

# GUIDE TO VERSION CONTROL WITH CVS

This is an idiot's guide to using CVS. Of course, proper documentation exists if you want to plough through that...

## Availability

CVS is GNU software, and so can probably be found in lots of places. One such place is <http://www.hensa.ac.uk/ftp/mirrors/gnu/cvs-1.9.tar.gz>. The version we have at DL is:

```
> cvs -v
```

```
Concurrent Versions System (CVS) 1.9 (client/server)
```

```
Copyright (c) 1993-1994 Brian Berliner  
Copyright (c) 1993-1994 david d `zoo' zuhn  
Copyright (c) 1992, Brian Berliner and Jeff Polk  
Copyright (c) 1989-1992, Brian Berliner
```

CVS may be copied only under the terms of the GNU General Public License, a copy of which can be found with the CVS distribution kit.

CVS is a front end to the program RCS, which you will need to install first. This is also GNU software, available e.g. from <http://www.hensa.ac.uk/ftp/mirrors/gnu/rcs-5.7.tar.gz>.

## Overview

CVS is a front end to the RCS revision control system. The basic idea is that CVS manages the revisions made to a given set of files, for example those associated with a project. A reference copy of the files is kept ("repository" version) which consists of the current revision plus a series of diffs to previous revisions. You work as normal with a "working" version making edits etc., and when you are happy with the changes you have made, they are "checked into" CVS. This means that the repository version is updated together with a log message clarifying the change.

CVS has many features, and RCS has yet more. We certainly don't use all these features, but we find the following useful:

- You can check the working copy of a file against the repository version. If they are different, then someone else is working on the file and some cooperation is required.
- A side effect of this is that if edits to a file are left unfinished, then it is obvious later when you discover that the file is not up-to-date with the repository version. This leads to greater discipline when making changes: i.e. finish off one set of changes before starting the next!
- A descriptive log is kept of all changes, together with who and when. This is often enough to answer questions such as "when was this change made?". If it is not enough, you can create diffs between any two revisions to look at the actual changes. This is very useful if you have several people working on lots of files, as in our case.
- Old revisions can be restored. We do this occasionally if we regret a change.
- Revisions can be tagged. For example, before a release, we tag all current revisions as "release-x\_y". Tags can be used to deal with files en masse: we use this to actually generate the release. One advantage of this is that it separates official CVS'd files from other junk which may be lying around.

These features allow several people to work on a single set of files. This is infinitely preferable to having several sets which rapidly get out of step. [Actually, you can have several sets of working files checked against a single repository version, but we don't work this way.]

## How to set up a CVS'd area

I'm not sure if this is the official method, but it worked for me ...

### 1. Initialise repository:

In the directory which you want to contain the repository (which I'll assume to be `/public/xtal`), type

```
mkdir CVSROOT
mkdir CVSROOT/CVSROOT
touch CVSROOT/CVSROOT/logininfo
cvs -d /public/xtal/CVSROOT init
```

This initialises a set of administrative files in `/public/xtal/CVSROOT/CVSROOT`. I'm sure the first 3 steps shouldn't be necessary, but it complains that `logininfo` doesn't exist otherwise. Step 4 requires an explicit path.

### 2. Create repository version:

Now go to the top level directory of the set of files you want to CVS, which in this example I'll call `progdir`. Type:

```
cvs import -m "log message" /public/xtal/CVSROOT/progdir "vendortag"
"releasetag"
```

This will create the repository version in the directory `/public/xtal/CVSROOT/progdir`. The quoted arguments are messages and labels which you can change to suit yourself.

### 3. Create working version:

You can now create one or more working versions from this repository version. [You ought to be able to create one over the top of the original set of files, but I had problems with this.] Go to the directory where you want a working version, and type:

```
cvs -d /public/xtal/CVSROOT checkout progdir
```

This will create a copy of the repository version which you can work with. You can use all the CVS commands on this working version, see below. You can add and remove individual files using the `cvs add` and `cvs remove` commands.

## Useful Commands

The following are the most useful commands, see CVS documentation for more details:

`cvs status <file>`

Does the working version agree with the repository version?

`cvs log <file>`

Print the log of all changes to the file.

`cvs commit <file>`

Save the changes made to the working version to the repository version. It will prompt for a log message, or use the `-m` switch.

`cvs diff <file>`

Do a diff between the working version and the repository version.

`cvs diff -r1.7 <file>`

Do a diff between the working version and revision 1.7 of the repository.

`cvs diff -r1.7 -r1.6 <file>`

Do a diff between revisions 1.7 and 1.6 of the repository.

`cvs add <file>`

Add a new file to the repository. You will need to "cvs commit" it afterwards.

`cvs remove <file>`

Remove a file from the repository. You will need to delete the working version first and "cvs commit" it afterwards.

## Other help

[http://www.loria.fr/~molli/cvs/doc/cvs\\_toc.html](http://www.loria.fr/~molli/cvs/doc/cvs_toc.html)

---

*Martyn Winn: m.d.winn@dl.ac.uk*

Last modified: Thu Dec 4 16:44:37 GMT 1997