A Gentle Introduction to CMSC311 labs and CVS
Or
How I learned to use CVS in CMSC311

William Arbaugh
September 2, 2004

This howto assumes that you already have scp and ssh installed on your computer. If you're using UNIX (Linux, FreeBSD, or OSX), then everything is already installed. If you're using Windows, then you'll have to first follow another howto to install scp and ssh.

There are several different version of scp available for Windows. Most are GUI based and I can't cover how to use each. Therefore, I'm only going to cover a command line version of scp which is the same for UNIX and Windows.

There are basically four steps you need to follow to complete a lab successfully:

1. Download the lab tar file from moodle to your local computer,
2. Move the lab tar file to the linux cluster via scp,
3. Edit the appropriate files to complete the lab while using CVS,
4. Move the file to be turned in and upload to moodle.

NOTE: I'm using the fictitious user account cmsc311999 in section 0101. You MUST use the account you were assigned in class and your appropriate section when following this howto. Also, all commands are shown in italics and begin with Prompt$.

Also, this HOWTO only deals with using CVS individually. If you were doing a project as a group using CVS, there are a few subtle differences- mostly copious use of cvs update.

Why use CVS on the Linux Cluster?

You're probably wondering why you should invest in learning CVS. There are two reasons basically. The first is that CVS is basically programmer's insurance. It creates a versioned back-up of your project so that you can "go back in time" to a version that was actually working. A important part of this strategy is that CVS repositories are usually on main servers which are backed up to tape on a frequent basis (in the case of the Linux cluster there is a nightly back-up done of your directory). This means that if the dog eats your laptop, you still have a copy of code some place and you don't have to spend the countless hours rewriting an entire assignment. The second reason why you should invest in learning CVS is that you'll get a 5% bonus on your lab if you use it! Finally, once you actually understand the cryptic syntax of CVS- it actually becomes easy and second nature. Then, you have the back-up when you need it.

Downloading the lab file from moodle

Downloading the lab tar file from moodle to your local machine is easy. Just click on the appropriate link in the Lab assignment and your browser will download the file to where ever it places your files (usually the desktop).

Once you've downloaded the tar file, you can choose to work on it on your local machine or upload it to the Linux cluster and work on it there. Right now, I'm only going to cover uploading the file to the cluster and working with it there. Later, I'll show you how you can still use CVS on the cluster and edit your files on your personal or local computer.

Uploading the lab file to the Linux cluster

To upload the lab file to the Linux cluster, you have to use scp or secure copy. To do this, you need to know two file name paths. The first is the location of the file you downloaded from moodle (this is the source file) and the second is the path to the directory on the Linux cluster where you want the file to end up (this is the destination directory). The destination directory is your home directory (or a sub-directory within your home directory). Now, you simply use scp on your own local computer as follows:

scp Lab1.tar.gz cmsc311999@linuxlab.csic.umd.edu:/fs/class/cmsc311/0101/cmsc311999

Now just ssh into the Linux cluster:

Prompt$ ssh –lcmsc311999 linuxlab.csic.umd.edu

Initializing CVS on the Linux Cluster

Now that we've ssh'd into the Linux cluster. The first thing we need to do (if you haven't already) is create the SSH repository. You do this with:

Prompt$ cvs –d /fs/class/cmsc311/0101/cmsc311999/CVS init

The above command creates and initializes the CVS repository under the "CVS" in your home directory.

You can now untar the lab tar file with:

Prompt$ tar xvfz Lab1.tar.gz
Lab1/
Lab1/bits.c
Lab1/bits.h
Lab1/btest.c
Lab1/btest.h
Lab1/datalab.pdf
Lab1/datalab.ps
Lab1/decl.c

Lab1/dlc
Lab1/Makefile
Lab1/README
Lab1/tests.c

which will create a Lab directory. Now, move into that directory:

Prompt$ cd Lab1

The first step now is to add the original lab files into the CVS repository. Before I do that, however, I want to set-up an environment variable so I don't have to type the pathname of the CVS repository all of the time. I do that under bash (tcsh and csh are slightly different) with:

Prompt$ export CVSROOT=/fs/class/cmsc311/0101/cmsc311999/CVS
Now, I can add my lab files to the repository with:

Prompt$ cvs import –m "Original lab files" Lab1 cmsc311999 start
N Lab1/bits.c
N Lab1/bits.h
N Lab1/btest.c
N Lab1/btest.h
N Lab1/datalab.pdf
N Lab1/datalab.ps
N Lab1/decl.c
N Lab1/dlc
N Lab1/Makefile
N Lab1/README
N Lab1/tests.c

No conflicts created by this import

This "import" command uses the log message "Original lab files" and creates a new project with the name *Lab1* within the CVS repository. The *cmsc311999* is known as the vendor tag (don't worry about it), and *start* is an initial tag (we'll talk about tags a bit later).

I now move up a level in the directory tree and rename the Lab1 directory temporarily to make sure my files were added to the repository correctly.

Prompt$ cd ..
Prompt$ mv Lab1 Lab1.orig

I can now checkout the Lab1 project from the repository.

Prompt$ cvs checkout Lab1
cvs checkout: Updating Lab1
U Lab1/Makefile
U Lab1/README
U Lab1/bits.c
U Lab1/bits.h
U Lab1/btest.c
U Lab1/btest.h
U Lab1/datalab.pdf
U Lab1/datalab.ps
U Lab1/decl.c
U Lab1/dlc
U Lab1/tests.c

Now, I move back into the new created Lab1 directory and verify that the files are there OK.

Prompt$ cd Lab1

If the files are OK, then I can delete the original files I with:

Prompt$ rm –rf ../Lab1.orig

Adding CVS Keywords to your file

OK. Time to start to edit the file you will turn in for grading, bits.c. Go ahead and use your favorite editor (Emacs suggested due to its tight integration with Make and GDB).

You want to add the following comment block at the top of the file bits.c.

```
/*
 * $Source$
 * $Revision$
 * $Date$
 * $Author$
 *
 * $Log$
 *
 */
```

The keywords (enclosed in dollar signs) will be expanded by CVS automagically each time you check out a file. Now, I can check the modified file back into CVS via process known as committing.

Committing changes to CVS

Committing the changes back to the CVS repository is easy. All I have to do is issue the following command:

Prompt$ Cvs commit –m "Added CVS keyword header"
cvs commit: Examining .
Checking in bits.c;
/afs/csic.umd.edu/users/waa/CVS/Lab1/bits.c,v <-- bits.c
new revision: 1.2; previous revision: 1.1
done

The "-m" switch tells cvs to use the string following as the log message. If you don't use the "-m" switch, cvs will automatically start an editor and ask you to enter a log message. The exact editor that is started is defined by the EDITOR environment variable.

If you don't understand the above paragraph, then ALWAYS use the "-m" switch with an appropriate log message.

If we now look at the beginning of bits.c now, we'll see that the keywords were expanded:

```
/*
 * $Source: /afs/csic.umd.edu/users/waa/CVS/Lab1/bits.c,v $
 * $Revision: 1.2 $
 * $Date: 2004/09/03 19:08:08 $
 * $Author: waa $
 *
 * $Log: bits.c,v $
 * Revision 1.2  2004/09/03 19:08:08  waa
 * Added CVS keyword header
 *
 *
 */
```

Now, I can edit my bits.c file and commit the changes each time I get a new function working.

Advanced Features of CVS

Tags

This lab is pretty easy because all you do is modify one file, bits.c. When you're working with a large number of files, tags become very handy.

Each file within CVS has a revision number which increases each time you check in the file. So in a large project, you may have a two files at revision 1.15 and one at 1.3.

Tagging allows you to mark each of these files so you can "recreate" your project down the road after you've made more changes to it. In a commercial setting, this is very useful when a software release is provided to a customer. The developers can tag the release for support reasons. For instance two months later (after significant changes to the project), the customer reports a bug. How do you recreate the customer's release and fix the bug? Tags! For you as a student, tags are useful for marking a project at specific milestones so you can revert back to a working version at a known point. This is most helpful when you've made a huge number of changes to a project that was working, and now it won't compile or has serious bugs.

You tag a project this way:

Prompt$ cvs –q tag Lab1-Function1-working
T Makefile
T README
T bits.c
T bits.h
T btest.c
T btest.h
T datalab.pdf
T datalab.ps
T decl.c
T dlc
T tests.c

If you have previously tagged a project, and you want to revert a single file to the tagged version you can do it this way:

Prompt$ cvs update –r Lab1-Function1-working bits.c

This would only update the bits.c file.

If you want to see what the changes are between a current file and the previously tagged version, you can do a diff:

Prompt$ cvs diff –c –r Lab1-Function1-working bits.c

Using a remote CVS repository

One of the nice features of CVS is that you can utilize a remote repository. That is- you can do development on your home machine (Windows, MacOS, Linux, whatever) and checkin and checkout projects just as if you were on the Linux cluster. Here's how on a UNIX machine, or on Windows using Cygwin.

First, you need to set the CVS_RSH environment variable to ssh using bash:

Prompt$ export CVS_RSH=ssh

You also need to set the CVSROOT environment variable to the remote repository:

Prompt$ export CVSROOT=linuxlab.csic.umd.edu:*/fs/class/cmsc311/0101/cmsc311999/CVS*

Now you can edit and commit, update, checkout etc. all from the comfort of you own machine, and the files will be stored on the repository on the Linux cluster (getting backed up every night).

If you decide to go this way, all you need to do then to test your project on the Linux cluster is to log into the cluster via ssh and checkout your project (assuming that you committed it when completed) and test!