

CVS Administration

Contributed by Aman

CVS--or Concurrent Versioning System--is a tool for saving the user's butt by maintaining a history of changes. It allows for the retrieval of older versions of files, records who makes each change, and prevents simultaneous changes from overwriting each other.

CVS expects the files it maintains to be stored in modules-- directories of related files and subdirectories. A module can be as small as a single directory containing a single file, or as large as the amount of disk space you have available.

Companion Article

This article is a companion to Introduction to CVS, which covers the use of a CVS repository. This article covers building and maintaining the repository, and discusses some of the most common issues faced by CVS administrators.

Creating a Repository

The repository needs to be hosted on a machine with sufficient disk space to store all your files and all the data for the changes you expect. As a rule of thumb, put the repository on a partition with enough room for three times the expected final size of the module. Then use the "Scotty principle," and double your estimate-- the project will expand. If you intend to store binary files, multiply by ten. After your first project, you'll have a feel for how much space to allow.

Ensure that all users have access to the repository machine from all the machines they intend to use.

Create your repository root directory. Repositories are often stored in /home/cvsroot or /usr/local/cvsroot. Use cvs init to set up the directory as a CVS repository.

```
cvs -d /home/cvsroot init
```

Debian Linux has a script, cvs-makerepos, which will build a repository based on pre-existing Debian configuration scripts. See man cvs-makerepos for more information and man cvsconfig for an automated system for configuring a Debian CVS repository.

Importing a New Module

Before loading your project into CVS, consider its structure. Moving or renaming files and directories can damage the CVS record of the files' history. Deleting a directory can cost you the record of all its files and subdirectories. For this reason, CVS has no facility for moving or renaming files and directories, or removing directories.

Make your initial directory structure--even if it's just one directory. Add any initial files you want. From within the root directory of your project, use the command cvs -d cvs repository import nameofmodule vendortag releasetag.

For most cases, you will not need to know about vendor tags and release tags. CVS requires them to be present, but you can simply use the name of the module as the vendor tag, and the current version as the release tag.

```
/home/jenn$ cd example
```

```
/home/jenn/example$ cvs -d /home/cvsroot import example example_project ver_0-1
```

Using Tags

Tags assign a symbolic name to a revision of a file or a set of files. cvs tag tags the repository versions of all the files in the current working directory and its subdirectories.

The cvs rtag command is based on a timestamp. It assigns the symbolic name to the version of the file or directory that is closest to and older than the timestamp; it does not look at the working directory.

Note that CVS doesn't allow the '.' character in tags.

```
cvs tag tagname filename
```

```
cvs tag tagname
```

(Tags the repository versions of all files in the current working directory.)

```
cvs tag -c tagname
```

(Aborts if the repository copy differs from the working copy.)

Example:

```
cvs/example$ cvs tag release-1-0 src/sample.c
```

```
cvs/example/src$ cvs tag release-1-0
cvs/example/src$ cvs tag -c release-1-0
```

To retrieve a tagged version, use the -r flag to checkout or update. If you checkout or update in your usual working directory, the tagged version will overwrite the existing files.

```
cvs checkout -r tagname
cvs update -r tagname
```

Example:

```
cvs$ mkdir example-rel-1.0
cvs/example-rel-1.0$ cvs checkout -r release-1-0
```

or

```
cvs/example$ cvs update -r release-1-0
```

Branching Development

If you need to fix a bug in an older version of your code without changing current code, or modify a configuration set for staging servers without modifying the set for your production servers, you might need to branch your modules. A branch allows storage and retrieval of a variation of the main module, without affecting the main module. Changes on the branch can be merged in later.

To make a branch:

```
cvs tag -b branchtag
```

Example:

```
cvs/example$ cvs tag -b release-1-0-patches
```

Retrieve a branch using either checkout or update. Checkout will create a new directory for the branch, and update will overwrite your current working directory with the branch.

```
cvs checkout -r branchtag
cvs update -r branchtag
```

Example:

```
cvs/example-rel-1.0$ cvs checkout -r release-1-0-patches
cvs/example$ cvs update -r release-1-0-patches
```

Branches can be merged back into the main trunk, using the conflict resolution system invoked by cvs update and cvs commit.

```
cvs checkout module
cvs update -j branchtag
```

Example:

```
/tmp/example$ cvs checkout example
/tmp/example$ cvs update -j release-1-0-patches
```

Or in a single command:

```
cvs checkout -j branchtag module
```

Example:

```
/tmp/example$ cvs checkout -j release-1-0-patches example
```

Resolve any conflicts the system reports, then cvs commit.

Removing Directories

Directories cannot be removed from the repository using CVS commands. If a directory is no longer required, empty the directory with `cvs remove`, and use `cvs update -P` and `cvs checkout -P` when retrieving a working copy. The `-P` flag ensures that empty directories are not retrieved.

If you must, you can remove a directory by using `rmdir` on the repository. Do this on a copy of the repository first and check that you aren't breaking anything: If the directory in the repository has an 'Attic' subdirectory, you will lose archived copies of files formerly stored there.

If you remove a directory from the repository, you should have all your users remove their existing working copies, and check out fresh copies of the module.

Watching and Locking Files

Unlike many versioning systems, CVS doesn't have file locking--it doesn't prevent simultaneous editing of files. However, you can set files to be watched, so CVS will mail watchers when a file is being edited. If files are being watched, developers need to use `cvs edit` and `cvs unedit` to release a file for editing. Unwatched files can be edited without notifying CVS in any way.

To set up files for being watched, use:

```
cvs watch on (files)
cvs watch off (files)
```

To set yourself as a watcher, use:

```
cvs watch add (files)
cvs watch remove (files)
```

or

```
cvs watch add -a edit|unedit|commit|all (files)
cvs watch remove -a edit|unedit|commit|all (files)
```

The special CVS file `notify` determines what occurs when a watched file is changed. It defaults to sending mail to the user's username, on the CVS server. If your users have other addresses, set up the file "users" in the repository's CVSROOT directory. Entries should be in the format `user:email`, one to a line.

```
jenn:jenn@cvs.example.com.au
```

Keeping CVS Secure Remote Repositories

If the repository is on the local machine, both access and security are fairly straightforward. You can set the `$CVSROOT` environment variable to the root directory of the CVS repository, or call `checkout` with the `-d <directory>` option.

If the repository is on a remote machine, it is necessary to tell CVS which machine it is on, and what method will be used to access the machine. There are several methods available, but for security and simplicity I prefer to use SSH. The syntax for defining a remote `$CVSROOT` is `:method:[[:user]:[:password]@]hostname[:[:port]]:/path/to/repository`; for example, `:ext:jenn@cvs.example.com.au:/usr/local/cvsroot`

(Note that my info cvs disagrees slightly with what my copy of CVS actually does. I have included the syntax that works for me--a colon between the host and the path. Use the syntax that works on your system.)

To use SSH, we use the `:ext:` method. This method uses an external-to-CVS `rsh` or `rsh-compatible` program to communicate with the CVS server. To tell CVS to use SSH instead of `rsh`, set the environment variable `$CVS_RSH` to `SSH`. Ensure that SSH is set up on the server and on all clients, that SSH keys are generated, and that users have usernames and passwords on both machines. If the usernames are the same, the `user@` part of the CVSROOT string is not necessary. If a standard SSH port is used, the port is not necessary.

```
cvs -d :ext:cvs.example.com.au:/usr/local/cvsroot checkout sample
```

Permissions

The files in the repository are all read-only. Permissions to those files shouldn't be changed. To control access, use the directory permissions. Most administrators make a group for the people who should have access to the module, and ensure that the group has write access for the directory.

If using a remote repository, set the root directory of the module setgid to ensure that all directories beneath it are made with the correct permissions. If using a local repository, \$CVSUMASK can be set to control the permissions of files and directories in the repository.

Developer Machines

Securing the project involves securing the repository and securing all checked out copies--typically your developer's machines. It's not enough to ensure that the repository is safe and all transmissions are properly encrypted if someone can walk into your developer's office on his day off and burn a CD of your code. Maintain the usual physical and Net-based security for your development machines, prototype and demonstration copies, and any other places the code gets checked out to.

Final Words

Managing a CVS repository can seem like an extra task for an overburdened administrator, but the repository can save the day when the client decides that they do need that abandoned feature after all, or when a minor change to a configuration file has unforeseen side effects--three weeks later.

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 alternate protocols in more depth.
- * From the Big Scary Demons column: BSD Tricks: CVS
- * Sourceforge has several articles on CVS. See sections 6 and 7 on the Sourceforge Site Docs page.