Getting started with CVS

Stephen A. Edwards, Columbia University

A version control system such as CVS or RCS provides a mechanism for managing multiple revisions of files, most often program source files, but any type of files can be used. Although version control systems were primarily designed to handle multiple developers on a single software project, I personally use CVS to track changes to every program, presentation, proposal, and paper I write. I also use CVS's ability to work remotely to keep versions of these files up-to-date on multiple machines.

1 Creating a repository

The first step in using CVS is to create a repository: a directory that will contain the revision history of every file along with other meta-data. Many people and projects can use the same repository. I suggest giving the directory a long name such as "repository" and putting it in your home directory:

```
% cd
% cvs -d ~/repository init
% ls repository
CVSROOT/
%
```

2 Adding an empty subdirectory to the repository

A repository starts out empty; the first thing to put into it is an empty directory that you will later fill with your files. Use the cvs "import" command to do this (import can bring a whole directory tree into the repository; we aren't using that feature here). Let's create a directory called "project1" (you can name it as you like).

```
% mkdir ~/project1
% cd ~/project1
% cvs -d ~/repository import \
   -m "Initial Version" project1 myself one
% ls ~/repository
CVSROOT/  project1/
%
```

"myself" and "one" are the vendor and release tags respectively. Their values are normally irrelevant, but you must list them here.4 The `-m` option specifies a log message; if you omit it, you'll be asked to add a message in a text editor.

3 Creating a working directory

Once one member of your development group has set up a repository and put at least one subdirectory in it, you don't need to repeat the process; you can start from this step.

Files stored in the repository contain version information; you shouldn't edit them directly. Instead, set up a different directory where you "check out" the master files to edit. I like to call this directory "cvs," also in my home directory. Create this directory and check out the directory we created in the last step:

```
% mkdir ~/cvs
% cd ~/cvs
% cvs -d ~/repository checkout project1
cvs checkout: Updating project1
% ls
project1/
% ls project1
CVS/
%
```

4 Adding files

Once you have a working directory, you can add files and directories to it and place them in the repository.

```
% cd ~/cvs/project1
% cat > Hello.java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello.");
  }
}
% cvs add Hello.java
cvs add: scheduling file 'Hello.java' for addition
cvs add: use 'cvs commit' to add this file permanently
%
```

At this point, we've told CVS that we want to add the Hello.java file to the repository, but we haven't actually done it yet. It's only when a file is *committed* that it actually enters the repository:

```
% cvs commit -m "Initial version" Hello.java
RCS file: /home/sedwards/repository/project1/Hello.java,v
done
Checking in Hello.java;
/home/sedwards/repository/project1/Hello.java,v
   <-- Hello.java
initial revision: 1.1
done
% ls ~/repository/project1
Hello.java,v
%
```

One a file has been added and committed, anybody who checks out or updates the same directory from the repository will get the new version of the file. For example, say someone created another working directory:

```
% mkdir ~/othercvs
cd ~/othercvs
% cvs -d ~/repository checkout project1
cvs checkout: Updating project1
U project1/Hello.java
% ls
project1/
% ls project1
Hello.java
%
```

5 Modifying files and getting the changes

If someone modified the Hello.java file in the "othercvs" working directory and committed the changes, this would place a different version in the repository.

```
% cd ~/othercvs/project1
% cat > Hello.java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World.");
  }
}
% cvs commit -m "Added World" Hello.java
Checking in Hello.java;
/home/sedwards/repository/project1/Hello.java,v
   <-- Hello.java
new revision: 1.2; previous revision: 1.1
done
%
```

The new version is now in the repository, but not in the first working directory. Use "cvs update" to get the newest versions:

```
% cd ~/cvs/project1
% cat Hello.java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello.");
  }
}
% cvs update
cvs update: Updating .
U Hello.java
% cat Hello.java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World.");
  }
}
%
```

6 Adding and updating directories

Directories are added just like normal files; CVS is smart enough to know the difference:

```
% cd ~/cvs/project1
% mkdir tests
% cvs add tests
Directory /home/sedwards/repository/project1/tests
  added to the repository
%
```

What's nice is that you don't have to run cvs commit after adding a directory.

Unfortunately (or fortunately), to get a copy of an added directory, you can't simply run cvs update, since by itself it only updates *files* in the current directory. You need to give update the `-d` flag:

```
% cd ~/othercvs/project1
% cvs update
cvs update: Updating .
% ls
CVS/  Hello.java
% cvs update -d
cvs update: Updating .
cvs update: Updating tests
% ls
CVS/  Hello.java  tests/
%
```

7 Adding binary files

Most of the time, you'll be adding text source files to your repository (in particular, don't add a generated file such as a `.class` or `.o` file to the repository, it will just confuse things), but occasionally you'll want to add a binary file such as a `.jpeg` or `.pdf` file. So that CVS's version storage system doesn't get confused (it normally only stores diffs), it's best to tell CVS a file is binary with the `-kb` flag when you first add it to the repository:

```
% cd ~/cvs/project1
% cvs add -kb foo.pdf
% cvs commit -m "Initial version"
cvs commit: Examining .
cvs commit: Examining tests
RCS file: /home/sedwards/repository/project1/foo.pdf,v
done
Checking in foo.pdf;
/home/sedwards/repository/project1/foo.pdf,v  <--  foo.pdf
initial revision: 1.1
done
%
```

(Note that you don't always need to specify the name of the file you're committing; if you don't specify one, CVS will commit all added or modified files.)

8 Removing files

Sometimes, a file has outlived its useful life and needs to be removed from the repository. The remove command does this, but only after you've removed the file from your working directory:

```
% cd ~/cvs/project1
% rm foo.pdf
% cvs remove foo.pdf
cvs remove: scheduling 'foo.pdf' for removal
cvs remove: use 'cvs commit' to remove this file permanently
% cvs commit -m "didn't like foo.pdf"
cvs commit: Examining .
cvs commit: Examining tests
Removing foo.pdf;
/home/sedwards/repository/project1/foo.pdf,v  <--  foo.pdf
new revision: delete; previous revision: 1.1
done
%
```