1 A Brief Overview of CVS

CVS is a simple revision control system. It allows individuals to check out working copies, make changes (even concurrently), and then commit these changes. In addition to the most recent version (or "head revision"), CVS keeps all previous versions of every file (including removed files). You can retrieve any version of a file, and even revert files to their previous state.

CVS keeps all of its files in a repository. This is a directory (or a whole hierarchy of directories) in which cvs stores modules. A module is a directory (or entire directory tree) which CVS considers to be a single entity. CVS also keeps "control files" in this repository. These control files include files which tell CVS where files are kept, what to do when someone commits files or checks out files. CVS also keeps a log of all changes in the repository.

2 Using CVS

The basic CVS command has the following form:

cvs cvsoptions command commandoptions

Several of the basic CVS commands are described in section 3. A full list of commands can be found by executing **cvs –help-commands**. There are two environment variables which are used by CVS.

CVSROOT This variable points to the CVS repository.

CVSEDITOR This variable tells CVS which editor it should use for recording log messages. If this variable is not set, then it will use the EDITOR environment variable.

There are also a few important options which are shared by all CVS commands.

-H Displays usage information for a command

-e editor Use 'editor' for editing log messages.

A complete list of cvs options can be found by executing **cvs –help-options**.

3 The Basic Commands

3.1 **cvs checkout** (a.k.a. cvs co or cvs get)

This command gets a fresh working copy out from the repository. You should modify this copy, and then commit (see 3.2) the changes. There is one important thing to remember when using **cvs checkout**. Whenever you check out a working copy, at least one directory will be created. For example, if you are checking out a module called *uml*, and you execute the command **cvs checkout uml**, a directory called "uml" will be created in the current directory.

3.2 **cvs commit** (a.k.a. cvs ci or cvs com)

This command commits changes made in your working copy to the repository. This makes your changes available to all other developers working on the module. Executing the command with no arguments will commit all changes made. If you want to commit only specific files, you can specify those specific files on the command line. This command will open your editor (see 2) and ask you to put in a message describing what was changed.

3.3 cvs update

cvs update brings your working copy up to date with the most recent revision in the repository. For each file in your repository, **cvs update** will replace the out of date file with the most recent copy. If you have made changes to a file, and another person has made changes to the same file, and committed them, then **cvs update** will "merge" those changes into your current copy. If the two of you edited the same sections of code (or if so much has changed that **cvs update** cannot figure out where the changes should go), then you will get a conflict. In this case, cvs will warn you of the condition, and put *both* versions of the code into the file. You should then go through and manually merge the changes.

When using **cvs update**, typically you want to supply the **-d** option, which tells cvs to get any new directories which have been created since you checked out your code. If you do not supply this option, you will not get new directories.

3.4 cvs add

This command adds a file to the repository. When you create a new file or directory, you should **cvs add** the file so that it a copy will now be kept in the cvs repository. The **cvs add** must be committed to permanently add the file to the repository.

3.5 cvs release

This command releases your working copy of the code. The typical usage is **cvs release -d** *directory*, where *directory* is your top level cvs directory. This is equivalent to running **rm -rf directory**, except it has a few added bonuses. One is that it properly fixes up the bookkeeping in the cvs repository, so that cvs knows who has what files checked out. The other use is that it will list all files which haven't been added to the repository. Typically this includes only .o files and other temporary files created while working on the file. However, occasionally you discover that you forgot to **cvs add** a file to the repository. This is a good sanity check to make sure that never happens.