

Version Control and Source Code Management



Essential

CVS

O'REILLY®

Jennifer Vesperman

Essential CVS

Other resources from O'Reilly

oreilly.com *oreilly.com* is more than a complete catalog of O'Reilly books. You'll also find links to news, events, articles, weblogs, sample chapters, and code examples.



oreillynet.com is the essential portal for developers interested in open and emerging technologies, including new platforms, programming languages, and operating systems.

Conferences O'Reilly & Associates bring diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit *conferences.oreilly.com* for our upcoming events.



Safari Bookshelf (*safari.oreilly.com*) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today with a free trial.

Essential CVS

Jennifer Vesperman

CVS Quickstart Guide

To help you get up to speed quickly using CVS, this chapter explains the most common CVS operations. The commands and examples in this chapter are based on standard situations and cover only the most common options. Future chapters go further in depth on each topic covered in this chapter.

The examples and instructions in this chapter are based on the Unix/Linux command-line CVS client. Most graphical clients use the CVS command names for their menu options and buttons, so if you're using a graphical client you should be able to follow this chapter reasonably easily. Graphical clients and clients for operating systems other than Unix/Linux are described in Appendix A.

You may not need to read all of this chapter; follow these guidelines:

- If you're working on an existing project that is already stored in CVS, skip the early sections and start at "Checking Out Files."
- If CVS is already installed and running, with a repository available for your project, go straight to "Importing Projects."

If you're not sure whether CVS is already installed and running, read the first part of "Installing CVS"; it tells you how to check. If you're uncertain about having a repository, try searching for the directory *CVSROOT*. The repository root is then the directory that *CVSROOT* is in. The other directories in the top level of the repository are CVS projects.

Installing CVS

CVS is client/server software that runs on Unix and Linux platforms. When you install CVS on a Unix/Linux server, you automatically get both server and client software. To access CVS across the network from any Unix/Linux machine, simply install CVS on the machine in question. The server and client software are one and the same.

CVS is available from <http://www.cvsui.org>. It is also available as an installation package from many GNU/Linux distributions, including Debian, Red Hat, and SuSE.

If you prefer GUI clients, I recommend a visit to <http://www.wincvs.org>. There you'll find gCVS, WinCVS, and MacCVS, which are GUI clients for Unix and GNU/Linux, Windows, and Macintosh (pre-OS X), respectively.



If you are a Macintosh user running OS X, you can install the standard Unix CVS server and client.

A Windows-compatible CVS server is available at <http://www.cvsnt.org>. This server is not identical with the Unix server, but the differences are clearly listed in the FAQ and an installation guide is available on the web site.

If you plan to use the *Secure Shell* (SSH) protocol to establish secure connections between CVS clients and CVS servers, you'll need to install compatible versions of SSH on your client and server machines. For clients, you may need to find a version of SSH that can be used from the command line. See "Accessing Remote Repositories" for more information on this topic.

The OpenSSH web site (<http://www.openssh.com>) is a good starting point for information on SSH and SSH clients. You may also want to read *SSH, The Secure Shell: The Definitive Guide* (O'Reilly) by Daniel J. Barrett, Ph.D. and Richard Silverman. For more than you ever wanted to know, check the SSH FAQ at <http://www.employees.org/~satch/ssh/faq/>. The Google list of SSH documentation is at http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Documentation/.

Most Unix and Linux systems have an SSH client installed as part of the standard set of programs. Mac OS X also has an SSH client preinstalled. If it is not installed automatically, an SSH client is usually available in your distribution as an optional program.

The web site for MacCVS has a useful article on SSH for Macintosh users at <http://www.heilancoo.net/MacCVSClient/MacCVSClientDoc/ssh.html>. The instructions are useful regardless of which Macintosh CVS client you use.

To find an SSH client for Windows, I usually start from OpenSSH's web site on Windows and Macintosh clients (<http://www.openssh.com/windows.html>).

If you are running Unix or Linux, you may already have CVS installed. If it's installed and in your path, typing *cvs* at the command line produces the results shown in Example 2-1.

Example 2-1. CVS help display

```
$ cvs
Usage: cvs [cvs-options] command [command-options-and-arguments]
  where cvs-options are -q, -n, etc.
  (specify --help-options for a list of options)
  where command is add, admin, etc.
  (specify --help-commands for a list of commands
   or --help-synonyms for a list of command synonyms)
```

Example 2-1. CVS help display (continued)

where `command-options-and-arguments` depend on the specific command (specify `-H` followed by a command name for command-specific help) Specify `--help` to receive this message

The Concurrent Versions System (CVS) is a tool for version control. For CVS updates and additional information, see the CVS home page at <http://www.cvshome.org/> or Pascal Molli's CVS site at <http://www.loria.fr/~molli/cvs-index.html>

If you already have CVS installed, you can skip this section and jump ahead to “Building Your First Repository.”

To install CVS on Unix, you typically download the source code and compile it on your system. This approach works with Linux too, but many Linux distributions also provide you with the option of installing precompiled CVS binaries. You'll find CVS installation instructions for various Linux distributions following the next section on installing from source.

Installing and Building CVS From Source

Download the compressed `.tar` file from <http://www.cvshome.org>. Decompress and unarchive the file. If you intend to keep the source after you compile it, unzip the compressed `.tar` file into `/usr/src/cvs`. Otherwise, you can decompress and unarchive it into `/tmp`. Next, `cd` into the `cvs` directory, read the `INSTALL` and `README` files, and run the commands there. I recommend that novice users disable `automake` and `autoconf`.



You should compile CVS as a user without superuser privileges, but you must have superuser privileges when installing CVS (running `make install`).

Example 2-2 shows a sample installation from source. In this example, I decompress and unarchive the `.tar` file into `/tmp` and change directory into the top level of the resulting source tree. I then disable `automake` and `autoconf` using the `noautomake` script provided with CVS, as described in the `INSTALL` file. The next steps described in `INSTALL` for CVS 1.11.5 are to run the `configure` script provided and then, if that exits successfully, run `make`, switch to the root user, and run `make install`.

Example 2-2. Installing from source

```
/tmp$ ls
cvs-1.11.5.tar.gz
/tmp$ gunzip cvs-1.11.5.tar.gz
/tmp$ tar -xpf cvs-1.11.5.tar
/tmp$ cd cvs-1.11.5
/tmp/cvs-1.11.5$ ./noautomake.sh --noautoconf
/tmp/cvs-1.11.5$ ./configure
creating cache ./config.cache
```

Example 2-2. Installing from source (continued)

```
checking for a BSD compatible install... /usr/bin/install -c
.
.
.
creating config.h
config.status: executing depfiles commands
/tmp/cvs-1.11.5$ make
make all-recursive
make[1]: Entering directory `/tmp/cvs-1.11.5'
.
.
.
make[2]: Leaving directory `/tmp/cvs-1.11.5'
make[1]: Leaving directory `/tmp/cvs-1.11.5'
/tmp/cvs-1.11.5$ su root
Password:
/tmp/cvs-1.11.5$ make install
Making install in lib
make[1]: Entering directory `/tmp/cvs-1.11.5/lib'
.
.
.
make[2]: Leaving directory `/tmp/cvs-1.11.5'
make[1]: Leaving directory `/tmp/cvs-1.11.5'
/tmp/cvs-1.11.5$
```

Using apt to Install CVS

apt is the package manager used in Debian Linux and distributions based on Debian. Ensure that your *apt* source list is set up, and run *apt-get update* before installing CVS. The CVS package is called *cvs*, so the installation command is:

```
apt-get install cvs
```

Example 2-3 shows a sample *apt* installation, omitting most of the results of the *apt-get update* command.

Example 2-3. Installation using apt

```
$ apt-get update
.
.
.
Reading Package Lists... Done
Building Dependency Tree... Done
$ apt-get install cvs
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  cvs
0 packages upgraded, 1 newly installed, 0 to remove and 10 not upgraded.
Need to get 0B/1085kB of archives. After unpacking 2626kB will be used.
```

Example 2-3. Installation using apt (continued)

```
Reading changelogs... Done
Preconfiguring packages ...
Selecting previously deselected package cvs.
(Reading database ... 39545 files and directories currently installed.)
Unpacking cvs (from .../cvs_1.11.1p1debian-8_i386.deb) ...
Setting up cvs (1.11.1p1debian-8) ...
```

Using rpm to Install CVS

rpm is the package manager used by Red Hat Linux. CVS is on the second CD-ROM of the Red Hat 7.3 distribution. These instructions assume that the CD is inserted and mounted or the package is available to the package manager some other way.

If you are using *rpm* from the command line, check your CD for the name of the CVS package. It will be in the format *cvs-version.rpm*. To install CVS, use the command:

```
rpm -Uh cvs-version.rpm
```

If the command is successful, there will be no output. If you prefer progress bars, add *-v* (for verbose) to the arguments. Example 2-4 shows a command-line installation in verbose mode.

Example 2-4. Installing with rpm

```
$ rpm -Uvh cvs-1.11.1p1-7.i386.rpm
Preparing...          ##### [100%]
1:cvs                 ##### [100%]
```

If you are running Gnome, you will probably use *GnoRPM*:

1. Select Install. This opens a new window and loads the packages from the CD-ROM.
2. Open the *Packages* folder, then the *Development* and *Tools* subfolders.
3. Select the *cvs-version* package (currently *cvs-1.11.1p1-7*).
4. Click on Install.

Figure 2-1 shows a *GnoRPM* example.

If you are running KDE, you will probably use *KPackage*:

1. Select the New tab. The packages should load from the CD-ROM.
2. Find and select the *cvs* package.
3. Click on Install. A new window opens; click on Install in that window to confirm the installation.

Figure 2-2 shows a *KPackage* example.

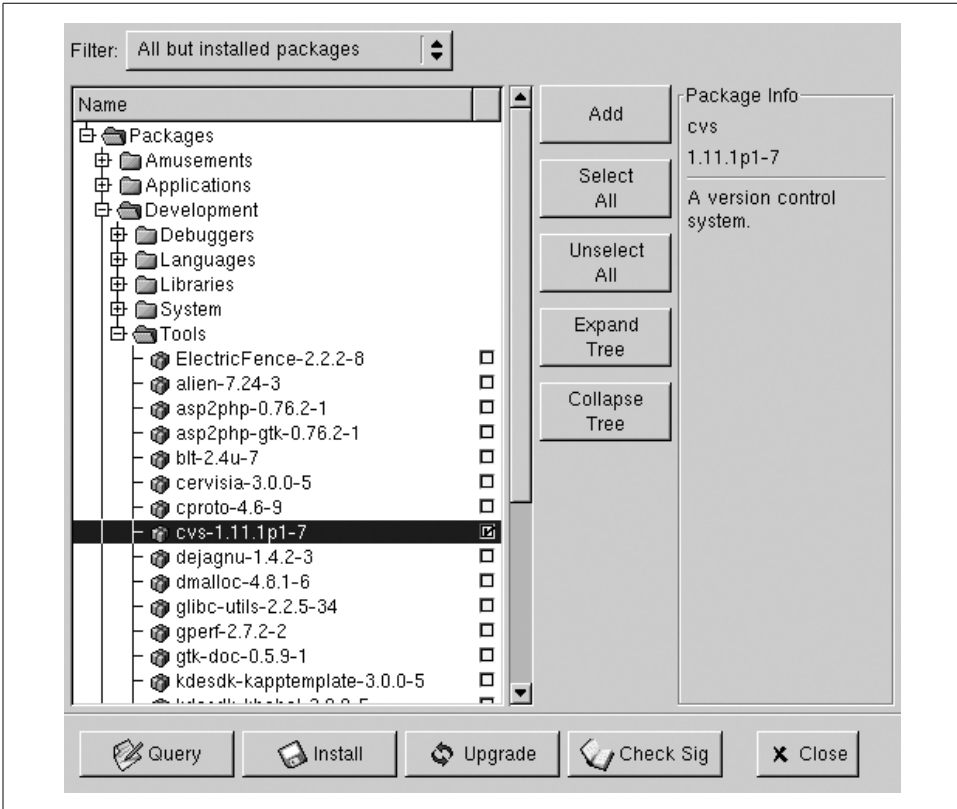


Figure 2-1. GnoRPM installation

Using yast to Install CVS

yast is the package manager used in SuSE Linux. CVS is on the second CD-ROM of SuSE 7.3, but in later versions of SuSE it may be on a different CD-ROM or in a different group than in the following list. Put the CD-ROM in the drive; then run the *YaST2* control center:

1. Select Software and open Install/Remove software.
2. Select the group Development/Version Control.
3. Select *cvs* and click OK.
4. If the CD-ROM is not in the drive and mounted, YaST2 will ask for it.

Figure 2-3 shows an example of a YaST2 installation.

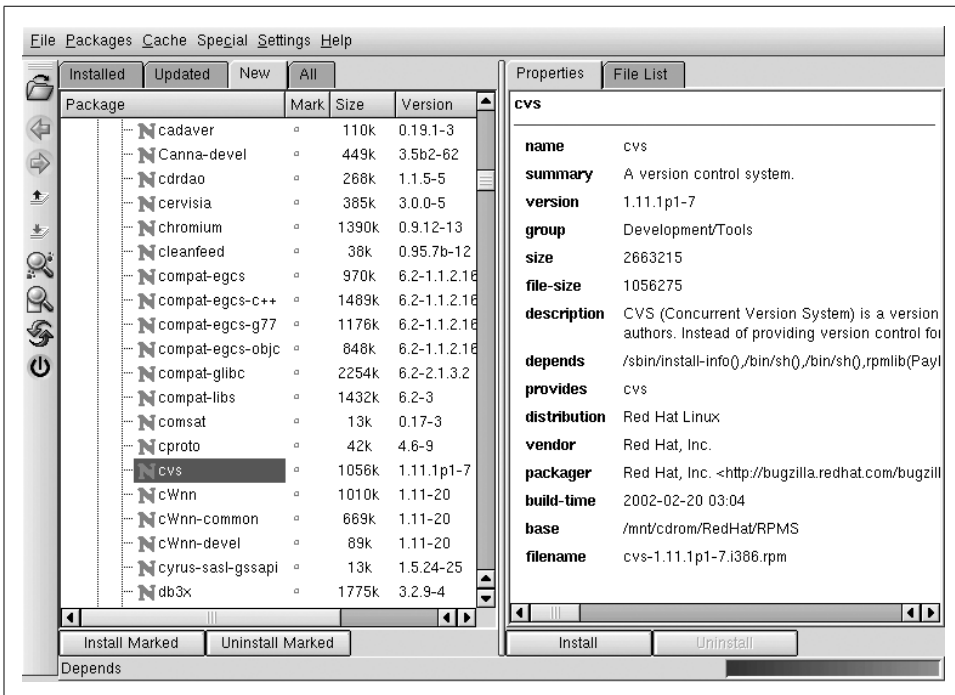


Figure 2-2. Kpackage installation

Building Your First Repository

CVS relies on a file-based database called the *CVS repository*. The repository needs to be hosted on a machine with sufficient disk space to store your files and all the change data for your projects. The repository should be accessible to all the users from their workstations.

If the repository is on a remote computer, I recommend that users access the repository via SSH. Ensure that the server is running an SSH server and the workstations have a compatible SSH client. For more information on remote repositories and SSH, see “Accessing Remote Repositories” later in this chapter. Chapter 8 contains a full discussion of remote repositories.

For any one project, ensure there is enough disk space for three times the expected final size of that project. If you intend to store binary files, multiply by at least five. After your first project, you’ll have a feel for how much space to allow.

A repository can store many projects. If you are creating a repository that may be required to handle several projects of unknown size, estimate what you can and ensure you can add more room later.

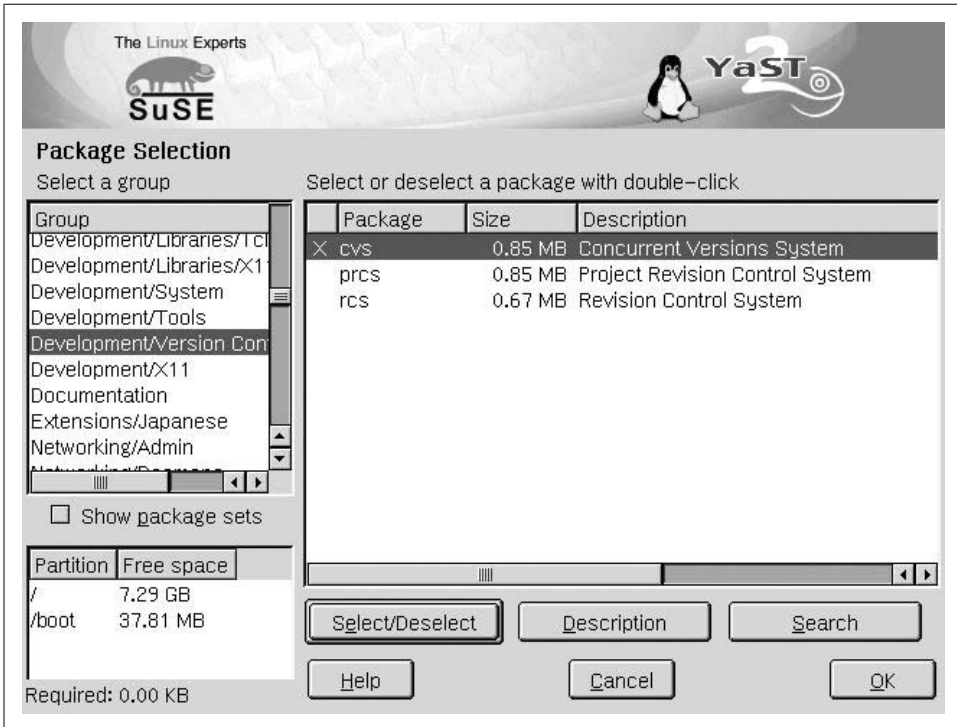


Figure 2-3. Yast2 installation

A CVS repository is user data, and it should be on a partition that is backed up and won't shut down the machine if it gets full. Repositories are often stored in `/cvroot`, `/var/lib/cvroot`, `/home/cvroot`, or `/usr/local/cvroot`. According to the Filesystem Hierarchy Standard (available at <http://www.pathname.com/fhs/>), the preferred location for a repository is `/var/lib/cvroot`.



Another possibility is to put the repository in `/cvroot`. This is easiest for users to remember.

Example 2-5 illustrates the steps involved in creating a repository. First, create the repository root directory on the CVS server. In Example 2-5, I create the `/var/lib/cvroot` directory.

Example 2-5. Creating a repository

```
$ mkdir /var/lib/cvroot
$ chgrp anthill /var/lib/cvroot
$ ls -la /home
total 2
```

Example 2-5. Creating a repository (continued)

```
drwxr-xr-x  2 root  anthill  4096 Jun 28 16:31 cvsroot
$ chmod g+srwx /var/lib/cvsroot
$ ls -la /home
total 2
drwxrwsr-x  2 root  anthill  4096 Jun 28 16:33 cvsroot
$ cvs -d /var/lib/cvsroot init
$ ls -la /home
total 3
drwxrwsr-x  3 root  anthill  4096 Jun 28 16:56 cvsroot
$ ls -la /var/lib/cvsroot
total 12
drwxrwsr-x  3 root  anthill  4096 Jun 28 16:56 .
drwxrwsr-x 10 root  staff    4096 Jun 28 16:35 ..
drwxrwsr-x  3 root  anthill  4096 Jun 28 16:56 CVSROOT
$ chown -R cvs /var/lib/cvsroot
```

To allow others to use the repository, create a Unix group for CVS users and *chgrp* the repository's root directory to that group. Set the directory's *SGID* bit, so that files created in the directory have the same group ID as the directory's group ID (this can prevent trouble later). Make the directory group-writable, -readable, and -executable.

To minimize security risks, create a user named *cvs* to own the repository and the administrative files. *chown* the repository's root directory and administrative files to that username. Chapter 6 explains security.

Execute the following command to set up your chosen directory as a CVS repository:

```
cvs -d repository_root_directory init
```

CVS commands follow the format:

```
cvs [cvs-options] command [command-options]
```

The CVS options modify the behavior of CVS as a whole, and the command options modify the behavior of a CVS command. This is explained more fully in Chapter 3.

Example 2-5 illustrates the entire process of creating a directory and then creating a CVS repository within that directory. In this case, the CVS option is *-d repository_path*, and the command is *init*. There is no command option. *anthill* is the name of the group with access to CVS, and *cvs* is the name of the CVS owner.



Debian Linux has a script, *cvs-makerepos*, that builds a repository based on preexisting Debian configuration scripts. See *man cvs-makerepos* for more information and *man cvsconfig* for an automated system for configuring a Debian CVS repository.

Setting up the repository produces a place to store projects and also adds the special *CVSROOT* directory. The *CVSROOT* directory contains configuration and meta-data files. Chapter 6 explains this directory in more detail. Projects are stored in sub-directories of the repository root directory, which is */var/lib/cvsroot* in our example.

Importing Projects

When you have created a new repository, you may want to import a *project*—a related collection of files stored under a single directory. It is possible to store a single file under CVS, but it will also be a project and you will need to store it under its own project directory. CVS needs to be able to create a subdirectory to store meta-data about the project.

Before loading a project into CVS, consider the project’s structure. If you move a file after it has been created and stored in CVS, CVS treats it as two files: the original in the original location and the new file in the new location. The history of the file is then split into two parts. Decide how you want to structure the source files for a project before you import it into CVS.

If you will eventually want to distribute your project’s files across several unrelated directories, it is best to develop the project under a single root directory, then distribute the files as part of the installation script. Chapter 7 describes the issue of project structure in more detail.



If you have binary files or other files that are not plain text, please see the information on binary files in Chapter 3 before adding them to the repository.

Create your initial project directory structure, possibly in */tmp*. Once the project is stored in CVS, this initial version can be removed. You won’t be using it as a sandbox and the project is duplicated in CVS, so there’s no reason to retain it.

Once you have your initial structure, add any initial files you want. Change into the root directory of the project. Then, from within that directory, import the project with the command:

```
cvs -d repository_path import name_of_project vendor_tag release_tag
```

If the repository is on the local machine, use the full path of the repository directory for the repository path. If the repository is on a remote server, see “Accessing Remote Repositories” for help on specifying the repository path.

For most cases, you will not need to know about vendor tags and release tags. CVS requires them to be present, but for now you can use the name of the project as the vendor tag and the current revision name as the release tag. These names must start with a letter and can contain only alphanumeric characters, underscores, and hyphens. See Example 2-6, which illustrates the process of creating a project structure and some project files, and then importing the project into CVS.

The vendor tag and release tag are explained in Chapter 8.

Example 2-6. Importing a project

```
/tmp$ mkdir example
/tmp$ touch example/file1
/tmp$ touch example/file2
/tmp$ cd example
/tmp/example$ cvs -d /var/lib/cvsroot import example example_project ver_0-1
```

After you run the commands in Example 2-6, CVS opens an editor window, shown in Figure 2-4. Enter a message to remind you what you intend this project to be.

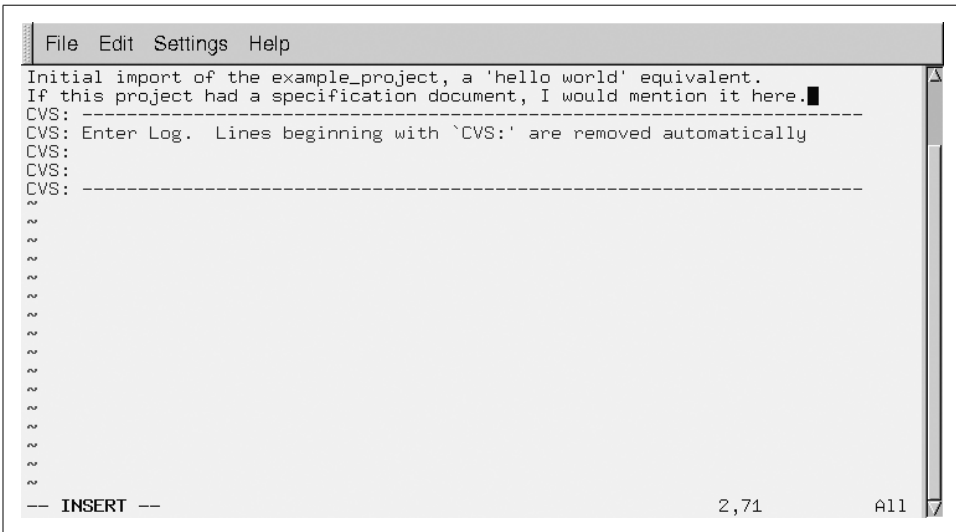


Figure 2-4. Enter an import message

The lines in the editor window that start with “CVS:” will not be included in the project’s history. The text displayed in the editor is configurable through the *rcsinfo* file described in Chapter 7.

The default editor is *vi*. You need to type an *i* before inserting text; type *ESC* to return to the mode in which you can move the cursor around. The movement keys are *h*, *j*, *k*, and *l*. To save and exit, use *ESC* followed by *:wq RETURN*. Chapter 3 explains how to change the editor to something other than *vi*.

After exiting from the editor, CVS completes the import, as shown in Example 2-7.

Example 2-7. Completing the import

```
N example/file2" 5L, 241C written
N example/file1

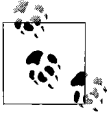
No conflicts created by this import
```

In the repository, the imported project is stored as a collection of RCS files. Example 2-8 shows the files for the project imported in Example 2-7.

Example 2-8. Repository contents

```
$ ls -la /var/lib/cvsroot
total 16
drwxrwsr-x  4 root    anthill  4096 Jun 28 17:06 .
drwxrwsr-x 10 root    staff    4096 Jun 28 16:35 ..
drwxrwsr-x  3 root    anthill  4096 Jun 28 16:56 CVSROOT
drwxrwsr-x  2 jenn    anthill  4096 Jun 28 17:09 example
$ ls -la /var/lib/cvsroot/example
total 16
drwxrwsr-x  2 jenn    anthill  4096 Jun 28 17:09 .
drwxrwsr-x  4 root    anthill  4096 Jun 28 17:06 ..
-r--r--r--  1 jenn    anthill   387 Jun 28 17:06 file1,v
-r--r--r--  1 jenn    anthill   387 Jun 28 17:06 file2,v
```

Once you've created your project, back up your CVS repository. You should continue to back up the repository periodically during your project's lifetime. Once the repository has been backed up and the project verified, you can remove the original files. You need sole use of the repository when you do a backup.



Chapter 6 explains how to back up a repository.

Before you do any work on the project, verify that the project is in CVS by checking out a sandbox (see “Checking Out Files” later in this chapter). Don't try to use your original files as a sandbox. You must do any new work in files that you check out to a sandbox. You may want to remove the original files to prevent yourself from accidentally modifying them.

Accessing Remote Repositories

There are several ways to access a remote repository. This quickstart guide uses the *ext* method with the SSH protocol, but if your system administrator gives you different instructions, follow those. Chapter 8 explains the use of remote repositories in detail. The *ext* and SSH approach uses the *ext* repository access method, with an SSH client as the program that performs the connection. These are also explained in Chapter 8.

Your first step, at least the first step that I recommend, is to install SSH on the client machine. Make sure that the client-end SSH protocol matches the server's SSH protocol. Set up SSH keys or passwords and test the connection. Using SSH enables you to create a secure connection to the remote repository.

Next, if you're on a Unix or Linux system, set the *CVS_RSH* environment variable on your client machine to the name of your SSH program, usually *ssh* or *ssh2*. Graphical clients that support the *ext* and SSH approach may have *ssh* as an authentication type option in the dialog that requests the repository path. Check the documentation for your client.



In WinCVS and gCVS, call up the Preferences dialog under the Admin menu and select “ssh” as the Authentication method under the General tab. In MacCVS (for OS X), *ssh* is available under the MacCVS menu. MacCVS for OS 9 and earlier versions do not support SSH as a standard option. Instead, you can make use of SSH from MacCVS via port tunnelling, as described in Appendix A.

If the repository is on the same machine as the client, the repository path is simply the full path to the repository’s root directory. On a remote machine, the repository path takes the form:

```
[ :method: ] [ [ [ user ] [ :password ] @ ] hostname [ : [ port ] ] ] /path
```

The method is the protocol used to connect to the repository. To use SSH, use *ext* as the method. Include the username and the *@* if the username on the server differs from the username on the client. If you don’t have an SSH key on the host, the system asks you for a password when you try to log in.



The *ext* method doesn’t use the password or port portions of the repository path.

Use the following command from your operating-system prompt to run a CVS command against the remote repository:

```
cvs -d repository_path command
```

For instance, Example 2-9 shows how to import a project into a remote CVS repository.

Example 2-9. Remote repository import

```
bash-2.05a$ cvs -d cvs:/home/cvs import example no-vendor release-0
N example/file1h" 5L, 241C written
No conflicts created by this import
bash-2.05a$
```

Checking Out Files

CVS stores projects and files in a central repository, but you work from a working copy, called a *sandbox*, in your local directories. You create that sandbox with *cvs checkout*.

CVS creates the sandbox as a subdirectory of your current working directory. I like to create a directory to contain all my CVS sandboxes, and I use *~/cvs*. From whichever directory you want a sandbox created in, run the command:

```
cvs -d repository_path checkout project_name
```

This command checks out all files for the named project. If your repository is on the local machine, the repository path is the full pathname of the CVS repository. If your repository is on a remote server, see the earlier “Accessing Remote Repositories” section. Example 2-10 shows a local checkout.

Example 2-10. Local repository checkout

```
$ mkdir ~/cvs
$ cd ~/cvs
$ cvs -d /var/lib/cvsroot checkout example
cvs checkout: Updating example
U example/file1
U example/file2
```

The *checkout* command puts a copy of the project’s files and subdirectories into a directory named for the project, created in the current working directory. It also puts some administrative files of its own in a subdirectory of the project directory, called CVS.

You can check out an individual file or subdirectory of a project by replacing *project_name* with the pathname to the file or directory, from the project’s root directory. See Chapter 3 for more information.

CVS stores the repository path as part of the sandbox, so you should never again need to use *-d repository_path* in commands executed within that sandbox.

If you are checking out a sandbox from a remote repository, the repository path is slightly different than the path shown in the previous example. Example 2-11 shows a checkout from a remote repository.

Example 2-11. Remote repository checkout

```
$ cvs -d :ext:cvs:/home/cvs checkout cvsbook
cvs server: Updating cvsbook
U cvsbook/1-introduction.html
U cvsbook/2-installation.html
U cvsbook/3-quickstart.html
U cvsbook/acknowledgements
U cvsbook/outline_schedule
U cvsbook/proposal
U cvsbook/sources
U cvsbook/template
```

Committing Changes

Once you’ve checked out project files into a sandbox, you can edit those files with your preferred editor. Changes are not synchronized with the repository until you run the *cvs commit* command. This command is best run from the root directory of your sandbox, and it must be run from within the sandbox.

Commit to the repository frequently. Rules of thumb for when to commit include “every time the code compiles cleanly” and “every day before lunch and before you leave.”



In programming projects with several developers, try to avoid committing code that doesn't compile.

When you commit, CVS examines each directory and subdirectory below the current working directory. It searches for files that it is tracking that have changed, and commits all changes to the repository. See Example 2-12 for an example of committing files. Remember, the repository path is stored in the sandbox, so you don't need to specify the path explicitly in your *cvs commit* command.

Example 2-12. Committing files

```
$ cd ~/cvs/example
$ cvs commit
cvs commit: Examining .
```

If your repository is not on the local machine and your repository server doesn't have your SSH public key, CVS will ask for a password for the remote machine. If the server has the public key, your SSH client can use the private key to authenticate you.

If any files have been changed, CVS opens an editor to allow you to record a change message. By default, the editor is *vi*, just as when importing a project. Chapter 3 gives instructions on changing your editor.

I strongly recommend meaningful change notes. If you're trying to do a rollback and all you have are messages that say “fixed a few bugs,” you won't know which revision to roll back to. See Example 2-13 for an example of a good change note.

Example 2-13. Enter a commit message

```
CVS:-----
CVS: Enter Log. Lines beginning with 'CVS:' are removed automatically
Corrected bug #35. 'hello' was misspelled as 'helo'.
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS: file1
CVS:-----
```

After you exit the editor, CVS completes the commit, displaying messages similar to those in Example 2-14.

Example 2-14. Completing the commit

```
Checking in file1;
/var/lib/cvsroot/example/file1,v &lt;-- file1
```

Example 2-14. Completing the commit (continued)

```
new revision: 1.2; previous revision: 1.1
done
```

If a revision in the repository is more recent than the revision the sandbox was based on, *cvs commit* fails. Use the *cvs update* command to merge the changed files; then run *cvs commit* again. Example 2-15 shows the response to a failed commit.

Example 2-15. Failed commit

```
cvs server: Up-to-date check failed for 'file2'
cvs [server aborted]: correct above errors first!
cvs commit: saving log message in /tmp/cvst7onmJ
```

Updating Sandboxes

The *cvs update* command checks your sandbox against the repository and downloads any changed files to the sandbox. It complements the *cvs commit* command, which uploads changes from the sandbox to the repository. Use the *-d* command option to download new directories as well. Example 2-16 shows the use of *cvs update*.

Example 2-16. Updating the sandbox

```
$ cvs update -d
cvs update: Updating .
U file2
cvs update: Updating directory
$ ls
CVS directory file1 file2
```

As with committing, you should not have to specify the repository; it should be stored in the special CVS subdirectory in the sandbox. You must run *cvs update* from within the sandbox, and it is best to run it from the root directory of the sandbox to ensure that it checks all the subdirectories.

Note that *-d* means two different things, depending on where it is in the command. Recall that CVS commands take the form:

```
cvs [cvs-options] command [command-options]
```

As a CVS option, *-d* defines the directory path. As a command option to the *update* command, *-d* downloads directories and files. This is explained more in Chapter 3.

As the *update* command runs, it generates a list of files that are modified. To the immediate left of each filename, you will see a single uppercase letter. Those letters report the status of each file listed, and they have the following meanings:

U filename

Updated successfully. A newer version in the repository has replaced your sandbox version.

A filename

Marked for addition but not yet added to the repository (need to run a *cvs commit*).

R filename

Marked for removal but not yet removed from the repository (need to run a *cvs commit*).

M filename

Modified in your working directory. The file in the sandbox is more recent than the repository version or the sandbox and the repository both had changes that the system could safely merge into your sandbox copy (need to run a *cvs commit*).

C filename

There was a conflict between the repository copy and your copy. The conflict requires human intervention.

?filename

The file is in your working directory but not in the repository. CVS doesn't know what to do with it.

The A, R, and M codes mean that your sandbox contains changes that are not in the repository and it would be a good idea to run a *cvs commit*.

If CVS can't merge a modified file successfully with the copy in the repository, it announces the conflict in the output of *cvs update*, as shown in Example 2-17.

Example 2-17. File conflict

```
cvs/example$ cvs update
cvs server: Updating .
RCS file: /var/lib/cvsroot/example/file1,v
retrieving revision 1.3
retrieving revision 1.4
Merging differences between 1.3 and 1.4 into file1
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in file1
C file1
```

CVS automatically merges files when the changes are on different lines. If a line in the repository copy is different from the corresponding line in the sandbox copy, CVS reports a conflict and creates a file with the two revisions of the line surrounded by special marks, as shown in Example 2-18.

Example 2-18. Conflict marks

```
<<<<<<file2
This line came from the sandbox.
=====
This line came from the repository..
>>>>>> 1.4
```

The contents of the original file are stored in *.#file.revision* in the file's working directory, and the results of the merge are stored as the original filename. Search for these marks in the file with the original name, edit the file, then commit the changed file to the repository.

Adding Files

To add a file to a project in the repository, first create the file in your sandbox. Be sure to consider your project's structure and place the file in the correct directory. Then, issue the following command from the sandbox directory containing the file:

```
cvs add filename
```

This command marks the new file for inclusion in the repository. Directories are added with the same command. Files within a directory can't be added until the directory itself is added. A file is only marked for addition when you run *cvs add*; it is actually added to the repository when the next *cvs commit* is run. A directory is added to the repository immediately. Example 2-19 shows a file being created and added to the repository. Remember, the file is not actually stored in the repository until the *cvs commit* command is run.

Example 2-19. Adding files

```
$ touch file3
$ cvs add file3
cvs add: scheduling file `file3' for addition
cvs add: use 'cvs commit' to add this file permanently
$ cvs commit
...
Log message editor opens
...
RCS file: /var/lib/cvsroot/example/file3,v
done
Checking in file3;
/var/lib/cvsroot/example/file3,v &lt;-- file3
initial revision: 1.1
done
```



If you have binary files or other files that are not plain text, please see the section on binary files in Chapter 3 before adding them to the repository. These files should be added with the *-kb* command option.

As with committing, an editor window will open asking you to enter a log message describing the files to be added.

Removing Files

To remove a file from the repository, first remove the file from the sandbox directory; then run the following command from the sandbox directory that contained the file:

```
cv$ remove filename
```

The deletion does not take effect until the next *cv\$ commit* command is run; the file remains in the repository until then.

Example 2-20 shows a deletion. After the *cv\$ commit* is run, CVS doesn't remove the file entirely; it puts it in a special subdirectory in the repository called *Attic*. This saves the file history and enables the file to be returned to the repository later.

CVS opens an editor so you can record the reason for the file deletion, as it does when you commit changes.

Example 2-20. Removing a file

```
$ rm file3
$ cvs remove file3
cvs remove: scheduling `file3' for removal
cvs remove: use 'cvs commit' to remove this file permanently
$ cvs commit
...
Log message editor opens
...
Removing file3;" 9L, 308C written
/var/lib/cvsroot/example/file3,v &lt;-- file3
new revision: delete; previous revision: 1.1
done
```

CVS does not remove directories from the repository, because doing so would break the change tracking. Use the *-P* flag to *cv\$ checkout* and *cv\$ update* to avoid empty directories in your sandbox.

Quick Tips for Success

CVS is a tool for improving project development and system maintainence. Like all tools, there are ways to make your use of it more effective. Here are some tricks for using CVS well:

- Synchronize the clocks of computers sharing a repository to the same universal time. CVS relies on file timestamps to determine which files have changed.
- Give each developer his own sandbox, and communicate all file changes through CVS. This maintains change tracking and prevents developers from irretrievably overwriting each other's work.

- Update frequently, at least before starting work every day, to keep your sandbox current.
- Commit frequently to keep your repository current. Programmers typically update every time their code compiles cleanly; other people may commit after completing each block of work.
- Programming teams: use build-management tools and ensure that all files in the build are committed to the repository. Ensure that builds for testing or release come from the repository rather than a sandbox, but allow sandbox builds for programmers to do pre-alpha tests.