Primer for using CVS at Milagro

Erik Blaufuss

August 6, 2002

Abstract

A document to help people get started with CVS.

1 Introduction to CVS

CVS is a program to help manage code. It allows many people to work on the same sets of code, as well as remote distribution of code.

This is more of a quick reference guide, but a excellent long writeup is available, see:
`gemini.umd.edu:/disk01/home/blaufuss/milagro/cvs/cvs.ps`
This file is also included with all Linux distributions. This document has many examples, and descriptions of all functions. For more detailed help, please see that manual. There are also man pages available.

1.1 How CVS works

CVS works by keeping a copy of all software in a special directory that only CVS changes. For our setup, this is an area on lepus called "cvs". The files here are in a special format, and should not be edited by hand, but feel free to look around and see what is there. This is known as the CVSROOT area. Currently there is just one area here: online, but it is possible to add the offline at some future point if desired. The online area is currently divided into a "server" subdirectory and a "archiver" subdirectory.

These pieces of the repository, called modules, are accessed relative to this directory. If you wanted to check out `/disk01/software/Milagro/cvs/online/server`, you would request `online/server`. There is also a file of aliases (see below) and this area is aliased to `server` or `kahuna`. Not all areas have aliases, sometimes you need to request files with the full filename.

The repository is accessed through a set of cvs commands that are used to check out, add, or update your local working copy of the software.

1.2 Getting access to the repository

You need to set a few enviroment variable so CVS know where to find the CVS server.

1. CVSROOT
   `setenv CVSROOT :ext:milagro@lepus.umd.edu:/disk01/software/Milagro/cvs`
   This variable tells CVS the location of the CVS server and the repository directory.

2. CVS_RSH
   `setenv CVS_RSH ssh`
   Tells CVS to use ssh to run remote commands. These remote commands will be run as milagro on lepus, so use the correct password.

3. CVS_SERVER
   `setenv CVS_SERVER 'env LOGNAME=username cvs'`
   Change username to your username, so that CVS log entries will be given proper credit or blame.

4. CVSEDITOR
   Defaults to vi, set to something else if you like. This one is optional.

That's all that is needed to use cvs. If you still can't access the repository, ask for some help. It is probably a simple permission problem.

1.3 What should and should not be in the repository

The only things that are needed to build the progras should be checked into the repository. This includes Makefiles, source files (.F, .c), and header files(.h, .fh).

It is possible to put binary files into the repository, but you shouldn't binaries created from source code in the repository.

2 A Summary of CVS commands

Here are some helpful points for using CVS commands: To get help for all these commands, you can type:
`cvs -H command`
This will also explain what other options are available for that command. Another thing to keep in mind is the recursive behaivor of CVS, if you request a directory, it will check it, and all the subdirectories out as well.

To obtain a listing of all possible CVS commands, use:
`cvs --help-commands`

2.1 Aliases in the modules file

There is a file that contains all the aliases for the repository. To get a listing of these aliases, use:
`cvs get -c`
You can also check out this file, and make changes to it:
`cvs get CVSROOT`

This will create a "CVSROOT" directory which contains a copy of the "modules" file. If you wish to add more, edit this file, then commit the changes.

2.2 Checking out stuff

To obtain files from the repository, you want to use the "get" command. This will create a directory, fill it with files, and put the directory you checked out under the control of CVS, so that you can commit changes, and update to the most current version at any time.

To check out a section of code, use:
`cvs get ''module''`
Where "module" is an alias, or a path relative to CVSROOT. For example, to check out the server stuff, you can use the alias (kahuna) or give the entire pathname, from CVSROOT (online/server).
`cvs get kahuna` or `cvs get online/server`

By defualt, the program will create a server directory, or a online/server directory, depending on how you asked for it, and it will populate it with the code from that module. A useful option is the -d option, this allows you to specify the name of the directory to put the checked out version into. See the help page (cvs -H get)

You are now free to make changes.

2.3 The update command

The update command is used to "update" a CVS controlled directory to the most current version. Usage:
`cvs update ''module''`
For example, if you wanted to update the server code that you checked out last month, then, in the directory above the source:
`cvs update server` or `cvs update online/server`
depending on where it is. If you are in the "server" directory, you can type:
`cvs update .`

2.4 Commiting changes.

When you have finished changing a source code, and want to check these changes back into the repository, you must use the commit command. This commits the changes and updates the repository, so everyone else can take advantage of these changes.

To commit:
`cvs commit files...`

For example, in the online/server/devel area, you had to make a change to the Makefile, and Worker.c, and you are ready to commit these changes. Then, type, in the online/server/devel directory:
`cvs commit Makefile Worker.c`
This will open your favorite editor (from CVSEDITOR enviroment variable)

and request you enter a log file entry. This is a required step, you must enter something. It is best to put a short explanation of the changes that were made to the software, so others can follow you changes months later. After the editor is done, it will check it into the repository and report if there are any conflicts.

The best way to avoid conficts it to make sure you have the most up to date version of the software when you start making changes, and commit the changes as quickly as possible. CVS is smart, and it will try to merge the changes of different people, but only if they are on different parts of the code. If there is any direct conflict, it will tell you there is, and ask you to resolve them. See the cvs manual for more details about this.

2.5 Release, and deleting

When you are finished working on a module, you can simply delete the directory, with out hurting anything, but any changes will be lost.

If you want to check to make sure that there are no changes being lost, then you can use the "release" command, which will check to make sure there are no changes being lost, and then delete the directory. Use:
`cvs release -d module`

2.6 Adding new stuff

When you want to add new stuff to an exisiting module, or add a new subtree to a directory, then "add" is the command.

2.6.1 Adding a new file to an existing module

When you have a directory under CVS control, and you want to add some new files in that directory, you must use the "add" command followed by the "commit" command.

For example, if you added some new files, `newstuff.c` and `newstuff.h` to the checked out "server", from our earlier example, then to check this into the repository, you would:
`cvs add newstuff.c newstuff.h`
The add command only reserves a space in the repository, checking to make sure there are no files with that name already in that module. To actually put it into the repository, you must now commit:
`cvs commit newstuff.c newstuff.h`
It will open an editor and ask for a message, then it will add it to the repostory.

2.6.2 Adding a new subdirectory to the repository

If you wanted to start a new subdirectory in the repository, for a new project, this is also possible. You should already have a tree that is under CVS control. For example, if you wanted to add something to the "server" module, then you

should have checked it out already, or check it out before starting.
`cvs get kahuna`
This will create the subdirectories, and fill then with whatever is already in the "server" module.

Next go into the "server" subdirectory, and create a new subdirectory (with the mkdir command). Fill this with the files that you want to add. (You can actually start your project this way, then things are ready to commit when you finish.) Remember, that you only want to add and commit the files that are going to be added to the repository (Do not `cvs commit *`). When the sources are ready, it is time to add and commit.

For example, you made a subdirectory `server/newstuff` and in that directory is a Makefile, newstuff.c and newstuff.h. It is possible there is an executable and .o files, you do not want to commit these!

To add and commit these, in the "programs" directory:
`cvs add newstuff`
This checks to make sure that the directory name you have chosen is not used already.

In the newstuff directory, you also must "add" the Makefile, and the sources:
```
cd newstuff
cvs add Makefile newstuff.c newstuff.h
```
Now you are ready to commit:
`cvs commit Makefile newstuff.c newstuff.h`
It will again open an editor and ask for a log entry and then check the new stuff into the repository.

2.7 Other commands

There are many other commands, but since this is only an introduction, I will leave it to the manual to explain the rest. You can use cvs to compare you copy of a program to the repository, obtain old versions of a piece of code, obtain a history of changes...and many other things. Please see the long writeup for this information.

3 Help

Of you need help with any of this, feel free to email me:
`blaufuss@umdgrb.umd.edu`.