Version Control with CVS

Casey Devet, Peter Beshai, Abraham Dubrisingh

Spring 2014

1 Version Control with CVS

When working on projects with other programmers, it becomes necessary to use some sort of a system to organize all the code that is written. For this purpose, version control systems were created. Amongst other things, version control systems store a repository of source files, giving you a central location to store code that several people may be working on. They also store a history of previous versions of code, allowing you to look back in time to examine what you have changed (and thus, help locate bugs).

CVS (Concurrent Versions System) is a version control system that we will discuss in this tutorial. While you are not required to use CVS, we highly recommend that you use some sort of version control when working on the project.

1.1 CVS Structure

With CVS, there is a central location where all source code is stored, called the *CVS Repository.* This location is not intended to be directly modified by the programmer; it should be modified through use of calls to different CVS commands.

When working on a project which is being managed by CVS, each programmer will have their own copy of the source code, which is obtained by *checking out* a copy from the repository. This allows each user to work on their own problems without affecting each other, until they have solved them.

After implementing something new, and ensuring that it does not break everything, it is a wise idea to *commit* your code to the central repository with your changes. Then, when any other programmer looks to see if changes have been made to the central repository, they will receive your modifications.

1.2 Repository

The CVS repository contains the master copy of the code you are writing, and must be placed in a location where all programmers working on the project can access it. In the case of the project, you will ensure that the CVS repository has the proper group permissions set on it.

To inform CVS of where the repository you wish to access is, you will need to set your `CVSROOT` environment variable. If your CVS repository was located at `$HOME/cvsrepo`, in `csh`, you type:

```
setenv CVSROOT $HOME/cvsrepo
```

If your shell client is `bash`, you would type:

```
export CVSROOT=$HOME/cvsrepo
```

Once this is set, all calls to the program `cvs` will use the path `$CVSROOT` to find the CVS repository.

1.3 Updating Code

command: `cvs update`

It is a wise idea to ensure that before you begin making any changes to your checked out copy of the project, you update your code to the most recent checked in version. Otherwise, you will run into all kinds of sticky situations where both you and your partner have made significant changes to the same file, and you will have to manually merge them back together.

This happens because CVS does not lock any files when you check out your project. So, if your partner commits his changes to `Deck.cc`, and you have also edited `Deck.cc` locally, when you try to update the code you are working with from the CVS repository, there will be a merging conflict. CVS will notify you of these conflicts and will attempt to merge the solutions together. If merging fails, you will need to go and manually inspect the file in question, fix the merge, and re-commit it.

After the merging conflict, the file `Deck.cc` may contain chunks that look like the following:

```
void shuffle() {
<<<<<<< Deck.cc
    foo();
=======
    bar();
>>>>>>> 1.2
```

This indicates that in your local copy, you had written `foo()` in the function `shuffle`, while your partner had written `bar()`.

When you run an update, the legend on the left-hand side of the files being updated will tell you what is happening:

Symbol	**Meaning**
A	File has been added to the CVS repository, but not yet committed
C	File has uncommitted changes, and a conflict occurred during merge
M	File has uncommitted changes
P	File has been updated (Remote CVS only)
R	File has been deleted, but the deletion has yet to be committed
U	File has been updated
?	File is unknown to CVS

1.4 Committing Code

command: `cvs commit`

Once you have made changes to your local copy of the project, and you are sure that the changes you have made have not broken anything, you should commit your code to the CVS repository.

1.5 Exporting For Submission

command: `cvs export`

When you are ready to submit your file for marks, it is a good idea to first export your project to a different directory, then zip and submit that. This will ensure that you have the most recent copy from the CVS repository, and also that you are not submitting any CVS files. The command you enter to ensure that you have the most recent copy is: `cvs export -DNOW <project name>`.

1.6 How To

This section goes into an example of how to use CVS.

1.6.1 Setting up a CVS Repository

1. Create a directory to hold the repository: `mkdir $HOME/cvsrepo`
2. Set your `CVSROOT` environment variable: `setenv CVSROOT $HOME/cvsrepo`
3. Initialize the repository: `cvs init`
4. Change to the directory that has the source code you would like to import: `cd $HOME/cs246/A3`
5. Import the source code into the repository: `cvs import -m "Initial import of Euchre" euchre pbeshai euchre-1`

1.6.2 Sharing Your Repository

After completing the steps above, you should ensure that your group members can read from the repository, so that they can get their own working copies of the code within. Note that only *one* member needs to create a repository.

If user `pbeshai` created the CVS repository, then he would need to do:

1. `chgrp -R cs246_74 $CVSROOT`
2. `chmod -R g+r $CVSROOT`
3. ``chmod g+wxs `find $CVSROOT -type d` ``
4. `chmod g+w $CVSROOT/CVSROOT/history $CVSROOT/CVSROOT/val-tags`
5. `chmod o+x $HOME $CVSROOT`

N.B. We must set world execute permissions since `/u/pbeshai ($HOME)` is not in the group `cs246_74`, and cannot be changed to be in that group.

The other members of `pbeshai`'s group will only need to set their `CVSROOT` environment variable properly: `setenv CVSROOT /u/pbeshai/cvsrepo`.

1.6.3 Obtaining a Working Copy

Once the repository is set up, you should not continue to edit the code in the directory from which you made the initial import. Instead, you should check out the copy that is in the CVS repository. This copy will contain information CVS needs to keep track of changes, and to know which project you are working on. To do this for the above example, you would type `cvs checkout euchre`, and a subdirectory of the current directory named `euchre` would be created containing the code that you initially imported.

1.6.4 Adding the Repository

To add a new file to the repository, you can use `cvs import` (as shown above) or use the `cvs add` command as shown below:

```
cvs add <file-to-be-added>
```

After using the `cvs add` the file is not yet added to the repository. You must commit your working copy to the repository for the newly added files to be placed in the repository.

1.6.5 Committing to the Repository

When you have finished implementing something new and you would like to add the new code to the repository, you type: `cvs commit -m "Some meaningful log message"` in the directory that you would like to submit the updated code from. If you would only like to commit a specific file or directory, you can specify it as an argument: `cvs commit -m "Some meaningful log message" Player.h Player.cc`.

1.6.6 Updating Your Working Copy

Your partner has committed some code to the repository, so to ensure that you do not cause any conflicts with his changes, you want to update your working copy to contain his changes. To do this, you type `cvs update` in the directory which you want updated. Similarly to commit, if you only want to update a specific file, then you can specify it as an argument.

1.6.7 Remotely Access CVS

To remotely access your CVS repository, you will need to configure two environment variables on the computer from which you would like to access it:

- `CVS_RSH`: ssh
- `CVSROOT`: user@machine:/path/to/CVS/repo

In the above example, using `csh`, we would type:

```
setenv CVS_RSH ssh
setenv CVSROOT pbeshai@student.cs.uwaterloo.ca:/u/pbeshai/cvsrepo
```

After setting those variables, all CVS commands you do will work with your remote CVS repository.

2 Additional Resources

The following resources were used in the creation of this document, and may be useful to refer to for additional information:

1. CVS - Open Source Version Control `http://www.nongnu.org/cvs/`
2. CS 350: Using CVS `http://www.student.cs.uwaterloo.ca/~cs350/common/cvs.html`