

Chapter 10

Complete CVS Reference

Organization and Conventions

This chapter attempts to be a complete reference to CVS commands, repository administrative files, keyword substitution, run control files, working copy files, and environment variables—everything CVS as of version 1.11.6 (as of July 2003).

This release is the first stable release under the new numbering scheme, where an odd minor release number means a stable release which contains only bug fixes from previous versions of CVS. This release fixes many bugs in CVS 1.11.5 and we recommend an upgrade!

The commands are the most important part of any CVS reference, so we'll start here.

Commands

This section is organized alphabetically to make it easy for you to look up a particular command or option. We're using the following conventions:

- Arguments to commands and options are in all-capital letters and are italicized in the synopsis that begins each explanation. (Note that this capitalization is just a convention; the arguments can also be shown in all-lowercase letters, as is the case on the command card at the beginning of this book.)

- Optional items appear between square brackets: [ ].
- If you must choose a command from a set, the choices are separated by bars, like this: "x | y | z."
- Plurals or ellipses indicate multiples, usually separated by whitespace. For example, ***FILES*** means one or more files, but **[*FILES*]** means zero or more files. The entry **[&*MOD*...]** means an ampersand followed immediately by a module name, then whitespace, then maybe another ampersand-module, and so on, zero or more times. (We use the ellipsis because a plural would have left it unclear whether the ampersand is needed only the first time or once for each module.)

  When a plural is parenthesized, as in ***FILE(S)***, it means that although technically there can be two or more files, usually there is only one.
- ***REV*** is often used to stand for a revision argument. This is usually either a revision number or a tag name (so it's sometimes seen as ***TAG***). There are very few places in CVS where you can use one but not the other, and those places are noted in the text.
- ***REPOSITORY*** and ***ROOTDIR*** are synonymous in CVS and can be used interchangeably.

General Patterns in CVS Commands

CVS commands follow this form:

```
cvs [GLOBAL_OPTIONS] COMMAND [OPTIONS] [FILES]
```

The second kind of options in this syntax are sometimes called "command options." Because there are so many of them, though, we will just call them "options" in most places to save space.

Many commands are meant to be run within a working copy and, therefore, can be invoked without file arguments. These commands default to all of the files in the current directory and below. So when we refer to the "file" or "files" in the text, we are talking about the files on which CVS is acting. Depending on how you invoked CVS, these files might or might not have been explicitly mentioned on the command line.

Date Formats

Many options take a date argument. CVS accepts a wide variety of date formats—too many to list here. When in doubt, stick with the standard ISO 8601 format:

```
2001-08-23
```

This means "23 August 2001" (in fact, "23 August 2001" is a perfectly valid date specifier, too, as long as you remember to enclose it in double quotes). If you need a time of day as well, you can do this:

```
"2001-08-23 21:20:30 CDT"
```

You can even use certain common English constructs, such as "now," "yesterday," and "12 days ago." In general, you can safely experiment with date formats; if CVS understands your format at all, it most likely will understand it in the way you intended. If it doesn't understand, it will exit with an error immediately.

Global Options

Here are all the global options to CVS.

--allow-root=*REPOSITORY*

The alphabetically first global option is one that is virtually never used on the command line. The **– –allow-root** option is used with the **pserver** command to allow authenticated access to the named repository (that is, a repository top level such as /usr/local/newrepos, not a project subdirectory such as /usr/local/newrepos/myproj).

This global option is virtually never used on the command line. Normally, the only place you'd ever use it is in /etc/inetd.conf files (see Chapter 3), which is also about the only place the **pserver** command is used.

Every repository to be accessed via **cvs pserver** on a given host needs a corresponding **– –allow-root** option in /etc/inetd.conf. This is a security device, meant to ensure that people can't use **cvs pserver** to gain access to private repositories.

(See also the node "Password Authentication Server" in the Cederqvist manual.)

-a

This option authenticates all communications with the server. This option has no effect unless you're connecting via the GSSAPI server (gserver). We don't cover GSSAPI connections in this book because they're still somewhat rarely used (although that might change). See the nodes "Global Options" and "GSSAPI Authenticated" in the Cederqvist manual for more information.

-b *(Obsolete)*

This option formerly specified the directory in which the RCS binaries could be found. CVS now implements the RCS functions internally, so this option has no effect; it is kept only for backward compatibility.

-d *REPOSITORY*

This option specifies the repository, which is either an absolute pathname or a more complex expression involving a connection method, username and host, and path. If it is an expression specifying a connection method, the general syntax is:

```
:METHOD:USER@HOSTNAME:PATH_TO_REPOSITORY
```

Here are examples using each of the connection methods:

- **:ext:jrandom@floss.red-bean.com:/usr/local/newrepos**—Connects using rsh, ssh, or some other external connection program. If the **CVS_RSH** environment variable is unset, this defaults to "rsh"; otherwise, it uses the value of that variable.
- **:server:jrandom@floss.red-bean.com:/usr/local/newrepos**—Like **:ext:**, but uses CVS's internal implementation of rsh. (This might not be available on all platforms.)
- **:pserver:jrandom@floss.red-bean.com:/usr/local/newrepos**—Connects using the password authenticating server (see "The Password-Authenticating Server" section in Chapter 3; see also the **login** command later in this chapter).
- **:kserver:jrandom@floss.red-bean.com:/usr/local/newrepos**—Connects using Kerberos authentication.
- **:gserver:jrandom@floss.red-bean.com:/usr/local/newrepos**—Connects using GSSAPI authentication.
- **:fork:jrandom@floss.red-bean.com:/usr/local/newrepos**—Connects to a local repository, but uses the client/server network protocol instead of directly accessing the repository files. This is useful for testing or debugging remote CVS behaviors from your local machine.
- **:local:jrandom@floss.red-bean.com:/usr/local/newrepos**—Accesses a local repository directly, as though only the absolute path to the repository had been given.

-e *EDITOR*

Invokes ***EDITOR*** for your commit message, if the commit message was not specified on the command line with the **-m** option. Normally, if you don't give a message with **-m**, CVS invokes the editor based on the **CVSEDITOR**, **VISUAL**, or **EDITOR** environment variables, which it checks in that order. Failing that, it invokes the popular Unix editor vi.

If you pass both the **-e** global option and the **-m** option to **commit**, CVS ignores the **-e** in favor of the commit message given on the command line (that way it's safe to use **-e** in a .cvsrc file).

-f

This global option suppresses reading of the .cvsrc file.

--help [*COMMAND*] or --H [*COMMAND*]

These two options are synonymous. If no ***COMMAND*** is specified, a basic usage message is printed to the standard output. If ***COMMAND*** is specified, a usage message for that command is printed.

--help-options

Prints out a list of all global options to CVS, with brief explanations.

--help-synonyms

Prints out a list of CVS commands and their short forms ("up" for "update," and so on).

-l

Suppresses logging of this command in the CVSROOT/history file in the repository. The command is still executed normally, but no record of it is made in the history file.

-n

This option doesn't change any files in the working copy or in the repository. In other words, the command is executed as a "dry run"—CVS goes through most of the steps of the command, but stops short of actually running it.

This option is useful when you want to see what the command would have done had you actually run it. One common scenario is when you want to see which files in your working directory have been modified, but don't want to do a full update (which would bring down changes from the repository). By running **cvs -n update**, you can see a summary of what's been done locally, without changing your working copy.

-q

This option tells CVS to be moderately quiet, by suppressing the printing of unimportant informational messages. What is considered "important" depends on the command. For example, in updates, CVS suppresses the messages that it normally prints on entering each subdirectory of the working copy, but still prints the one-line status messages for modified or updated files.

-Q

This option tells CVS to be very quiet, by suppressing all output except what is absolutely necessary to complete the command. Commands whose sole purpose is to produce some output (such as **diff** or **annotate**), of course, still give that output. However, commands that could have an effect independent of any messages that they might print (such as **update** or **commit**) print nothing.

-r

This option causes new working files to be created as read-only files (the same effect as setting the **CVSREAD** environment variable).

If you pass this option, checkouts and updates make the files in your working copy read-only (assuming your operating system permits it). Frankly, we are not sure why anyone would ever want to use this option.

-s *VARIABLE=VALUE*

This option sets an internal CVS variable named ***VARIABLE*** to ***VALUE***.

On the repository side, the CVSROOT/*info trigger files can expand such variables to values that were assigned in the **-s** option. For example, if CVSROOT/loginfo contains a line like this

```
myproj  /usr/local/bin/foo.pl ${=FISH}
```

and someone runs **commit** from a myproj working copy like this

```
yarkon$ cvs -s FISH=carp commit -m "fixed the bait bug"
```

the foo.pl script is invoked with **carp** as an argument. Note the funky syntax, though: The dollar sign, equal sign, and curly braces are all necessary—if any of them are missing, the expansion will not take place (at least not as intended). Variable names can contain alphanumerics and underscores only. Although it is not required that they consist entirely of capital letters, most people seem to follow that convention.

You can use the **-s** flag as many times as you like in a single command. However, if the trigger script refers to variables that aren't set in a particular invocation of CVS, the command still succeeds, but none of the variables are expanded, and the user sees a warning. For example, if loginfo has this

```
myproj  /usr/local/bin/foo.pl  ${=FISH}  ${=BIRD}
```

but the same command as before is run

```
yarkon$ cvs -s FISH=carp commit -m "fixed the bait bug"
```

then the person running the command sees a warning something like this (placed last in the output):

```
loginfo:31: no such user variable ${=BIRD}
```

and the foo.pl script is invoked with no arguments. However, if this command were run

```
yarkon$ cvs -s FISH=carp -s BIRD=vulture commit -m "fixed the bait bug"
```

there would be no warning, and both **${=FISH}** and **${=BIRD}** in loginfo would be correctly expanded. In either case, the commit itself would still succeed.

Note

*Although these examples all use **commit**, variable expansion can be done with any CVS command that can be noticed in a CVSROOT/ trigger file—which is why the **-s** option is global.*

See the section "Repository Administrative Files" later in this chapter for more details about variable expansion in trigger files.

-T *TEMPDIR*

This option stores any temporary files in ***TEMPDIR*** instead of wherever CVS normally puts them (specifically, this overrides the value of the **TMPDIR** environment variable, if any exists). ***TEMPDIR*** should be an absolute path.

This option is useful when you don't have write permission (and, therefore, CVS doesn't have it, either) to the usual temporary locations.

-t

This option traces the execution of a CVS command, causing CVS to print messages showing the steps that it's going through to complete a command. You might find it particularly useful in conjunction with **-n** to preview the effects of an unfamiliar command before running it for real. It can also be handy when you're trying to discover why a command failed.

-v or --version

This option causes CVS to print out its version and copyright information and then exit with no error. Since version 1.10, this option prints out a shorter text with version information. If the repository is remote, both the client and server versions are reported.

-w

This option causes new working files to be created read-write (overrides any setting of the **CVSREAD** environment variable). Because files are created read-write by default, this option is rarely used.

If both **-r** and **-w** are passed, **-w** dominates.

-x

This option encrypts all communications with the server and has no effect unless you're connecting via the GSSAPI server (**gserver**). We don't cover GSSAPI connections in this book, because they're still somewhat rarely used (although that might change). See the nodes "Global Options" and "GSSAPI Authenticated" in the Cederqvist manual for more information.

-z *GZIPLEVEL*

This option sets the compression level on communications with the server. The argument ***GZIPLEVEL*** must be a number from 1 to 9. Level 1 is minimal compression (very fast, but doesn't compress much); Level 9 is highest compression (uses a lot of CPU time, but sure does squeeze the data). Level 9 is useful only on very slow network connections. Most people find levels between 3 and 5 to be most beneficial.

A space between **-z** and its argument is optional.

List of Commands

Following is a list of all the CVS commands.

add [*OPTIONS*] [*FILES*]

- *Alternate names*—**ad**, **new**
- *Requires*—Working copy, repository
- *Changes*—Working copy

This command adds a new file or files to an existing project. Although CVS contacts the repository for confirmation, the file does not actually appear in it until a subsequent commit is performed. (See also **remove** and **import**.)

Options:

- **-k***KEYWORD_SUBSTITUTION_MODE* (or **-kKFLAG**)—Specifies that the file is to be stored with the given RCS keyword substitution mode. There is no space between the **-k** and its argument. (See the section "Keyword Substitution (RCS Keywords)" later in this chapter for a list of valid modes and examples.)
- **-m** ***MESSAGE*** (or **-m** ***MSG***)—Records ***MESSAGE*** as the creation message, or description, for the file. This is different from a per-revision log message—each file has only one description. Descriptions are optional.

Note

As of version 1.10.7, there is a bug in CVS whereby the description is lost if you add a file via client/server CVS. The rest of the ***add*** *process seems to work fine, however, if that's any comfort.*

admin [*OPTIONS*] [*FILES*]

- *Alternate names*—**adm**, **rcs**
- *Requires*—Working copy, repository
- *Changes*—Repository

This command is an interface to various administrative tasks—specifically, tasks applicable to individual RCS files in the repository, such as changing a file's keyword substitution mode or changing a log message after it's been committed.

Although **admin** behaves recursively if no files are given as arguments, you normally will want to name files explicitly. It's very rare for a single **admin** command to be meaningful when applied to all files in a project, or even in a directory. Accordingly, when the following explanations refer to the "file," they mean the file or (rarely) files passed as arguments to the **admin** command.

Note

If there is a system group named "cvsadmin" on the repository machine, only members of that group can run ***admin*** *(with the exception of the* ***cvs admin -k*** *command, which is always permitted). Thus you can disallow* ***admin*** *for all users by setting the group to have no users.*

Options:

- **-A*OLDFILE***—(Obsolete) Appends the RCS access list of ***OLDFILE*** to the access list of the file that is the argument to **admin**. CVS ignores RCS access lists, so this option is useless.
- **-a *USER1* [*,USER2...*]**—(Obsolete) Appends the users in the comma-separated list to the access list of the file. Like **-A**, this option is useless in CVS.
- **-b[*REV*]**—Sets the revision of the file's default branch (usually the trunk) to ***REV***. You won't normally need this option, because you can usually get the revisions you need via sticky tags, but you can use it to revert to a vendor's version if you're using vendor branches. There should be no space between the **-b** and its argument.
- **-e*USER1* [*,USER2...*]**—(Obsolete) Removes the usernames appearing in the comma-separated list from the access list of the RCS file. Like **-a** and **-A**, this option is now useless in CVS.
- **-i** or **-I**—These two are so obsolete we are not even going to tell you what they used to do. (See the Cederqvist manual if you're curious.)
- **-k*MODE*** (or **k*SUBST***)—Sets the file's default keyword substitution mode to ***MODE***. This option behaves like the **-k** option to **add**, only it gives you a way to change a file's mode after it's been added. There should be no space between **-k** and its argument. (See the section "Keyword Substitution (RCS Keywords)" later in this chapter for valid modes.)
- **-L**—Sets locking to "strict." (See **-l**.)
- **-l[*REV*]**—Locks the file's revision to ***REV***. If ***REV*** is omitted, it locks the latest revision on the default branch (usually the trunk). If ***REV*** is a branch, it locks the latest revision on that branch.

  The intent of this option is to give you a way to do "reserved checkouts," where only one user can edit the file at a time. We are not sure how useful this really is, but if you want to try it, you should probably do so in conjunction with the rcslock.pl script in the CVS source distribution's contrib/ directory. See comments in that file for further information. Among other things, those comments indicate that the locking must be set to "strict." (See **-L**.) There is no space between **-l** and its argument.
- **-m*REV*:*MESSAGE***—Changes the log message for revision ***REV*** to ***MESSAGE***. Very handy—along with **-k**, this is probably the most frequently used **admin** option. There are no spaces between the option and its arguments or around the colon between the two arguments. Of course, ***MESSAGE*** might contain spaces within itself (in which case, remember to surround it with quotes so the shell knows it's all one thing).
- **-N*NAME* [:[*REV*]]**—Just like **-n**, except it forces the override of any existing assignment of the symbolic name ***NAME***, instead of exiting with an error.

- **-n*NAME* [:[*REV*]]**—This is a generic interface to assigning, renaming, and deleting tags. There is no reason, as far as we can see, to prefer it instead of the **tag** command and the various options available there (**-d**, **-r**, **-b**, **-f**, and so on). We recommend using the **tag** command instead.

  In all cases where a ***NAME*** is assigned, CVS exits with an error if there is already a tag named ***NAME*** in the file (but see **-N**). There are no spaces between **-n** and its arguments. The ***NAME*** and optional ***REV*** can be combined in the following ways:

  - If only the ***NAME*** argument is given, the symbolic name (tag) named ***NAME*** is deleted.
  - If ***NAME:*** is given but no ***REV***, ***NAME*** is assigned to the latest revision on the default branch (usually the trunk).
  - If ***NAME:REV*** is given, ***NAME*** is assigned to that revision. ***REV*** can be a symbolic name itself, in which case it is translated to a revision number first (can be a branch number).
  - If ***REV*** is a branch number and is followed by a period ("."), ***NAME*** is attached to the highest revision on that branch. If ***REV*** is just **$**, ***NAME*** is attached to revision numbers found in keyword strings in the working files.

- **-o*RANGE***—Deletes the revisions specified by ***RANGE*** (also known as "outdating," hence the **-o**). Range can be specified in one of the following ways:

  - ***REV1::REV2***—Collapses all intermediate revisions between ***REV1*** and ***REV2***, so that the revision history goes directly from ***REV1*** to ***REV2***. After this, any revisions between the two no longer exist, and there will be a noncontiguous jump in the revision number sequence.
  - ***::REV***—Collapses all revisions between the beginning of ***REV***'s branch (which might be the beginning of the trunk) and ***REV***, noninclusively, of course. ***REV*** is then the first revision on that line.
  - ***REV::***—Collapses all revisions between ***REV*** and the end of its branch (which might be the trunk). ***REV*** is then the last revision on that line.
  - ***REV***—Deletes the revision ***REV*** (**-o1.8** would be equivalent to **-o1.7::1.9**).
  - ***REV1:REV2***—Deletes the revisions from ***REV1*** to ***REV2***, inclusive. They must be on the same branch. After this, you cannot retrieve ***REV1***, ***REV2***, or any of the revisions in between.
  - ***:REV***—Deletes revisions from the beginning of the branch (or trunk) to ***REV***, inclusive. (See the preceding warning.)
  - ***REV:***—Deletes revisions from ***REV*** to the end of its branch (or trunk), inclusive. (See the preceding warning.)

Instead of using this option to undo a bad commit, you should commit a new revision that undoes the bad change. There are no spaces between **-o** and its arguments.

Note

None of the revisions being deleted can have branches or locks. If any of the revisions have symbolic names attached, you have to delete them first with ***tag -d*** *or* ***admin -n****. (Actually, right now CVS only protects against deleting symbolically named revisions if you're using one of the* ***::*** *syntaxes, but the single-colon syntaxes might soon change to this behavior as well.)*

- **-q**—Tells CVS to run quietly; don't print diagnostic messages (just like the global **-q** option).
- **-s*STATE*[:*REV*]**—Sets the state attribute of revision ***REV*** to ***STATE***. If ***REV*** is omitted, the latest revision on the default branch (usually the trunk) is used. If ***REV*** is a branch tag or number, the latest revision on that branch is used.

  Any string of letters or numbers is acceptable for ***STATE***; some commonly used states are **Exp** for experimental, **Stab** for stable, and **Rel** for released. (In fact, CVS sets the state to **Exp** when a file is created.) Note that CVS uses the state **dead** for its own purposes, so don't specify that one.

  States are displayed in **cvs log** output, and in the **$Log** and **$State** RCS keywords in files. There is no space between **-s** and its arguments.
- **-t[*DESCFILE*]**—Replaces the description (creation message) for the file with the contents of ***DESCFILE***, or reads from standard input if no ***DESCFILE*** is specified.

  This useful option, unfortunately, does not currently work in client/server CVS. In addition, if you try it in client/server and omit ***DESCFILE***, any existing description for the file is wiped out and replaced with the empty string. If you need to rewrite a file's description, either use only local CVS on the same machine as the repository or **-t-*STRING***. There is no space between **-t** and its argument. ***DESCFILE*** may not begin with a hyphen ("-"). (See **-t-*STRING***.)
- **-t-*STRING***—Like **-t**, except that ***STRING*** is taken directly as the new description. ***STRING*** can contain spaces, in which case you should surround it with quotes. Unlike the other syntax for **-t**, this works in client/server as well as locally.
- **-U**—Sets locking to nonstrict. (See **-l** and **-L** options, discussed earlier.)
- **-u[*REV*]—**Unlocks revision ***REV***. (See **-l**.) If ***REV*** is omitted, CVS unlocks the latest lock held by the caller. If ***REV*** is a branch, CVS unlocks the latest revision on that branch. If someone other than the owner of a lock breaks the lock, a mail message is sent to the original locker. The content for this message is solicited on standard input from the person breaking the lock. There is no space between **-u** and its argument.

- ***-VRCS_VERSION_NUMBER***—(Obsolete) This used to be a way to tell CVS to produce RCS files acceptable to earlier versions of RCS. Now the RCS format used by CVS is drifting away from the RCS format used by RCS, so this option is useless. Specifying it results in an error.
- ***-xSUFFIX—***(Obsolete) Theoretically, this gives you a way to specify the suffix for RCS file names. However, CVS and related tools all depend on that suffix being the default (**,v**), so this option does nothing.

annotate [*OPTIONS*] [*FILES*]

- *Alternate name*—**ann**
- *Requires*—Working copy, repository
- *Changes*—Nothing

This command shows information on who last modified each line of each file and when. Each line of output corresponds to one line of the file. From left to right, the line displays the revision number of the last modification of that line, a parenthetical expression containing the user and date of the modification, a colon, and the contents of the line in the file.

For example, if a file looks like this

```
this is a test file
it only has too lines
I mean "two"
```

the annotations for that file could look like this

```
1.1          (jrandom  22-Aug-99): this is a test file
1.1          (jrandom  22-Aug-99): it only has too lines
1.2          (jrandom  22-Aug-99): I mean "two"
```

from which you would know that the first two lines were in the initial revision, and the last line was added or modified (also by jrandom) in revision 1.2.

Options:

- **-D *DATE***—Shows the annotations as of the latest revision no later than ***DATE***.
- **-f**—Forces use of the head revision if the specified tag or date is not found. You can use this in combination with **-D** or **-r** to ensure that there is some output from the **annotate** command, even if only to show revision 1.1 of the file.
- **-l**—Local. Runs in the current working directory only. Does not descend into subdirectories.
- **-R—**Recursive. Descends into subdirectories (the default). The point of the **-R** option is to override any **-l** option set in a .cvsrc file.

- **-r *REV***—Shows annotations as of revision ***REV*** (can be a revision number or a tag).

checkout [*OPTIONS*] *PROJECT(S)*

The annotate command does not annotate binary files unless you specify –F option.

- *Alternate names*—**co**, **get**
- *Requires*—Repository
- *Changes*—Current directory

This command checks out a module from the repository into a working copy. The working copy is created if it doesn't exist already and updated if it does. (See also **update**.)

Options:

- **-A**—Resets any sticky tags, sticky dates, or sticky **-k** (RCS keyword substitution mode) options. This is like the **-A** option to **update** and is probably more often used there than with **checkout**.
- **-c**—Doesn't check anything out; just prints the CVSROOT/modules file, sorted, on standard output. This is a good way to get an overview of what projects are in a repository. However, a project without an entry in modules does not appear. (This situation is quite normal because the name of the project's top-level directory in the repository functions as the project's "default" module name.)
- **-D *DATE***—Checks out the latest revisions no later than ***DATE***. This option is sticky, so you won't be able to commit from the working copy without resetting the sticky date. (See **-A**.) This option also implies **-P**, described later.
- **-d *DIR***—Creates the working copy in a directory named ***DIR***, instead of creating a directory with the same name as the checked-out module. If you check out only a portion of a project and the portion is located somewhere beneath the project's top level, CVS omits the locally empty intermediate directories. You can use **-N** to suppress this directory-collapsing behavior.
- **-f**—Forces checkout of the head revision if CVS does not find the specified tag or date. Most often used in combination with **-D** or **-r** to ensure that something always gets checked out.
- **-j *REV*[:*DATE*]** or **-j *REV1*[:*DATE*] -j *REV2*[:*DATE*]**—Joins (merges) two lines of development. This is just like the **-j** option to **update**, where it is more commonly used. (See **update** for details.)
- **-k *MODE***—Substitutes RCS keywords according to ***MODE*** (which can override the default modes for the files). The mode chosen will be sticky—future updates of the working copy will keep that mode. (See the section "Keyword Substitution (RCS Keywords)" later in this chapter for valid modes.)

- **-l**—Local. Checks out the top-level directory of the project only. Does not process subdirectories.
- **-N**—Suppresses collapsing of empty directories with **-d** option. (See **-d**.)
- **-n**—Doesn't run any checkout program that was specified with **-o** in CVSROOT/modules. (See the section "Repository Administrative Files" later in this chapter for more on this.)
- **-P**—Prunes empty directories from the working copy (like the **-P** option to **update**).
- **-p**—Checks out files to standard output, not into files (like the **-p** option to **update**).
- **-R**—Checks out subdirectories as well (the default). (See also the **-f** option.)
- **-r *TAG***—Checks out the project as of revision ***TAG*** (it would make almost no sense to specify a numeric revision for ***TAG***, although CVS lets you). This option is sticky and implies **-P**.
- **-s**—Like **-c**, but shows the status of each module and sorts by status. (See CVSROOT/modules in the section "Repository Administrative Files" for more information.)

commit [*OPTIONS*] [*FILES*]

- *Alternate names*—**ci**, **com**
- *Requires*—Working copy, repository
- *Changes*—Repository (and working copy administrative area)

This command commits changes from a working copy to the repository.

Options:

- **-F *MSGFILE***—Uses the contents of ***MSGFILE*** for the log message instead of invoking an editor. This option cannot be combined with **-m**.
- **-f**—Forces commit of a new revision even if no changes have been made to the files. **commit** does not recurse with this option (it implies **-l**). You can force it to recurse with **-R**.

Note

*This meaning of **-f** is at odds with its usual meaning ("force to head revision") in CVS commands.*

- **-l**—Local. Commits changes from the current directory only. Doesn't descend into subdirectories.
- **-m *MESSAGE***—Uses ***MESSAGE*** as the log message instead of invoking an editor. Cannot be used with **-F**.
- **-n**—Does not run any module program. (See the section "Repository Administrative Files" later in this chapter for information about module programs.)

- **-R**—Commits changes from subdirectories as well as from the current directory (the default). This option is used only to counteract the effect of a **-l** in .cvsrc.
- **-r *REV***—Commits to revision ***REV***, which must be either a branch or a revision on the trunk that is higher than any existing revision. Commits to a branch always go on the tip of the branch (extending it); you cannot commit to a specific revision on a branch. Use of this option sets the new revision as a sticky tag on the file. This can be cleared with **update -A**.

  The **-r REV** option implies **-f** as well. A new revision is committed even if there are no changes to commit.

diff [*OPTIONS*] [*FILES*]

- *Alternate names*—**di**, **dif**
- *Requires*—Working copy, repository
- *Changes*—Nothing

This command shows the difference between two revisions (in Unix **diff** format). When invoked with no options, CVS diffs the repository base revisions against the (possibly uncommitted) contents of the working copy. The "base" revisions are the latest revisions of this working copy retrieved from the repository. Note that there could be even later revisions in the repository, if someone else committed changes but this working copy hasn't been updated yet. (See also **rdiff**.)

Options:

- **-D *DATE***—Diffs against the latest revisions no later than ***DATE***. Behaves like **-r *REV***, except uses dates rather than revisions. See **-r** for details.
- **-k *MODE***—Expands RCS keywords in the diffs according to ***MODE***. See the section "Keyword Substitution (RCS Keywords)" later in this chapter for possible modes.
- **-l**—Local. If no files were specified as arguments, this option diffs files in the current directory, but does not descend into subdirectories.
- **-R**—Recursive. This option is the opposite of **-l**. This is the default behavior, so the only reason to specify **-R** is to counteract a **-l** in a .cvsrc file.
- **-r *REV*** or **-r *REV1* -r *REV2***—Diffs against (or between) the specified revisions. With one **-r** option, this diffs revision ***REV*** against your working copy of that file (so when multiple files are being diffed, ***REV*** is almost always a tag). With two **-r** options, it diffs ***REV1*** against ***REV2*** for each file (and the working copy is, therefore, irrelevant). The two revisions can be in any order—***REV1*** does not have to be an earlier revision than ***REV2***. The output reflects the direction of change. With no **-r** options, it shows the difference between the working file and the revision on which it is based.

Diff Compatibility Options

In addition to the preceding options, **cvs diff** also shares a number of options with the GNU version of the standard command-line **diff** program. Following is a complete list of these options, along with an explanation of a few of the most commonly used ones. (See the GNU diff documentation for the others.)

```
-0 -1 -2 -3 -4 -5 -6 -7 -8 -9
    --binary
    --brief
    --changed-group-format=ARG
    -c
      -C NLINES
      --context[=LINES]
    -e --ed
    -t --expand-tabs
    -f --forward-ed
    --horizon-lines=ARG
    --ifdef=ARG
    -w --ignore-all-space
    -B --ignore-blank-lines
    -i --ignore-case
    -I REGEXP
       --ignore-matching-lines=REGEXP
    -h
    -b --ignore-space-change
    -T --initial-tab
    -L LABEL
      --label=LABEL
    --left-column
    -d --minimal
    -N --new-file
    --new-line-format=ARG
    --old-line-format=ARG
    --paginate
    -n --rcs
    -s --report-identical-files
    -p
    --show-c-function
    -y --side-by-side
-F REGEXP
--show-function-line=REGEXP
-H --speed-large-files
--suppress-common-lines
-a --text
--unchanged-group-format=ARG
```

```
-u
  -U NLINES
  --unified[=LINES]
-V ARG
-W COLUMNS
    --width=COLUMNS
```

Following are the GNU **diff** options most frequently used with **cvs diff**:

- **-B**—Ignores differences that are merely the insertion or deletion of blank lines (lines containing nothing but whitespace characters).
- **-b**—Ignores differences in the amount of whitespace. This option treats all whitespace sequences as being equal and ignores whitespace at line end. More technically, this option collapses each whitespace sequence in the input to a single space and removes any trailing whitespace from each line, before taking the diff. (See also **-w**.)
- **-c**—Shows output in context diff format, defaulting to three lines of context per difference (for the sake of the patch program, which requires at least two lines of context).
- **-C NUM**--**context=NUM**—Like -c, but with NUM lines of context.
- **-i**—Compares case insensitively. Treats upper- and lowercase versions of a letter as the same.
- **-u**—Shows output in unified diff format.
- **-w**—Ignores all whitespace differences, even when one side of the input has whitespace where the other has none. Essentially a stronger version of **-b**.

---

edit [*OPTIONS*] [*FILES*]

- *Alternate names*—None
- *Requires*—Working copy, repository
- *Changes*—Permissions in working copy, watchlist in repository

This command signals that you are about to begin editing a watched file or files. It also adds you as a temporary watcher to the file's watch list (you'll be removed when you do **cvs unedit**). (See also **watch**, **watchers**, **unedit**, and **editors**.)

Options:

- **-a *ACTIONS***—Specifies for which actions you want to be a temporary watcher. ***ACTIONS*** should be **edit**, **unedit**, **commit**, all, or none. (If you don't use **-a**, the temporary watch will be for all actions.)
- **-l**—Local. Signals editing for files in the current working directory only.
- **-R**—Recursive (this is the default). Opposite of **-b**; you would only need to pass **-R** to counteract a **-l** in a .cvsrc file.

editors [*OPTIONS*] [*FILES*]

- *Alternate names*—None
- *Requires*—Working copy, repository
- *Changes*—Nothing

This command shows who is currently editing a watched file. (See also **watch**, **watchers**, **edit**, and **unedit**.)

Options:

- **-l**—Local. Views editors for files in current directory only.
- **-R**—Recursive. Views editors for files in this directory and its subdirectories (the default). You might need to pass **-R** to counteract a **-l** in a .cvsrc file, though.

export [*OPTIONS*] *PROJECT(S)*

- *Alternate names*—**exp**, **ex**
- *Requires*—Repository
- *Changes*—Current directory

This command exports files from the repository to create a project tree that is not a working copy (has no CVS/ administrative subdirectories). Useful mainly for packaging distributions.

Options:

- **-D *DATE***—Exports the latest revisions no later than ***DATE***.
- **-d *DIR***—Exports into ***DIR*** (otherwise, defaults to the module name).
- **-f**—Forces use of head revisions, if a given tag or date would result in nothing being found (for use with **-D** or **-r**).
- **-k *MODE*** (or **-k *KFLAG***)—Expands RCS keywords according to ***MODE***. (See the section "Keyword Substitution (RCS Keywords)" later in this chapter.)
- **-l**—Local. Exports only the top level of the project, no subdirectories.
- **-N**—Doesn't "collapse" empty intermediate directories. This option is like the **-N** option to **checkout**.
- **-n**—Does not run a module program as might be specified in CVSROOT/modules. (See the section "Repository Administrative Files" later in this chapter for more about this.)
- **-P**—Prunes empty directories (like the **-P** option to **checkout** or **update**).

- **-R**—Recursive. Exports all subdirectories of the project (the default). The only reason to specify **-R** is to counteract a **-l** in a .cvsrc file.
- ***-r REV***—Exports revision ***REV***. ***REV*** is almost certainly a tag name, not a numeric revision.

gserver

This is the GSSAPI (Generic Security Services API) server. This command is not normally run directly by users. Instead, it is started up on the server side when a user connects from a client with the **:gserver:** access method:

```
cvs -d :gserver:floss.red-bean.com:/usr/local/newrepos checkout myproj
```

Note

GSSAPI provides, among other things, Kerberos version 5; for Kerberos version 4, use ***:kserver:****.*

Setting up and using a GSSAPI library on your machines is beyond the scope of this book. (See the node "GSSAPI Authenticated" in the Cederqvist manual for some useful hints, however.)

Options: None.

history [*OPTIONS*] [*FILENAME_SUBSTRING(S)*]

- *Alternate names*—**hi**, **his**
- *Requires*—Repository, CVSROOT/history
- *Changes*—Nothing

This command shows a history of activity in the repository. Specifically, this option shows records of checkouts, commits, rtags, updates, and releases. By default, the option shows checkouts (but see the **-x** option). This command won't work if there's no CVSROOT/history file in the repository.

The **history** command differs from other CVS commands in several ways. First, it must usually be given options to do anything useful (and some of those options mean different things for **history** than they do elsewhere in CVS). Second, instead of taking full file names as arguments, it takes one or more substrings to match against file names (all records matching at least one of those substrings are retrieved). Third, the output of the **history** command looks a lot like line noise until you learn to read it, so after we list the options of this command, we will explain the output format in the "Output of the **history** Command" sidebar. (See also **log**.)

Options:

- **-a**—Shows history for all users (otherwise, defaults to **self**).

- **-b *STR***—Shows data back to record containing string ***STR*** in the module name, file name, or repository path.
- **-c**—Reports commits.
- **-D *DATE***—Shows data since ***DATE*** (the usual CVS date formats are available).
- **-e**—Everything; reports on all record types.
- **-f *FILE***—Reports the most recent event concerning ***FILE***. You can specify this option multiple times. This is different from the usual meaning of **-f** in CVS commands: "Force to head revision as a last resort."
- **-l**—Shows the record representing the last (as in "most recent") event of each project. This is different from the usual meaning of **-l** in CVS commands: "Run locally, do not recurse."
- **-m *MODULE***—This provides a full report about ***MODULE*** (a project name). You can specify this option multiple times.
- **-n *MODULE***—Reports the most recent event about ***MODULE***. For example, checking out the module is about the module itself, but modifying or updating a file inside the module is about that file, not about the module. You can specify this option multiple times. This is different from the usual meaning of **-n** in CVS commands: "Don't run a CVSROOT/modules program."
- **-o**—Shows checkout records (the default).
- **-p *REPOSITORY***—Shows data for a particular directory in the repository. You can specify this option multiple times. The meaning of this option differs from the usual meaning of **-p** in CVS commands: "Pipe the data to standard output instead of a file."
- **-r *REV***—Shows records referring to revisions since the revision or tag named ***REV*** appears in individual RCS files. Each RCS file is searched for the revision or tag.
- **-T**—Reports on all tag events.
- **-t *TAG***—Shows records since tag ***TAG*** was last added to the history file. This differs from the **-r** flag in that it reads only the CVSROOT/history file, not the RCS files, and is therefore much faster.
- **-u *USER***—Shows events associated with ***USER***. You can specify this option multiple times.
- **-w**—Shows records that are associated with the same working directory from which you are invoking history.
- **-X *HISTORYFILE***—Uses ***HISTORYFILE*** instead of CVSROOT/history. This option is mainly for debugging and is not officially supported; nevertheless, you might find it useful (perhaps for generating human-readable reports from old history files you've kept around).

- ***-x TYPES***—Reports on events specified in ***TYPES***. A single letter, from the set "TOEFWUCGMAR," represents each type; any number of letters can be combined. Here is what they mean:
  - **T**—Tag
  - **O**—Checkout
  - **E**—Export
  - **F**—Release
  - **W**—Update (newly obsolete file removed from working copy)
  - **U**—Update (file was checked out over user file)
  - **C**—Update (merge, with conflicts)
  - **G**—Update (merge, no conflicts)
  - **M**—Commit (file was modified)
  - **A**—Commit (file was added)
  - **R**—Commit (file was removed)

  The default, if no **-x** option is given, is to show checkouts (like **-x O**).
- ***-z ZONE***—Displays times in output as for time zone ***ZONE***. ***ZONE*** is an abbreviated time zone name, such as UTC, GMT, BST, CDT, CCT, and so on. A complete list of time zones is available in the TimezoneTable in the file lib/getdate.c in the CVS source distribution.

Output of the history Command

The output of the **history** command is a series of lines, and the date is in the ISO 8601 format (YYYY-MM-DD); each line represents one "history event" and starts with a single code letter indicating what type of event it is. For example:

```
yarkon$ cvs history -D yesterday -x TMO
M 2001-08-20 20:19 +0000 jrandom 2.2             baar       myproj == <remote>
M 2001-08-20 04:18 +0000 jrandom 1.2             README     myproj == <remote>
O 2001-08-20 05:15 +0000 jrandom myproj =myproj= ~/src/*
M 2001-08-20 05:33 +0000 jrandom 2.18            README.txt myproj == ~/src/myproj
O 2001-08-20 14:25 CDT jrandom myproj =myproj= ~/src/*
O 2001-08-20 14:26 CDT jrandom [2001.08.23.19.26.03] myproj =myproj= ~/src/*
O 2001-08-20 14:28 CDT jrandom [Exotic_Greetings-branch] myproj =myproj= ~/src/*
```

The code letters are the same as for the **-x** option just described. Following the code letter is the date of the event (expressed in UTC/GMT time, unless the **-z** option is used), followed by the user responsible for the event.

A revision number, tag, or date might appear after the user, but only if such is appropriate for the event (date or tag will be in square brackets and formatted as shown in the preceding example). If you commit a file, it shows the new revision number; if you check out with **-D** or **-r**, the sticky date or tag is shown in square brackets. For a plain checkout, nothing extra is shown.

Next comes the name of the file in question, or module name if the event is about a module. If the former, the next two things are the module/project name and the location of the working copy in the user's home directory. If the latter, the next two things are the name of the module's checked-out working copy (between two equal signs), followed by its location in the user's home directory. (The name of the checked-out working copy might differ from the module name if the **-d** flag is used with checkout.)

---

import [*OPTIONS*] *REPOSITORY VENDOR_TAG RELEASE_TAG(S)*

- *Alternate names*—**im**, **imp**
- *Requires*—Repository, current directory (the source directory)
- *Changes*—Repository

This command imports new sources into the repository, either creating a new project or creating a new vendor revision on a vendor branch of an existing project. (See Chapter 4 for a basic explanation of vendor branches in **import**, which will help you to understand the following options.)

It's normal to use **import** to add many files or directories at once or to create a new project. To add single files, you should use **add**.

Options:

- **-b *BRANCH***—Imports to vendor branch ***BRANCH***. (***BRANCH*** is an actual branch number, not a tag.) This is rarely used but can be helpful if you get sources for the same project from different vendors. A normal **import** command assumes that the sources are to be imported on the default vendor branch, which is "1.1.1." Because it is the default, you normally don't bother to specify it with **-b**:

  ```
  yarkon$ cvs import -m "importing from vendor 1" theirproj THEM1 THEM1-0
  ```

  To import to a vendor branch other than the default, you must specify a different branch number explicitly:

  ```
  yarkon$ cvs import -b 1.1.3 -m "from vendor 2" theirproj THEM2 THEM2-0
  ```

  The 1.1.3 branch can absorb future imports and be merged like any other vendor branch. However, you must make sure any future imports that specify **-b 1.1.3** also use the same vendor tag (**THEM2**). CVS does not check to make sure that the vendor branch matches the vendor tag. However, if they mismatch, odd and unpredictable things will happen.

Note

Vendor branches are odd-numbered, the opposite of regular branches.

- **-d**—Takes the file's modification time as the time of import instead of using the current time. This does not work with client/server CVS.
- **-I *NAME*** or **-I *IGN***—Gives file names that should be ignored in the import. You can use this option multiple times in one import. Wildcard patterns are supported: ***.foo** means ignore everything ending in ".foo." (See CVSROOT/cvsignore in "Repository Administrative Files" for details about wildcards.)

  The following file and directory names are ignored by default:

  - .
  - ..
  - .#*
  - #*
  - ,*
  - _$*
  - *~
  - *$
  - *.a
  - *.bak
  - *.BAK
  - *.elc
  - *.exe
  - *.ln
  - *.o
  - *.obj
  - *.olb
  - *.old
  - *.orig

- *.rej
- *.so
- *.Z
- .del-*
- .make.state
- .nse_depinfo
- core
- CVS
- CVS.adm
- cvslog.*
- RCS
- RCSLOG
- SCCS
- tags
- TAGS

You can suppress the ignoring of those file name patterns, as well as any specified in .cvsignore, CVSROOT/cvsignore, and the **CVSIGNORE** environment variable, by using **-I !**. That is,

```
yarkon$ cvs import -I ! -m "importing the universe" proj VENDOR VENDOR_0
```

imports all files in the current directory tree, even those that would otherwise be ignored.

Using a **-I !** clears whatever ignore list has been created to that point, so any **-I** options that came before it would be nullified, but any that come after will still count. Thus,

```
yarkon$ cvs import -I ! -I README.txt -m "some msg" theirproj THEM THEM_0
```

is not the same as

```
yarkon$ cvs import -I README.txt -I ! -m "some msg" theirproj THEM THEM_0
```

The former ignores (fails to import) README.txt, whereas the latter imports it.

- **-k *MODE*** or **-k *KFLAG***—Sets the default RCS keyword substitution mode for the imported files. (See the section "Keyword Substitution (RCS Keywords)" later in this chapter for a list of valid modes.)

- **-m *MESSAGE*** (or **-m *MSG***)—Records **MESSAGE** as the import log message.
- **-W *SPEC***—Specifies filters based on file names that should be in effect for the **import**. You can use this option multiple times. (See CVSROOT/cvswrappers in "Repository Administrative Files" for details about wrapper specs.)

init *NEW_REPOSITORY*

- *Alternate names*—None
- *Requires*—Location for new repository
- *Creates*—Repository

This command creates a new repository (that is, a root repository in which many different projects are stored). You will almost always want to use the global **-d** option with this, as in

```
yarkon$ cvs -d /usr/local/yet_another_repository init
```

because even if you have a **CVSROOT** environment variable set, it's probably pointing to an existing repository, which would be useless and possibly dangerous in the context of this command. (See Chapter 3 for additional steps that you should take after initializing a new repository.)

Options: None.

kserver

This is the Kerberos server. (If you have Kerberos libraries version 4 or below—version 5 just uses GSSAPI; see the section that covers the **gserver** command earlier in this chapter.) This command is not normally run directly by users but is instead started up on the server side when a user connects from a client with the **:kserver:** access method:

```
cvs -d :kserver:floss.red-bean.com:/usr/local/newrepos checkout myproj
```

Setting up and using Kerberos on your machine is beyond the scope of this book. (However, see the node "Kerberos Authenticated" in the Cederqvist manual for some useful hints.)

Options: None.

log [*OPTIONS*] [*FILES*]

- *Alternate names*—**lo**, **rlog**
- *Requires*—Working copy, repository
- *Changes*—Nothing

This command shows log messages for a project, or for files within a project. The output of **log** is not quite in the same style as the output of other CVS commands, because **log** is based on an older RCS program (**rlog**). Its output format gives a header, containing various pieces

of non–revision-specific information about the file, followed by the log messages (arranged by revision). Each revision shows not merely the revision number and log message, but also the author and date of the change and the number of lines added or deleted. All times are printed in UTC (GMT), not local time.

Because **log** output is per file, a single commit involving multiple files might not immediately appear as a conceptually atomic change. However, if you read all of the log messages and dates carefully, you might be able to reconstruct what happened. (For information about a tool that can reformat multifile log output into a much more readable form, see "**cvs2cl.pl**: Generate GNU-Style ChangeLogs from CVS Logs" in Chapter 10 for details.) (See also **history**.)

Options:

As you read over the following filtering options, it might not be completely clear how they behave when combined. A precise description of **log**'s behavior is that it takes the intersection of the revisions selected by **-d**, **-s**, and **-w**, intersected with the union of those selected by **-b** and **-r**.

- **-b**—Prints log information about the default branch only (usually the highest branch on the trunk). This is usually done to avoid printing the log messages for side branches of development.
- **-d*DATES***—Prints log information for only those revisions that match the date or date range given in ***DATES***, a semicolon-separated list. Dates can be given in any of the usual date formats (see the "Date Formats" section earlier in this chapter) and can be combined into ranges as follows:
  - ***DATE1*<*DATE2***—Selects revisions created between ***DATE1*** and ***DATE2***. If ***DATE1*** is after ***DATE2***, use "**>**" instead; otherwise, no log messages are retrieved.
  - **<*DATE*** ***DATE*>**—All revisions from ***DATE*** or earlier.
  - **>*DATE*** ***DATE*<**—All revisions from ***DATE*** or later.
  - ***DATE***—Just selects the most recent single revision from ***DATE*** or earlier.

  You can use "**<=**" and "**>=**" instead of "**<**" and "**>**" to indicate an inclusive range (otherwise, ranges are exclusive). Multiple ranges should be separated with semicolons; for example

```
yarkon$ cvs log -d"2001-06-01<2001-07-01;2001-08-01<2001-09-01"
```

  selects log messages for revisions committed in June or August of 2001 (skipping July). There can be no space between **-d** and its arguments.
- **-h**—Prints only the header information for each file, which includes the file name, working directory, head revision, default branch, access list, locks, symbolic names (tags), and the file's default keyword substitution mode. No log messages are printed.

- **-l**—Local. Runs only on files in the current working directory.
- **-N**—Omits the list of symbolic names (tags) from the header. This can be helpful when your project has a lot of tags but you're interested in seeing only the log messages.
- **-R**—Prints the name of the RCS file in the repository.

Note

*This is different from the usual meaning of **-R**: "recursive." There's no way to override a **-l** for this command, so don't put **log -l** in your .cvsrc.*

- **-r*REVS***—Shows log information for the revisions specified in ***REVS***, a comma-separated list. ***REVS*** can contain both revision numbers and tags. Ranges can be specified like this:
  - ***REV1:REV2***—Revisions from ***REV1*** to ***REV2*** (they must be on the same branch).
  - ***:REV***—Revisions from the start of ***REV***'s branch up to and including ***REV***.
  - ***REV:***—Revisions from ***REV*** to the end of ***REV***'s branch.
  - ***BRANCH***—All revisions on that branch, from root to tip.
  - ***BRANCH1:BRANCH2***—A range of branches; all revisions on all the branches in that range.
  - ***BRANCH.***—The latest (tip) revision on ***BRANCH***.

  Finally, a lone **-r**, with no argument, means select the latest revision on the default branch (normally the trunk). There can be no space between **-r** and its argument.

Note

*If the argument to **-r** is a list, it is comma-separated, not semicolon-separated like **-d**.*

- **-S**—Suppress the header information when no revisions are selected.
- **-s*STATES***—Selects revisions whose state attribute matches one of the states given in ***STATES***, a comma-separated list. There can be no space between **-s** and its argument.

Note

*If the argument to **-s** is a list, it is comma-separated, not semicolon-separated like **-d**.*

- **-t**—Like **-h**, but also includes the file's description (its creation message).
- **-w*USERS*** (or **w*LOGINS***)—Selects revisions committed by users whose usernames appear in the comma-separated list ***USERS***. A lone **-w** with no ***USERS*** means to take the username of the person running **cvs log**.

Remember that when user aliasing is in effect (see the section "The Password-Authenticating Server" in Chapter 3), CVS records the CVS username, not the system username, with each commit. There can be no space between **-w** and its argument.

Note

*If the argument to **-w** is a list, it is comma-separated, not semicolon-separated like **-d**.*

login

- *Alternate names*—**logon**, **lgn**
- *Requires*—Repository
- *Changes*—~/.cvspass file

This command contacts a CVS server and confirms authentication information for a particular repository. This command does not affect either the working copy or the repository; it just confirms a password (for use with the **:pserver:** access method) with a repository and stores the password for later use in the .cvspass file in your home directory. Future commands accessing the same repository with the same username will not require you to rerun **login**, because the client-side CVS will just consult the .cvspass file for the password.

If you use this command, you should specify a repository using the **pserver** access method, like this

```
yarkon$ cvs -d :pserver:jrandom@floss.red-bean.com:/usr/local/newrepos
```

or by setting the CVSROOT environment variable.

If the password changes on the server side, you have to rerun **login**.

Options: None.

logout

- *Alternate names*—None
- *Requires*—~/.cvspass file
- *Changes*—~/.cvspass file

This command is the opposite of **login**—removes the password for this repository from .cvspass.

Options: None.

pserver

- *Alternate names*—None
- *Requires*—Repository

♦ *Changes*—Nothing

This is the password-authenticating server. This command is not normally run directly by users but is started up from /etc/inetd.conf on the server side when a user connects from a client with the **:pserver:** access method. (See also the sections on the **login** and **logout** commands earlier in this chapter, and the section on the .cvspass file later in this chapter. See "The Password-Authenticating Server" section of Chapter 3 for details on setting up a password-authenticating CVS server.)

Options: None.

rannotate [*OPTIONS*] *TAG PROJECT(S)*

This is a new simple command to get log messages and annotations without having to have a checked-out copy.

rdiff [*OPTIONS*] *PROJECTS*

♦ *Alternate names*—**patch**, **pa**

♦ *Requires*—Repository

♦ *Changes*—Nothing

This command is like the **diff** command, except it operates directly in the repository and, therefore, requires no working copy. This command is meant for obtaining the differences between one release and another of your project, in a format suitable as input to the **patch** program (perhaps so you can distribute patch files to users who want to upgrade).

The operation of the **patch** program is beyond the scope of this book. However, note that if the patch file contains diffs for files in subdirectories, you might need to use the **-p** option to **patch** to get it to apply the differences correctly. (See the **patch** documentation for more about this.) (See also **diff**.)

Options:

♦ **-c**—Prints output in context diff format (the default).

♦ **-D** *DATE* or **-D** *DATE1* **-D** *DATE2*—With one date, this shows the differences between the files as of *DATE* and the head revisions. With two dates, it shows the differences between the dates.

♦ **-f**—Forces the use of head revision if no matching revision is found for the **-D** or **-r** flag (otherwise, **rdiff** would just ignore the file).

♦ **-l**—Local. Won't descend into subdirectories.

♦ **-R**—Recursive. Descends into subdirectories (the default). You only specify this option to counteract a **-l** in your .cvsrc.

- **-r *REV* -r *REV1* -r *REV2***—With one revision, this shows the differences between revision ***REV*** of the files and the head revisions. With two, it shows the differences between the revisions.
- **-s**—Displays a summary of differences. This shows which files have been added, modified, or removed, without showing changes in their content. The output looks like this:

```
yarkon$ cvs -Q rdiff -s -D 2001-08-20 myproj
File myproj/Random.txt is new; current revision 1.4
File myproj/README.txt changed from revision 2.1 to 2.20
File myproj/baar is new; current revision 2.3
```

- **-t**—Shows the diff between the top two revisions of each file. This is a handy shortcut for determining the most recent changes to a project. This option is incompatible with **-D** and **-r**.
- **-u**—Prints output in unidiff format. Older versions of **patch** can't handle unidiff format; therefore, don't use **-u** if you're trying to generate a distributable patch file—use **-c** instead.

release [*OPTIONS*] *DIRECTORY*

- *Alternate names*—**re**, **rel**
- *Requires*—Working copy
- *Changes*—Working copy, CVSROOT/history

This command cancels a checkout (indicates that a working copy is no longer in use). Unlike most CVS commands that operate on a working copy, this one is not invoked from within the working copy but from directly above it (in its parent directory). You either have to set your **CVSROOT** environment variable or use the **-d** global option, because CVS will not be able to find out the repository from the working copy.

Using **release** is never necessary. Because CVS doesn't normally do locking, you can just remove your working copy.

However, if you have uncommitted changes in your working copy, or you want your cessation of work to be noted in the CVSROOT/history file (see the **history** command), you should use **release**. CVS first checks for any uncommitted changes; if there are any, it warns you and prompts for continuation. Once the working copy is actually released, that fact is recorded in the repository's CVSROOT/history file.

Options:

- **-d**—Deletes the working copy if the release succeeds. Without **-d**, the working copy remains on disk after the release.

Note

If you created any new directories inside your working copy but did not add them to the repository, they are deleted along with the rest of the working copy, if you specified the ***-d*** *flag.*

remove [*OPTIONS*] [*FILES*]

- *Alternate names*—**rm**, **delete**
- *Requires*—Working copy
- *Changes*—Working copy

This command removes a file from a project. Normally, the file itself is removed from disk when you invoke this command (but see **-f**). Although this command operates recursively by default, it is common to explicitly name the files being removed. Note the odd implication of the previous sentence: Usually, you run **cvs remove** on files that don't exist anymore in your working copy.

Although the repository is contacted for confirmation, the file is not actually removed until a subsequent **commit** is performed. Even then, the RCS file is not really removed from the repository; if it is removed from the trunk, it is just moved into an Attic/ subdirectory, where it is still available to exist on branches. If it is removed from a branch, its location is not changed, but a new revision with state **dead** is added on the branch. (See also **add**.)

Options:

- **-f**—Force. Deletes the file from disk before removing it from CVS. This meaning differs from the usual meaning of **-f** in CVS commands: "Force to head revision."
- **-l**—Local. Runs only in current working directory.
- **-R**—Recursive. Descends into subdirectories (the default). This option exists only to counteract a **-l** in .cvsrc.

rlog [*OPTIONS*] *TAG PROJECT(S)*

This is a new simple command to get log messages and annotations without having to have a checked-out copy.

rtag [*OPTIONS*] *TAG PROJECT(S)*

- **-S**—Suppress the header information when no revisions are selected.
- *Alternate names*—**rt**, **rfreeze**
- *Requires*—Repository
- *Changes*—Repository

This command tags a module directly in the repository (requires no working copy). You probably need to have your **CVSROOT** environment variable set or use the **-d** global option for this command to work. (See also **tag**.)

Options:

- **-a**—Clears the tag from any removed files, because removed files stay in the repository for historical purposes, but are no longer considered part of the live project. Although it's illegal to tag files with a tag name that's already in use, there should be no interference if the name is used only in removed files (which, from the current point of view of the project, don't exist anymore).
- **-b**—Creates a new branch, with branch name ***TAG***.
- **-D *DATE***—Tags the latest revisions no later than ***DATE***.
- **-d**—Deletes the tag. No record is made of this change—the tag simply disappears. CVS does not keep a change history for tags.
- **-F**—Forces reassignment of the tag name, if it happens to exist already for some other revision in the file.
- **-f**—Forces to head revision if a given tag or date is not found. (See **-r** and **-D**.)
- **-l**—Local. Runs in the current directory only.
- **-n**—Won't execute a tag program from CVSROOT/modules. (See the section "Repository Administrative Files" later in this chapter for details about such programs.)
- **-R**—Recursive. Descends into subdirectories (the default). The **-R** option exists only to counteract a **-l** in .cvsrc.
- **-r *REV***—Tags revision ***REV*** (which may itself be a tag name).

To move or delete branch tages, you need to use the -b switch.

server

This command starts up a CVS server. This command is never invoked by users (unless they're trying to debug the client/server protocol), so forget it was even mentioned.

Options: None.

status [*OPTIONS*] [*FILES*]

- *Alternate names*—**st**, **stat**
- *Requires*—Working copy
- *Changes*—Nothing

This command shows the status of files in the working copy.

Options:

- **-l**—Local. Runs in the current directory only.
- **-R**—Recursive. Descends into subdirectories (the default). The **-R** option exists only to counteract a **-l** in .cvsrc.
- **-v**—Shows tag information for the file.

tag [*OPTIONS*]*TAG* [*FILES*]

- *Alternate names*—**ta**, **freeze**
- *Requires*—Working copy, repository
- *Changes*—Repository

This command attaches a name to a particular revision or collection of revisions for a project. Often called "taking a snapshot" of the project. This command is also used to create branches in CVS. (See the **-b** option. See also **rtag**.)

Options:

- **-b**—Creates a branch named ***TAG***.
- **-c**—Checks that the working copy has no uncommitted changes. If it does, the command exits with a warning, and no tag is made.
- **-D *DATE***—Tags the latest revisions no later than ***DATE***.
- **-d**—Deletes the tag. No record is made of this change; the tag simply disappears. CVS does not keep a change history for tags.
- **-F**—Forces reassignment of the tag name, if it happens to exist already for some other revision in the file.
- **-f**—Forces to head revision if a given tag or date is not found. (See **-r** and **-D**.)
- **-l**—Local. Runs in the current directory only.
- **-n**—No execution of tag program.
- **-R**—Recursive. Descends into subdirectories (the default). The **-R** option exists only to counteract a **-l** in .cvsrc.
- **-r *REV***—Tags revision ***REV*** (which can itself be a tag name).

To move or delete branch tages, you need to use the –B switch.

unedit [*OPTIONS*] [*FILES*]

- *Alternate names*—None

- *Requires*—Working copy, repository
- *Changes*—edit/watch lists in the repository

This command signals to watchers that you have finished editing a file. (See also **watch**, **watchers**, **edit**, and **editors**.)

Options:

- **-l**—Local. Signals editing for files in the current working directory only.
- **-R**—Recursive (opposite of **-l**). Recursive is the default; the only reason to pass **-R** is to counteract a **-l** in your .cvsrc file.

update [*OPTIONS*] [*FILES*]

- *Alternate names*—**up**, **upd**
- *Requires*—Working copy, repository
- *Changes*—Working copy

This command merges changes from the repository into your working copy. As a side effect, it indicates which files in your working copy are modified (but if the **-Q** global option is passed, these indications won't be printed). (See also **checkout**.)

Options:

- **-A**—Clears any sticky tags, sticky dates, or sticky RCS keyword expansion modes. This might result in the contents of files changing, if the trunk-head revisions are different from the former sticky revisions. (Think of **-A** as being like a fresh checkout of the project trunk.)
- **-D *DATE***—Updates to the most recent revisions no later than ***DATE***. This option is sticky and implies **-P**. If the working copy has a sticky date, commits are not possible.
- **-d**—Retrieves absent directories (that is, directories that exist in the repository but not yet in the working copy). Such directories might have been created in the repository after the working copy was checked out. Without this option, **update** operates on only the directories present in the working copy; new files are brought down from the repository, but new directories are not. (See also **-P**.)
- **-f**—Forces to head revision if no matching revision is found with the **-D** or **-r** flags.
- **-I *NAME*** or **-I *IGN***—Like the **-I** option of **import**.
- **-j *REV*[:*DATE*]** or **-j *REV1*[:*DATE*] -j *REV2*[:*DATE*]**—Joins, or merges, two lines of development. Ignoring the optional ***DATE*** arguments for the moment (we'll get to them later), here's how **-j** works: If only one **-j** is given, it takes all changes from the common ancestor to ***REV*** and merges them into the working copy. The "common ancestor" is the

latest revision that is ancestral to both the revisions in the working directory and to ***REV***. If two **-j** options are given, it merges the changes from ***REV1*** to ***REV2*** into the working copy.

The special tags ***HEAD*** and ***BASE*** can be used as arguments to **-j**; they mean the most recent revision in the repository, and the revision on which the current working copy file is based, respectively.

As for the optional ***DATE*** arguments, if ***REV*** is a branch, it is normally taken to mean the latest revision on that branch, but you can restrict it to the latest revision no later than ***DATE***. The date should be separated from the revision by a colon, with no spaces. For instance:

```
yarkon$ cvs update -j ABranch:2001-07-01 -j ABranch:2001-08-01
```

In this example, different dates on the same branch are used, so the effect is to take the changes on that branch from July to August and merge them into the working copy. However, note that there is no requirement that the branch be the same in both **-j** options.

- **-k *MODE*** (or **-k *KFLAG***)—Does RCS keyword substitution according to ***MODE***. (See the section "Keyword Substitution (RCS Keywords)" later in this chapter.) The mode remains sticky on the working copy, so it will affect future updates (but see **-A**).
- **-l**—Local. Updates the current directory only.
- **-P**—Prunes empty directories. Any CVS-controlled directory that contains no files at the end of the update is removed from the working copy. (See also **-d**.)
- **-p**—Sends file contents to standard output instead of to the files. Used mainly for reverting to a previous revision without producing sticky tags in the working copy. For example:

  ```
  yarkon$ cvs update -p -r 1.3 README.txt > README.txt
  ```

  Now README.txt in the working copy has the contents of its past revision 1.3, just as if you had hand-edited it into that state.
- **-R**—Recursive. Descends into subdirectories to update (the default). The only reason you'd specify it is to counteract a **-l** in .cvsrc.
- **-r *REV***—Updates (or downdates, or crossdates) to revision ***REV***. When updating a whole working copy, ***REV*** is most often a tag (regular or branch). However, when updating an individual file, it is just as likely to be a revision number as a tag.

  This option is sticky. If the files are switched to a nonbranch tag or sticky revision, they cannot be committed until the stickiness is removed. (See **-A**.) If ***REV*** was a branch tag, however, commits are possible. They'll simply commit new revisions on that branch.

- **-WSPEC**—Specifies wrapper-style filters to use during the update. You can use this option multiple times. There is no space between **-W** and its argument. See CVSROOT/cvswrappers in the section "Repository Administrative Files" in this chapter for details about wrapper specs.

watch on|off|add|remove [*OPTIONS*] [*FILES*]

- *Alternate names*—None
- *Requires*—Working copy, repository
- *Changes*—Watch list in repository

This command sets a watch on one or more files. Unlike most CVS commands, **watch** requires a further subcommand to do something useful. (See also **watchers**, **edit**, **editors**, and **unedit**, and CVSROOT/users in the "Repository Administrative Files" section later in this chapter.)

Subcommands:

- **on**—Declares that the files are being watched. This means that they are created read-only on checkout, and users should do **cvs edit** to make them read-write (and notify any watchers that the file is now being edited). Turning on a **watch** does not add you to the watch list for any files. (See **watch add** and **watch remove** for that.)
- **off**—Opposite of **watch on**. Declares that the files are no longer being watched.
- **add**—Adds you to the list of watchers for this file. You are notified when someone commits or runs **cvs edit** or **cvs unedit** (but see the **-a** option).
- **remove**—Opposite of **watch add**. Removes you from the list of watchers for this file.

Options (for use with any **watch** subcommand). All three options have the same meanings as they do for **edit**:

- **-a *ACTIONS***
- **-l**
- **-R**

watchers [*OPTIONS*] [*FILES*]

- *Alternate names*—None
- *Requires*—Working copy, repository
- *Changes*—Nothing

This command shows who is watching which files.

Options—these options mean the same thing here as they do for **edit**:

- **-l**
- **-R**

Keyword Substitution (RCS Keywords)

CVS can perform certain textual substitutions in files, allowing you to keep some kinds of information automatically up to date in your files. All of the substitutions are triggered by a certain keyword pattern, surrounded by dollar signs. For example,

```
$Revision$
```

in a file expands to something like

```
$Revision: 1.5 $
```

and CVS continues to keep the revision string up to date as new revisions are committed.

Controlling Keyword Expansion

By default, CVS performs keyword expansion unless you tell it to stop. You can permanently suppress keyword expansion for a file with the **-k** option when you use **add** to add the file to the project, or you can turn it off later by invoking **admin** with **-k**. The **-k** option offers several different modes of keyword control; usually you want mode **o** or **b**; for example:

```
yarkon$ cvs add -ko chapter-9.sgml
```

This command added chapter-9.sgml to the project with keyword expansion turned off. It sets the file's default keyword expansion mode to **o**, which means no substitution. (Actually, the "o" stands for "old," meaning to substitute the string with its old value, which is the same as substituting it for itself, resulting in no change. Surely, this logic made sense to somebody at the time.)

Each file's default keyword mode is stored in the repository. However, each working copy can also have its own local keyword substitution mode—accomplished with the **-k** options to **checkout** or **update**. You can also have a mode in effect for the duration of just one command, with the **-k** option to **diff**.

Here are all the possible modes, presented with the **-k** option prepended (as one would type at a command line). You can use any of these options as either the default or local keyword substitution mode for a file:

- **-kkv**—Expands to keyword and value. This is the default keyword expansion mode, so you don't need to set it for new files. You might use it to change a file from another keyword mode, however.
- **-kkvl**—Like **-kkv**, but includes the locker's name if the revision is currently locked. (See the **-l** option to **admin** for more on this.)
- **-kk**—Won't expand values in keyword strings, just uses the keyword name. For example, with this option,

  ```
  $Revision: 1.5 $
  ```

  and

  ```
  $Revision$
  ```

  would both "expand" (okay, contract) to:

  ```
  $Revision$
  ```

- **-ko**—Reuses the keyword string found in the file (hence "o" for "old"), as it was in the working file just before the commit.
- **-kb**—Like **-ko**, but also suppresses interplatform line-end conversions. The "b" stands for "binary"; it is the mode you should use for binary files.
- **-kv**—Substitutes the keyword with its value. For example

  ```
  $Revision$
  ```

  might become:

  ```
  1.5
  ```

  Of course, after that's happened once, future substitutions will not take place, so you should use this option with care.

List of Keywords

These are all the dollar-sign-delimited keywords that CVS recognizes. Following is a list of the keywords, brief descriptions, and examples of the expanded forms:

- **$Author$**—Author of the change:

  ```
  $Author: jrandom $
  ```

- **$Date$**—The date and time of the change, in UTC (GMT):

```
$Date: 2001/08/23 18:21:13 $
```

- **$Header$**—Various pieces of information thought to be useful: full path to the RCS file in the repository, revision, date (in UTC), author, state, and locker. (Lockers are rare, although in the following example, qsmith has a lock.):

```
$Header: /usr/local/newrepos/myproj/hello.c,v 1.1 2001/06/01 03:21:13 \
       jrandom Exp qsmith $
```

- **Id**—Like **$Header$**, but without the full path to the RCS file:

```
$Id: hello.c,v 1.1 2001/06/01 03:21:13 jrandom Exp qsmith $
```

- **$Locker$**—Name of the person who has a lock on this revision (usually no one):

```
$Locker: qsmith $
```

- **Log**—The log message of this revision, along with the revision number, date, and author. Unlike other keywords, the previous expansions are not replaced. Instead, they are pushed down, so that the newest expansion appears at the top of an ever-growing stack of **Log** messages:

```
$Log: hello.c,v $    Revision 1.12  2001/07/19 06:12:43  jrandom
    say hello in Aramaic
```

  Any text preceding the **Log** keyword on the same line will be prepended to the downward expansions too; if you use it in a comment in a program source file, all of the expansion is commented, too.

- **$Name$**—Name of the sticky tag:

```
$Name: release_1_14 $
```

- **$RCSfile$**—Name of the RCS file in the repository:

```
$RCSfile: hello.c,v $
```

- **$Revision$**—Revision number:

```
$Revision: 1.1 $
```

- **$Source$**—Full path to the RCS file in the repository:

```
$Source: /usr/local/newrepos/myproj/hello.c,v $
```

- **$State$**—State of this revision:

```
$State: Exp $
```

Repository Administrative Files

The repository's administrative files are stored in the CVSROOT subdirectory of the repository. These files control various aspects of CVS behavior (in that repository only, of course).

Generally, the administrative files are kept under revision control just like any other file in the repository (the exceptions are noted). However, unlike other files, checked-out copies of the administrative files are stored in the repository right next to the corresponding RCS files. These checked-out copies actually govern CVS's behavior.

The normal way to modify the administrative files is to check out a working copy of the CVSROOT module, make your changes, and commit. CVS updates the checked-out copies in the repository automatically. (See the "checkoutlist" subsection in the "List of Repository Administrative Files" section later in this chapter.) In an emergency, however, it is also possible to edit the checked-out copies in the repository directly.

You might also want to refer to the discussion of administrative files in Chapter 4, which includes examples.

Shared Syntax

In all of the administrative files, a "**#**" at the beginning of a line signifies a comment; that line is ignored by CVS. A backslash preceding a newline quotes the newline out of existence.

Some of the files (commitinfo, loginfo, taginfo, and rcsinfo) share more syntactic conventions, as well. In these files, on the left of each line is a regular expression (which is matched against a file or directory name), and the rest of the line is a program, possibly with arguments, which is invoked if something is done to a file matching the regular expression. The program is run with its working directory set to the top of the repository.

In these files, there are two special regular expressions that may be used: **ALL** and **DEFAULT**. **ALL** matches any file or directory, whether or not there is some other match for it, and **DEFAULT** matches only if nothing else matched.

Shared Variables

The info files also allow certain variables to be expanded at runtime. To expand a variable, precede it with a dollar sign (and put it in curly braces just to be safe). Here are the variables CVS knows about:

- **${CVSROOT}**—The top of the repository.
- **${RCSBIN}**—(Obsolete) Don't use this variable. It is applicable only in CVS version 1.9.18 and older. Specifying it in more current versions might result in an error.
- **${CVSEDITOR} ${VISUAL} ${EDITOR}**—These all expand to the editor that CVS is using for a log message.
- **${USER}**—The user running CVS (on the server side).

User Variables

Users can also set their own variables when they run any CVS command. (See the **-s** global option.) These variables can be accessed in the *info files by preceding them with an equal sign, as in **${=*VAR*}**.

List of Repository Administrative Files

The following sections discuss all the repository administrative files.

checkoutlist

This file contains a list of files for which checked-out copies should be kept in the repository. Each line gives the file name and an error message for CVS to print if, for some reason, the file cannot be checked out in the repository:

```
FILENAME  ERROR_MESSAGE
```

Because CVS already knows to keep checked-out copies of the existing administrative files, they do not need to be listed in checkoutlist. Specifically, the following files never need entries in checkoutlist: loginfo, rcsinfo, editinfo, verifymsg, commitinfo, taginfo, ignore, checkoutlist, cvswrappers, notify, modules, readers, writers, and config.

commitinfo

This file specifies programs to run at commit time, based on what's being committed. Each line consists of a regular expression followed by a command template:

```
REGULAR_EXPRESSION PROGRAM [ARGUMENTS]
```

The program ***PROGRAM*** is passed additional arguments following any arguments you might have written into the template. These additional arguments are the full path to the

repository, followed by the name of each file about to be committed. These files can be examined by ***PROGRAM***; their contents are the same as those of the working copy files about to be committed. If ***PROGRAM*** exits with nonzero status, the commit fails; otherwise, it succeeds. (See also the "Shared Syntax" section earlier in this chapter.)

config

This file controls various global (non–project-specific) repository parameters. The syntax of each line is

```
ParameterName=yes|no
```

except for the **LockDir** parameter, which takes an absolute pathname as argument.

The following parameters are supported:

- **RCSBIN** (default: **=no**)—(Obsolete). This option is silently accepted for backward compatibility, but no longer has any effect.
- **SystemAuth** (default: **=no**)—If "yes," CVS **pserver** authentication tries the system user database—usually /etc/passwd—if a username is not found in CVSROOT/passwd. If "no," the user must exist in CVSROOT/passwd to gain access via the **:pserver:** method.
- **PreservePermissions** (default: **=no**)—If "yes," CVS tries to preserve permissions and other special file system information (such as device numbers and symbolic link targets) for files. You probably don't want to do this, as it does not necessarily behave as expected. (See the node "Special Files" in the Cederqvist manual for details.)
- **TopLevelAdmin** (default: **=no**)—If "yes," checkouts create a CVS/ subdirectory next to each working copy tree (in the parent directory of the working copy). This can be useful if you will be checking out many working copies from the same repository; on the other hand, setting it here affects everyone who uses this repository.
- **LockDir** (unset by default)—The argument after the equal sign is a path to a directory in which CVS can create lockfiles. If not set, lockfiles are created in the repository, in locations corresponding to each project's RCS files. This means that users of those projects must have file-system-level write access to those repository directories.

cvsignore

This file ignores certain files when doing updates, imports, or releases. By default, CVS already ignores some kinds of files. (For a full list, see the **-I** option to **import**, earlier in this chapter.) You can add to this list by putting additional file names or wildcard patterns in the cvsignore file. Each line gives a file name or pattern; for example:

```
README.msdos
*.html
blah?.out
```

This causes CVS to ignore any file named "README.msdos," any file ending in ".html," and any file beginning with "blah" and ending with ".out." (Technically, you can name multiple files or patterns on each line, separated by whitespace, but it is more readable to keep them to one per line. The whitespace separation rule does, unfortunately, mean that there's no way to specify a space in a file name, except to use wildcards.)

A "**!**" anywhere in the list cancels all previous entries. (See **$CVSIGNORE** in the section "Environment Variables" later in this chapter for a fuller discussion of ignore processing.)

cvswrappers

This file specifies certain filtering behaviors based on file name. Each line has a file-globbing pattern (that is, a file name or file wildcards), followed by an option indicating the filter type and an argument for the option.

Options:

- **-m**—Specifies an update method. Possible arguments are **MERGE**, which means to merge changes into working files automatically, and **COPY**, which means don't try to automerge, but rather present users with both versions of the file and let them work it out. **MERGE** is the default, except for binary files (those whose keyword substitution mode is **-kb**). (See the "Keyword Substitution (RCS Keywords)" section earlier in this chapter.) Files marked as binary automatically use the **COPY** method, so there is no need to make a **-m COPY** wrapper for them.
- **-k**—Specifies a keyword substitution mode. All of the usual modes are possible. (See the "Keyword Substitution (RCS Keywords)" section earlier in this chapter for a complete list.)

Here is an example cvswrappers file:

```
*.blob    -m COPY
*.blink   -k o
```

This cvswrappers file says not to attempt merges on files ending in ".blob" and to suppress keyword substitution for files ending in ".blink." (See also .cvswrappers in the "Run Control Files" section later in this chapter.)

editinfo

This file is obsolete. Very.

history

This files stores an ever-accumulating history of activity in the repository, for use by the **cvs history** command. To disable this feature, simply remove the history file. If you don't remove the file, you should probably make it world-writable to avoid permission problems later.

The contents of this file do not modify CVS's behavior in any way (except for the output of **cvs history**, of course).

loginfo

This file specifies programs to run on the log message for each commit, based on what's being committed. Each line consists of a regular expression followed by a command template:

```
REGULAR_EXPRESSION PROGRAM [ARGUMENTS]
```

The program ***PROGRAM*** is passed the log message on its standard input.

Several special codes are available for use in the arguments: **%s** expands to the names of the files being committed, **%V** expands to the old revisions from before the commit, and **%v** expands to the new revisions after the commit. When multiple files are involved, each element of the expansion is separated from the others by whitespace. For example, in a commit involving two files, **%s** might expand into **hello.c README.txt**, and **%v** into **1.17 1.12**.

You can combine codes inside curly braces, in which case each unit of expansion is internally separated by commas and externally separated from the other units by whitespace. Continuing the previous example, **%{sv}** expands into **hello.c,1.17 README.txt,1.12**.

If any % expansion is done at all, the expansion is prefixed by the subdirectory in the repository. So that last expansion would actually be:

```
myproj  hello.c,1.17  README.txt,1.12
```

If ***PROGRAM*** exits with nonzero status, the commit fails; otherwise, it succeeds. (See also the "Shared Syntax" section earlier in this chapter.)

modules

This file maps names to repository directories. The general syntax of each line is:

```
MODULE [OPTIONS] [&OTHERMODULE...] [DIR] [FILES]
```

DIR need not be a top-level project directory—it could be a subdirectory. If any ***FILES*** are specified, the module consists of only those files from the directory.

An ampersand followed by a module name means to include the expansion of that module's line in place.

Options:

- **-a**—This is an "alias" module, meaning it expands literally to everything after the ***OPTIONS***. In this case, the usual ***DIR/FILES*** behavior is turned off, and everything after the ***OPTIONS*** is treated as other modules or repository directories.

If you use the **-a** option, you can exclude certain directories from other modules by putting them after an exclamation point (!). For example

```
top_proj -a !myproj/a-subdir !myproj/b-subdir myproj
```

means that checking out **top_proj** will get all of **myproj** except **a-subdir** and **b-subdir**.

- **-d *NAME***—Names the working directory ***NAME*** instead of the module name.
- **-e *PROGRAM***—Runs ***PROGRAM*** whenever files in this module are exported.
- **-i *PROGRAM***—Runs ***PROGRAM*** whenever files in this module are committed. The program is given a single argument—the full pathname in the repository of the file in question. (See commitinfo, loginfo, and verifymsg for more sophisticated ways to run commit-triggered programs.)
- **-o *PROGRAM***—Runs ***PROGRAM*** whenever files in this module are checked out. The program is given a single argument: the name of the module.
- **-s *STATUS***—Declares a status for the module. When the modules file is printed (with **cvs checkout -s**), the modules are sorted by module status and then by name. This option has no other effects in CVS, so go wild. You can use it to sort anything—status, person responsible for the module, or the module's file language, for example.
- **-t *PROGRAM***—Runs ***PROGRAM*** whenever files in this module are tagged with **cvs rtag**. The program is passed two arguments: the name of the module and the tag name. The program is not used for **tag**, only for **rtag**. We have no idea why this distinction is made. You might find the taginfo file more useful if you want to run programs at tag time.
- **-u *PROGRAM***—Runs ***PROGRAM*** whenever a working copy of the module is updated from its top-level directory. The program is given a single argument: the full path to the module's repository.

notify

This file controls how the notifications for watched files are performed. (You might want to read up on the **watch** and **edit** commands, or see the section "Watches" in Chapter 6.) Each line is of the usual form:

```
REGULAR_EXPRESSION  PROGRAM  [ARGUMENTS]
```

A **%s** in ***ARGUMENTS*** is expanded to the name of the user to be notified, and the rest of the information regarding the notification is passed to ***PROGRAM*** on standard input (usually this information is a brief message suitable for emailing to the user). (See the section "Shared Syntax" earlier in this chapter.)

As shipped with CVS, the notify file has one line

```
ALL mail %s -s "CVS notification"
```

which is often all you need.

passwd

This file provides authentication information for the **pserver** access method. Each line is of the form:

```
USER:ENCRYPTED_PASSWORD[:SYSTEM_USER]
```

If no ***SYSTEM_USER*** is given, **USER** is taken as the system username.

rcsinfo

This file specifies a form that should be filled out for log messages that are written with an interactive editor. Each line of rcsinfo looks like:

```
REGULAR_EXPRESSION  FILE_CONTAINING_TEMPLATE
```

This template is brought to remote working copies at checkout time, so if the template file or rcsinfo file changes after checkout, the remote copies won't know about it and will continue to use the old template. (See also the section "Shared Syntax" earlier in this chapter.)

taginfo

This file runs a program at tag time (usually done to check that the tag name matches some pattern). Each line is of the form:

```
REGULAR_EXPRESSION  PROGRAM
```

The program is handed a set group of arguments. In order, they are the tag name, the operation (see below), the repository, and then as many file name/revision-number pairs as there are files involved in the tag. The file/revision pairs are separated by whitespace, like the rest of the arguments.

The operation is one of **add**, **mov**, or **del** (**mov** means the **-F** option to tag was used).

If ***PROGRAM*** exits with nonzero status, the tag operation will not succeed. (See also the section "Shared Syntax" earlier in this chapter.)

users

This file maps usernames to email addresses. Each line looks like:

```
USERNAME:EMAIL_ADDRESS
```

This command sends watch notifications to ***EMAIL_ADDRESS*** instead of to ***USERNAME*** at the repository machine. (All this really does is control the expansion of **%s** in the notify file.) If ***EMAIL_ADDRESS*** includes whitespace, make sure to surround it with quotes.

If user aliasing is being used in the passwd file, the username that will be matched is the CVS username (the one on the left), not the system username (the one on the right, if any).

val-tags

This file caches valid tag names for speedier lookups. You should never need to edit this file, but you might need to change its permissions, or even ownership, if people are having trouble retrieving or creating tags.

verifymsg

This file is used in conjunction with rcsinfo to verify the format of log messages. Each line is of the form:

```
REGULAR_EXPRESSION  PROGRAM  [ARGUMENTS]
```

The full path to the current log message template (see rcsinfo earlier in this chapter) is appended after the last argument written in the verifymsg file. If ***PROGRAM*** exits with nonzero status, the commit fails.

Run Control Files

There are a few files on the client (working copy) side that affect CVS's behavior. In some cases, they are analogs of repository administrative files; in other cases, they control behaviors that are appropriate only for the client side.

.cvsrc

This file specifies options that you want to be used automatically with every CVS command. The format of each line is

```
COMMAND   OPTIONS
```

where each ***COMMAND*** is an unabbreviated CVS command, such as **checkout** or **update** (but not **co** or **up**). The ***OPTIONS*** are those that you want to always be in effect when you run that command. Here is a common .cvsrc line:

```
update -d -P
```

To specify global options, simply use **cvs** as the ***COMMAND***.

.cvsignore

This file specifies additional ignore patterns. (See cvsignore in the "Repository Administrative Files" section earlier in this chapter for the syntax.)

You can have a .cvsignore file in your home directory, which will apply every time you use CVS. You can also have directory-specific files in each project directory of a working copy (these last apply only to the directory where the .cvsignore is located, and not to its subdirectories).

See **$CVSIGNORE** in the section "Environment Variables" later in this chapter for more on ignore processing.

.cvspass

This file stores passwords for each repository accessed via the **pserver** method. Each line is of the form:

```
REPOSITORY  LIGHTLY_SCRAMBLED_PASSWORD
```

The password is essentially stored in cleartext—a very mild scrambling is done to prevent accidental compromises (such as the root user unintentionally looking inside the file). However, this scrambling will not deter any serious-minded person from gaining the password if he or she gets access to the file.

The .cvspass file is portable. You can copy it from one machine to another and have all of your passwords at the new machine, without ever having run **cvs login** there. (See also the **login** and **logout** commands.)

.cvswrappers

This file is a client-side version of the cvswrappers file. (See the "Repository Administrative Files" section earlier in this chapter.) There can be a .cvswrappers file in your home directory and in each directory of a working copy directory, just as with .cvsignore.

Working Copy Files

The CVS/ administrative subdirectories in each working copy contain some subset of the following files.

CVS/Base/ (directory)

If watches are on, **cvs edit** stores the original copy of the file in this directory. That way, **cvs unedit** can work even if it can't reach the server.

CVS/Baserev

This file lists the revision for each file in **Base/**. Each line looks like this:

```
FILE/REVISION/EXPANSION
```

EXPANSION is currently ignored to allow for, well, future expansion.

CVS/Baserev.tmp

This file is the temp file for the preceding. (See CVS/Notify.tmp or CVS/Entries.Backup later in this section for further explanation.)

CVS/Checkin.prog

This file records the name of the program specified by the **-i** option in the modules file. (See the "Repository Administrative Files" section earlier in this chapter.)

CVS/Entries

This file stores the revisions for the files in this directory. Each line is of the form:

```
[CODE_LETTER]/FILE/REVISION/DATE/[KEYWORD_MODE]/[STICKY_OPTION]
```

If ***CODE_LETTER*** is present, it must be **D** for directory (anything else is silently ignored by CVS, to allow for future expansion), and the rest of the items on the line are absent.

This file is always present.

CVS/Entries.Backup

This file is just a temp file. If you're writing some program to modify the Entries file, have it write the new contents to Entries.backup and then atomically rename it to "Entries."

CVS/Entries.Log

This file is basically a patch file to be applied to Entries after Entries has been read (this is an efficiency hack, to avoid having to rewrite all of Entries for every little change). The format is the same as Entries, except that there is an additional mandatory code letter at the front of every line: An **A** means this line is to be added to what's in Entries, and **R** means it's to be removed from what's in Entries. Any other letters should be silently ignored, to allow for future expansion.

CVS/Entries.Static

If this file exists, it means only part of the directory was fetched from the repository, and CVS will not create additional files in that directory. You can usually clear this condition by using **update -d**.

CVS/Notify

This file stores notifications that have not yet been sent to the server.

CVS/Notify.tmp

This file is a temp file for Notify. The usual procedure for modifying Notify is to write out Notify.tmp and then rename it to "Notify."

CVS/Repository

This file stores the path to the project-specific subdirectory in the repository. This might be an absolute path, or it might be relative to the path given in **Root**.

This file is always present.

CVS/Root

This file is the repository—that is, the value of the **CVSROOT** environment variable or the argument to the **-d** global option.

This file is always present.

CVS/Tag

If there is a sticky tag or date on this directory, it is recorded in the first line of this file. The first character is a single letter indicating the type of tag: **T**, **N**, or **D**, for branch tag, nonbranch tag, or date, respectively. The rest of the line is the tag or date itself.

CVS/Template

This file contains a log message template as specified by the rcsinfo file. (See the "Repository Administrative Files" section earlier in this chapter.) It is relevant only for remote working copies; working copies on the same machine as the repository just read rcsinfo directly.

CVS/Update.prog

This file records the name of the program specified by the **-u** option in the modules file. (See the "Repository Administrative Files" section earlier in this chapter.)

Environment Variables

Following is a list of all of the environment variables that affect CVS.

$COMSPEC

This variable is used in OS/2 only; it specifies the name of the command interpreter and defaults to "CMD.EXE."

$CVS_CLIENT_LOG

This variable is used for debugging the client/server protocol. Set this variable to a file name before you start using CVS; all traffic to the server will be logged in filename.in, and everything from the server will be logged in filename.out.

$CVS_CLIENT_PORT

This variable is used in Kerberos-authenticated client/server access.

$CVSEDITOR

This variable specifies the program to use to edit log messages for commits. This overrides **$EDITOR** and **$VISUAL**.

$CVSIGNORE

This variable is a whitespace-separated list of file names and wildcard patterns that CVS should ignore. (See also the **-I** option to the **import** command.)

This variable is appended last to the ignore list during a command. The list is built up in this order: CVSROOT/cvsignore, the .cvsignore file in your home directory, the **$CVSIGNORE** variable, a **-I** command option, and finally the contents of .cvsignore files in the working copy used as CVS works in each directory. A "**!**" as the ignore specification at any point nullifies the entire ignore list built up to that point.

$CVS_IGNORE_REMOTE_ROOT

This variable is recently obsolete.

$CVS_PASSFILE

This variable tells CVS to use some file other than .cvspass in your home directory. (See .cvspass in the "Run Control Files" section earlier in this chapter.)

$CVS_RCMD_PORT

This variable specifies the port number to contact the rcmd daemon on the server side. (This variable is currently ignored in Unix CVS clients.)

$CVSREAD

This variable makes working copy files read-only on checkout and update, if possible (the default is for them to be read-write). (See also the **-r** global option.)

$CVSROOT

This variable specifies the path to the repository. This is overridden with the **-d** global option and by the ambient repository for a given working copy. The path to the repository may be preceded by an access method, username, and host, according to the following syntax:

```
[[:METHOD:][[USER@]HOST]:]/REPOSITORY_PATH
```

See the **-d** global option, in the section "Global Options" near the beginning of this chapter, for a list of valid methods.

$CVS_RSH

This variable specifies an external program for connecting to the server when using the **:ext:** access method. Defaults to **rsh**, but **ssh** is a common replacement value.

$CVS_SERVER

This variable specifies a program to invoke for CVS on the server side. Defaults to **cvs**, of course.

$CVS_SERVER_SLEEP

This variable delays the start of the server child process by the specified number of seconds. This is used only for debugging, to allow time for a debugger to connect.

$CVSUMASK

This variable specifies permissions for files and directories in the repository. (You probably don't want to set this; it doesn't work for client/server anyway.)

$CVSWRAPPERS

This variable is a whitespace-separated list of file names, wildcards, and arguments that CVS should use as wrappers. (See cvswrappers in the "Repository Administrative Files" section earlier in this chapter for more information.)

$EDITOR

See "**$CVSEDITOR**."

$HOME $HOMEDRIVE $HOMEPATH

These variables specifiy where the .cvsrc, .cvspass, and other such files are found (under Unix, only **HOME** is used). In Windows NT, **HOMEDRIVE** and **HOMEPATH** might be set for you; in Windows 95, you might need to set them for yourself.

Note

*In Windows 95, you might also need to set **HOME**. Make sure not to give it a trailing backslash; use **set HOME=C:** or something similar.*

$PATH

This variable is obsolete.

$TEMP $TMP $TMPDIR

These variables specify where temporary files go (the server uses **TMPDIR**; Windows NT uses **TMP**). Setting this on the client side will not affect the server. Setting this on either side will not affect where CVS stores temporary lock files. (See config in the "Repository Administrative Files" section earlier in this chapter for more information.)

$VISUAL

See "**$CVSEDITOR**."