



Published on [The O'Reilly Network](http://www.oreillynet.com/) (<http://www.oreillynet.com/>)  
[http://www.oreillynet.com/pub/a/linux/2002/01/03/cvs\\_intro.html](http://www.oreillynet.com/pub/a/linux/2002/01/03/cvs_intro.html)  
[See this](#) if you're having trouble printing code examples

## Introduction to CVS

by [Jennifer Vesperman](#)

01/03/2002

This is the first in a two-part series on CVS. This article is intended for folks who will be using CVS already installed on a system. In it, the author explains check-out, update, adding, merging, and other functions. In the second part, scheduled to run later this month, she will show how to create and manage a CVS repository, for those who need to start from scratch. -- Ed.

CVS -- or Concurrent Versioning System -- is a system for managing simultaneous development of files. It is in common use in large programming projects, and is also useful to system administrators, technical writers, and anyone who needs to manage files.

CVS stores files in a central repository, set (using standard Unix permissions) to be accessible to all users of the files. Commands are given to "check out" a copy of a file for development, and "commit" changes back to the repository. It also scans the files as they are moved to and from the repository, to prevent one person's work from overwriting another's.

This system ensures that a history of the file is retained, which is extremely useful when the boss decides he wants a feature you trashed months ago. It also ensures that backing up the repository is enough to backup a project (providing all necessary files are kept in repository).

CVS is usually used to help manage projects, but can also be used for individual files.

### Typical Uses

CVS is designed for developers, either individually or in teams. For individuals, CVS provides a repository from which you can work from home, the office, or the client site without having to haul disks around. It also provides version control, allowing rollbacks without loss of data. For teams, it also keeps a record of who changed which lines of a file and prevents direct overwriting of each other's work.

System administrators can keep configuration files in CVS. You can make a change, `cvsv commit` it, test it. If it fails, roll back the change -- even if you only discover the failure six months down the track.

Administrators can keep a CVS tree of the configurations for server farms. Adding a new server? Just `cvsv checkout` the config tree for that type of server. Committing all changes also helps you keep track of who did what, when.

CVS is useful for writers. I keep these articles in a CVS tree -- if I lose my local copy, I have an automatic backup. It is also useful if I'm collaborating, or if I discover that I need to retrieve a section I'd removed.

### Notes For the Examples

The examples in this article assume that there is a pre-existing CVS repository for the project.

The next article in this series will be "CVS Administration," which will explain how to create a CVS repository, branch projects, and manage the repository. For now, check `cv`s `init` and `cv`s `import` in the `cv`s man page.

## Necessary Environment Variables

If you expect to use one repository, set the `$CVSROOT` environment variable to the repository's full path, in the format `:protocol:user@host:path`. It should look something

like `:ext:jenn@cv`s.`example.com.au:/home/cv`s.

For security, set the `$CVS_RSH` environment variable to `ssh`. By default, CVS uses `rsh` to access a repository on a remote machine.

## Getting the Tree: cvs checkout

CVS stores the files in a central repository, but users work from a working copy of a file.

- Make a directory to do your work in. I tend to use `~/cv`s.
- `cd` into that directory.
- `cv`s `checkout module`, e.g. `cv`s `checkout example`. (`module` is CVS' name for a related collection of files.)

If you don't have a `$CVSROOT` environment variable, or want to use a different repository, use `cv`s `-d repository-path` to checkout the module instead.

The checkout will put a copy of that module's files and subdirectories into your `cv`s directory.

```
cv
```

s\$ ls
example
cvs\$ cd example; ls
CVS src
cvs/example\$ cd CVS; ls
Entries Repository Root

The `cv`s directory is a special directory CVS uses for its own purposes. `CVS/Entries` lists files and subdirectories CVS knows about. `CVS/Repository` contains the path to the corresponding directory in the repository. `CVS/Root` contains the path to the repository, so you won't need to use the `-d repository-path` option again for these files.

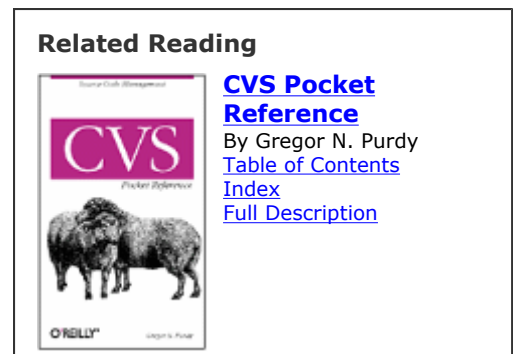
Note that `CVS/Root` overrides the `$CVSROOT` environment variable, so if you change the repository, you should check out the module again.

```
cv
```

s/src\$ ls
CVS Makefile sample.h sample.c

The `src` directory contains the source files for the example project. `sample.c`, `sample.h`, and `Makefile` are ordinary files in the working copy -- in the repository, they are stored in a format which tracks the changes.

## Receiving Changes: cvs update



Every day before you start work, and any time someone else may have made and committed changes, `cd` into your working directory and run `cv update`. This checks your working copies against the repository files and imports any changed files for you. `cv update -d` also gives you any new directories.

Update reports on the status of each file as it checks it:

- U *file***  
updated successfully
- A *file***  
added but not yet committed (need to run a `cv commit`)
- R *file***  
removed but not yet committed (need to run a `cv commit`)
- M *file***  
modified in your working directory: the file in the repository was changed and your working directory file was older than the last time CVS checked it OR the repository had changes which the system could safely merge
- C *file***  
there was a conflict between the repository copy and your copy which requires human intervention
- ? *file***  
the file is in your working directory but not the repository and CVS doesn't know what to do with it

### When Changes Conflict: Merging Files

If CVS can't merge a modified file successfully with the copy in the repository, it announces the conflict in the output of `cv update`. The original file is stored in `.#file.version` in the file's working directory, and the results of the merge are stored as the original filename.

```
cv/example$ cv update
jenn@cvs.example.com.au's password:
cv server: Updating .
RCS file: /home/cvs/example/sample.c,v
retrieving revision 1.3
retrieving revision 1.4
Merging differences between 1.3 and 1.4 into sample.c
rcsmerge: warning: conflicts during merge
cv server: conflicts found in sample.c
C sample.c
```

CVS writes the merge with the conflicting lines surrounded by CVS tags. CVS can't automatically merge conflicts where the same line is changed in both versions of a file.

```
<<<<<< sample.c
Deliberately creating a conflict.
=====
Let's make a conflict.
>>>>>> 1.4
```

### Making Changes: `cv commit`

Once your files are checked out, edit them and compile them normally. Apply the updates to the repository with `cv commit`. This command needs to be run from higher in the hierarchy than all the files you have changed -- you can run it from the base of your working copy.

You can also `cv commit filename`, which will commit a single file or recursively commit a directory.

Different project teams have different opinions on how often to `cv commit`. Good rules of thumb include "every time you have a clean compile," and "every day before lunch and before you leave."

```
cv/example$ cv commit
cv commit: Examining .
cv commit: Examining src
jenn@cv.sample.com.au's password:
```

CVS examines each directory and subdirectory below the current working directory. Any files that CVS knows about will be checked for changes. If your `cv` repository is not on the local machine, CVS will ask for a password for the remote machine.

CVS then opens whichever editor is the default in your environment -- based on the `$CVSEEDITOR` or `$EDITOR` environment variables. Add change-notes for the appropriate files.

```
CVS:-----
CVS: Enter Log.  Lines beginning with 'CVS:' are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS:  example/src/sample.h example/src/sample.c
CVS:-----
```

I strongly recommend meaningful change-notes -- if you're trying to do a rollback and all you have are messages which say "fixed a few bugs," you'll not know which version to roll back to without using `cv diff`.

If there is a potential conflict, `cv commit` fails. Correct this by running a `cv update` on the repository -- CVS will attempt to merge the files, and will ask for human help if it cannot do this without losing data.

```
cv server: Up-to-date check failed for 'cv_intro.html'
cv [server aborted]: correct above errors first!
cv commit: saving log message in /tmp/cvst7onmJ
```

### **Making New Files: `cv add`**

Files CVS doesn't know what to do with are reported with a question mark after the commit process and during a `cv update`. They need to be added to the repository before CVS will recognize them.

Use `cv add filename` to mark a new file for inclusion. CVS doesn't put the file in the repository until you do a `cv commit`.

Directories are added with the same command. Files within a directory can't be added until the directory is added.

### **Removing Files: `cv remove`**

To mark a file for removal from the working copies, use `cv remove filename`. Before CVS will remove a file from the repository, you have to actually delete it from the filesystem. CVS doesn't actually remove the file entirely, it puts it in a special subdirectory in the repository called `Attic`.

## Directories Don't Remove

CVS does not remove directories -- it would break the change tracking. Directories can be removed by changing the repository -- this is discussed in "CVS Administration."

## Final Words

CVS is useful as a tool for concurrent development, maintaining file histories, and providing a central database for files of any sort. It's well worth looking into and playing with.

## Further Reading

- `man cvs`
- `man 5 cvs`
- `info cvs` has a good section on "what CVS is" and "what CVS is not." It's also a useful expansion on the manual. The "Repository" section discusses protocols.
- From the Big Scary Daemons column: [BSD Tricks: CVS](#)
- Sourceforge has several articles on CVS. See sections 6 and 7 on the [Sourceforge Site Docs](#) page.

*[Jennifer Vesperman](#) is the author of *Essential CVS*. She writes for the O'Reilly Network, the Linux Documentation Project, and occasionally *Linux.Com*.*

---

Return to the [Linux DevCenter](#).