

## *A Simple Primer on How to Use the Command cvs in a Project*

Craig C. Douglas  
September 4, 2005

### **Introduction**

The version control system *cvs* has been around for many years. It is used in many, many group projects and is readily available on many operating systems including all Linux-like systems, Mac OS X, and Windows (Cygwin-like systems). A newer system called *subversion* will eventually replace *cvs* in most open source projects. However, *cvs* is a very mature program and has many users who can help you if you get stuck.

Projects are maintained in *cvs repositories*, using the standard *cvs* jargon. A repository is a set of directories, each of which is considered a *project* at the top level of the directory hierarchy.

In projects that I give to students or colleagues, I always create a default project that has a *bin* and *doc* directories. In the *bin* directory are two important files:

<i>bin/env.csh</i>	Environment variables suitable for csh or tcsh
<i>bin/env.sh</i>	Environment variables suitable for sh, bash, or ksh

These files cannot be executed by a Linux (or Linux-like) shell program, but must be sourced or included by your shell program. In the *doc* directory are the following files:

<i>doc/cvs-primer.doc</i>	The Word document
<i>doc/cvs-primer.pdf</i>	The PDF version of the Word document

There are lots of places on the web to find better documentation and the O'Reilly book on *cvs* is a gem if you want to know how to do all sorts of things with a repository, particularly when you need debug it when subtle things go completely awry with obscure and inscrutable error messages.

I am going to assume throughout the rest of this document that you are using a Linux-like operating system (I am writing this using Word 2004 on Mac OS/X 10.4). I will use standard Linux-like notation and commands that require a terminal or command line window.

**Warning:** This is an informal document. It is intended to provide techniques that work on the day that I wrote this document versus being a formal document for all time. What I have documented here works for me, works for my colleagues, and works for some random strangers who take classes from me that I want to guarantee an always working environment for. I do not claim that what I describe in this document works anywhere else or for anyone else or at any given time (past, present, or future). The See No-*cvs*, Hear No-*cvs*, Speak No-*cvs* philosophy is the motto of this document.

### **A Sample Repository**

Consider the following definitions in *bin/env.csh*:

setenv	CVSROOT	www.foo.org:/u/cvsroot
setenv	CVSEEDITOR	vi

```
setenv CVS_RSH ssh
```

What does this gibberish mean? CVSROOT is the location on the Internet of the repository. In this case, it is on the machine `www.foo.org` in the directory `/u/cvsroot`. Every time you modify the project in the repository, you will have to document what you are doing using an editor. In this case, you are using the *vi* editor. Almost any editor will work fine, so feel free to use whatever one you like that can write a plain text file. Finally, you must connect to the repository's computer using a secure shell program so that passwords are encoded, not in clear text. There are many ssh implementations for all current operating systems (see <http://www.openssh.org> for a list).

Using `csh` or `tcsh`, you incorporate these definitions into your current environment using the command

```
source bin/env.csh
```

This can be typed by hand or the definitions can be put into your `~/.cshrc` or `~/.tcsh` file.

Using `sh`, `bash`, or `ksh`, you incorporate these definitions into your current environment using the command

```
. bin/env.sh
```

This can be typed by hand or the definitions can be put into your `~/.profile` file.

### **Checking Out a Project**

Suppose you are working on a collaborative project called *Uh-oh*. If you want to get a copy to work on, just type

```
cvs checkout Uh-oh
```

or

```
cvs co Uh-oh
```

You will have a new directory called *Uh-oh*. You do not have exclusive rights to the files. Anyone else with the correct permissions can check out your project and modify it at the same time.

### **Committing Changes for a Project**

Once you have modified, added, or deleted files and/or directories in a project, you can make the changes to the repository. Be careful, however. Just type

```
cvs commit
```

And `cvs` will walk through all of the subdirectories committing all changes that it knows about.

### **Updating Your Copy of a Project**

If you have already checked out a copy of project *Uh-oh* and you want to keep it up to date, just type

*cvs update*

in the Uh-oh directory. If you have added subdirectories, you can do an update on just one of the subdirectories (and all of its subdirectories) the same way.

It is a good idea to do an update as soon as you start working on a project again after any sort of break. If you are certain that you have not changed anything since you last committed changes, you should consider simply checking out the project again. Also, before committing things, it is a good idea to do a *cvs update* first in order to see if any conflicts exist, and fix them before trying to do an unsuccessful *cvs commit*.

### **Adding a File to Your Project**

This is easy. Create all of the files that you want to add in a directory. Then just type

*cvs add files*

Note that none of the files will actually be added to the repository until you do a *cvs commit*. Do not forget the *cvs commit* once you are convinced of the validity of your addition.

### **Adding a Directory to Your Project**

Directories are painful to add to a *cvs* repository, unfortunately. First you have to add the directory, then all of the files in it. First type

*cvs add dir\_name*  
*cvs commit*

Now add files to the new directory. In particular, it should not be empty. If it is, it will probably not be created when you update an existing copy, nor will it necessarily exist if you do a *cvs checkout*. I normally include a file in all of my directories that I create using the command

*date > .Created*

and add the *.Created* file to the project. (I know, this is ridiculous.)

### **Deleting a File from Your Project**

This one of the places that *cvs* shines brightly. You can delete a file, but it never actually goes away. You can get it back by backing out of commits. This is tricky, but can be done. Not that I ever have done this myself, but I have heard of lots of people who have... ☺ Before deleting a file, make certain that if it is modified that you commit the changes to the repository first. So, just type

*rm files*  
*cvs delete files*  
*cvs commit files*

and after the *cvs commit*, the files appear to have been deleted from the repository.

## Deleting a Directory from Your Project

You cannot except by tricking  *cvs* . Delete all of the files in the directory (as just described) and ignore the directory. If it is completely empty, the next time you checkout your project, the directory will not be there.

## Moving a File, but Not a Directory

This is really obscure until you think about it and see the simplicity  *cvs*  employs. Just type

```
 mv Old_file New_file  
 cvs delete Old_file  
 cvs add New_file  
 cvs commit -m "Move Old_file to New_file" Old_file New_file
```

Do not move directories. It usually requires changes to the repository itself.

## Backing Out of Changes

Buy the O'Reilly book on  *cvs*  if you really want to learn how to move forwards and backwards through changes with any real hope of being successful. You need to become an expert on branching, revisions, and history tracking. The  *info cvs*  command can help get you started, but the O'Reilly book is the best source of information. It even has some examples in it.

## Final Thoughts

There are many, many more options to  *cvs*  that I am not going to comment on. You can find more on any Linux-like system by typing the command

```
 info cvs 
```

and working your way through the menus. Hosting multiple projects is easy in a repository using the  *import*  option. Hosting multiple  *cvs repositories*  on a single machine is also easy using multiple definitions of  *env.csh*  and  *env.sh* .

The most important thing to remember is that you should communicate regularly with your colleagues who are working on the same project at the same time. Just once having to sort out a conflict (which someone does by hand comparing files a line at a time) will convince you never, ever to voluntarily do it again.

When problems unresolvable  *cvs*  problems occur, send e-mail to the poor person running the repository. I can assure you that that person can wait as long as it takes before hearing from you. Good luck.