This version number is automatically incremented on a per−file basis as changes are made.

Remember that the way CVS is designed, once the code is sent, or "committed", to the repository, you, as a developer, don't get to work on the original source tree any more. If you need to make changes to the code, you need to "check out" the code from the repository into a separate working directory, make your modifications, and "commit" the revised code back to the central repository.

Each time you commit code to the repository, CVS will note the files which have changed, add revision numbers and ask you to enter a comment describing the changes, so that you can easily revert to any previous version of the file(s).

It's worth noting here that you'll probably encounter a few problems with file permissions for the various CVS administration files, as well as for the CVS directories that contain your source tree. The rule of thumb here is very simple: typically, all developers who will be working on the source tree need read and write access to the source tree directory in the repository. The easiest way to accomplish this is to create a new "group" with the

```
$ groupadd
```

command, and ensure that the source tree directory has the appropriate group ownerships.

Test Drive

Now that you know the principle, how about a little practice? Let's say that you've decided to make a small change to your community's welcome page by having the server display the current time and date in a corner. Well, the first thing you need to do is get yourself a copy of the latest source code from the repository.

```
$ cvs checkout community cvs checkout: Updating community U
community/functions.php3 U community/index.php3 U
community/post.php3 U community/read.php3 U
community/search.php3
```

The CVS "cvs checkout" command, sometimes abbreviated as "cvs co", allows you to "check out" a complete or partial copy of the current source tree to a working directory so that you can make changes to it. You'll see a directory called CVS/ in your working directory as well − don't worry about it.

Once the files are in your working directory, go ahead and make changes to them as you normally would − pop open a text editor, add the relevant code, save the file, test it, fix the twenty−four bugs that popped up, test it again and save it for the final time. But don't stop there − remember, this is *your* version of the source tree and the CVS repository has no knowledge of the changes you have just made. So tell it, by committing your revised code back to the repository with the "cvs commit" command (note that you'll be asked for a comment describing the changes you made):

```
$ cvs commit Added date() function to display time and date
CVS:
----------------------------------------------------------------------
CVS: Enter Log. Lines beginning with `CVS: ' are removed
automatically CVS: CVS: Committing in . CVS: CVS: Modified
Files: CVS: index.php3 CVS:
----------------------------------------------------------------------
Checking in index.php3;
/usr/local/cvsroot/community/index.php3,v
```

Again, problems checking out or committing code at this stage are typically the result of a glitch in system file permissions, and are usually simple to fix.

If you need to quickly refresh your working directory with the latest source tree, use the "cvs update" command:

```
$ cvs update
```

The "cvs update" command also works with single files − to get the latest version of the file "lib.php3", you could use:

```
$ cvs update lib.php3
```

The "cvs update" command typically prints a one−letter indicator next to each file − this could be

U file has been updated
M file has been modified
A file has been added to your working directory
R file has been removed from your working directory
C conflict detecten when trying to merge file into source tree
And you can check that you have the latest version of a specific file, or obtain detailed file information, with the "cvs status" command:

```
$ cvs status functions.php3
===================================================================
File: functions.php3 Status: Up−to−date Working revision:
1.1.1.1 Tue Nov 14 06:26:23 2000 Repository revision: 1.1.1.1
/usr/local/cvsroot/community/functions.php3,v Sticky Tag:
(none) Sticky Date: (none) Sticky Options: (none)
```

Once you've got your initial project going, you may often find it necessary to create new files in your working directory and add these to the existing source tree in the repository. CVS allows you to do this with the "cvs add" command.

```
$ cvs add sql.php3 cvs add: scheduling file `sql.php3' for
addition cvs add: use 'cvs commit' to add this file
permanently
```

The file "sql.php3" must exist prior to running this command. At this point, CVS knows about the file and will schedule it to be added to the source tree; however, to actually add it, you need to commit it with the "cvs commit" command.

And there's a corresponding command to delete files from the repository.

```
$ rm index.php3 $ cvs remove index.php3 cvs remove: scheduling
`index.php3' for removal cvs remove: use 'cvs commit' to
remove this file permanently
```

Playing The Numbers

As you already know, every file in the CVS repository is assigned a version number to assist in tracking its history. The initial version number for a file is 1.1, with the last digit being incremented by one each time the file is revised (the number on the left side of the decimal is sometimes referred to as the "major" revision number, while the number on the right is the "minor" revision number). When new files are added to the repository, they are assigned a major revision number equivalent to the highest major revision in the directory and a minor revision number of 1.

Since CVS changes revision numbers on a per−file basis, it's unlikely that all the files in your project will have the same revision number. If you'd like to stamp all the files with a specific revision number (commonly seen immediately prior to a software release), you can use the "cvs commit" command, like this:

```
$ cvs commit −r 2.0 cvs commit: Examining . cvs commit:
Committing . Checking in functions.php3;
/usr/local/cvsroot/community/functions.php3,v
```

Note, however, that the version number that you stamp on the files must be higher than the existing version number of the files in the directory.

Since CVS' revision−numbering mechanism is largely automatic, allowing the developer little control over the process, CVS offers a "tagging" system which assists developers in taking snapshots of the code tree at specific points, marking these snapshots with "tags" and then returning to the snapshot at any time by using the tag as reference.

For example, if you remember, you assigned the initial source tree a release tag of "base". Now, if you wanted to go back to square one and start off from the initial source tree, you could check out the "base" release like this:

```
$ cvs co −r base community
```

Or, if you're at release 3.8 of a piece of code, you could tag all the files at the point like this:

```
$ cvs tag release−38 cvs tag: Tagging . T functions.php3 T
index.php3 T post.php3 T read.php3 T search.php3
```

And, if at some time in the future you need to extract an image of the files at the time of "release−38", you can use:

```
$ cvs co −r release−38 community cvs checkout: Updating
community U community/functions.php3 U community/index.php3 U
community/post.php3 U community/read.php3 U
community/search.php3 U community/sql.php3
```

You can use the "cvs update" command to accomplish the same thing:

```
$ cvs update -r release-38 community
```

Branching Out

CVS also allows "code branches", which make it possible to spin off specific sections of the code tree and develop them independently. For example, let's suppose that you have released version 2.0 of the script file "index.php3" and are now proceeding to add new capabilities to your code in order to support release 3.0 of the community project.

As you work on "index.php3" in readiness for the new look of the site, CVS will continue to increment its version number. Let's suppose you're at version 2.8 when the webmaster requests you to add a personalized welcome message to the community page.

At this point, you have two options: you could ask him to wait for release 3.0, which will include this feature (not very customer−friendly), or you could add it immediately to the existing release 2.0 (since it's just a few additional lines of code). If you choose the latter option, you will need to instruct CVS to create a "branch" at the release point of version 2.0 and tag this branch with a new name.

```
$ cvs rtag −b −r release−20 release−20−personalized community
cvs rtag: Tagging community
```

Now that the branch is tagged, you can check it out with the tag name and begin work on it.

```
$ cvs co −r release−20−personalized community
```

Note that creating a branch does not affect the main code tree − you can still check out version 2.8 of your file whenever you're in the mood to work on the upcoming release.

At a later point, you also have the option of merging the branch back into the main code tree with the "−j branch" modifier to the "cvs commit" command. For example, to merge the branch in the example above into the current working copy of the source tree, you could use

```
$ cvs update −j release−20−personalized index.php3
```

Note, however, that merging files in this manner can result in conflicts between the different versions of the merged file − CVS will warn you if this happens.

You can obtain detailed information on the current status of each file in your working directory with the "cvs status" command, including tags, revision numbers and branch information.

```
$ cvs status
```

or

```
$ cvs status <filename>
```

for information on a specific file.

```
$ cvs status -v index.php3
===================================================================
File: index.php3 Status: Up-to-date Working revision: 2.0.2.1
Tue Nov 14 07:52:58 2000 Repository revision: 2.0.2.1
/usr/local/cvsroot/community/Attic/index.php3,v Sticky Tag:
release-20-personalized (branch: 2.0.2) Sticky Date: (none)
Sticky Options: (none) Existing Tags: release-20-personalized
(branch: 2.0.2) release-20 (revision: 2.0) base (revision:
1.1.1.1) mfre (branch: 1.1.1)
```

provides a list of tags currently active.

Miscellanea

There are a few other commands that might come in handy when working with CVS:

cvs log − displays a listing of the changes logged for each "commit" operation

cvs export − creates a working directory without the administrative CVS/ folder (primarily used when transferring the source tree from one location to another)

cvs diff − displays differences between files in working directory and files in repository.

If you're interested in using a CVS server and having developers connect to it to download the latest copy of the source tree, you should take a look at the CVS manual at http://www.cvshome.org/. Alternatively, if you don't like the command−line interface, there are a number of Web−based graphical interfaces to CVS available as well − take a look at http://www.cvshome.org/dev/addons.html for more information. Finally, if you're a Web developer, you'll find some very useful tips on managing your Web site scripts with CVS at http://www.durak.org/cvswebsites/

And that's it for the moment. Thanks for reading this far, and I'll see you soon!