CVS Crash Course

*From Wiki**3*

This is a very short introduction to CVS, the Concurrent Versions System. Further information should be obtained from the official sources and manuals.

Structure

File:CVSInteractions.png CVS defines a source repository that will be the hub for all the entities involved in a project. Each entity will request a module (in CVS parlance, a directory belonging to a tree controlled by the repository) to work on locally. Then, each local copy may be the object of several editing and management operations for keeping it in sync with the repository. Since changes are concurrent, compatibility problems may arise during the life of a local copy, until it again becomes synchronized with the central repository (see image).

The first contact with CVS may depend on the specific project and participants: the repository and/or the project may not yet exist (and have to be created), or they may both already exist (in which case, only check-out, update, and commit operations are performed).

The repository's location may be specified directly as an option to the "cvs" command:

```
cvs -d /root/of/the/cvs/repository some-cvs-subcommand
```

Or it may be read from the `CVSROOT` environment variable (defined, for instance, in the shell's configuration file):

```
export CVSROOT=/root/of/the/cvs/repository
cvs some-cvs-subcommand
```

Various modes of access are possible: local (i.e., direct access to the CVS repository's file system exists) or remote (otherwise). Remote access may be done through CVS's own protocol (pserver), but this is insecure. Another remote access method is the one specified by the `CVS_RSH` command (which defines the method for extended access methods). A common definition for this variable (in fact, the default in some systems) is "ssh":

```
export CVS_RSH=ssh
```

The CVSROOT variable now can be written as:

```
export CVSROOT=:ext:username@somemachine.somedomain:/root/of/the/remote/cvs/repository
```

The repository manages files by giving them revision numbers. Special tags may be used to group various files in various satges of evolution, creating snapshots of the repository's state for further reference. The default tag (corresponding to the most recent versions) is called HEAD.

Interaction with the repository is done by first performing a "check-out" operation (for obtaining a local copy) and, then, through a cycle of "update"/"commit" operations. A "release" operation may or may not be needed (when the local copy is no longer needed).

Creating the repository

The following command is almost never used and is presented here just for reference (we assume that the `CVSROOT` variable has been defined):

```
cvs init
```

This command creates a special control module (in the directory specified by the `CVSROOT` variable) called `CVSROOT`. This module will control how CVS manages each of the projects and files in the repository.

Creating a new module

A new CVS module may be created from a source tree (assumed outside CVS control) as explained below. We assume that the future module contents are in the `/some/directory/to/be/imported` directory.

```
cd /some/directory/to/be/imported
cvs import module-name vendor-tag initial-release-tag
```

Where *module-name* is the name of the new module (e.g. the project's name), *vendor-tag* is a label (e.g. a nickname of an acronym of the author or context), and *initial-release-tag* is the initial tag (e.g. "initial").

Note that the import operation does not automatically put the imported contents under CVS control: it is still necessary to check them out from the repository. Also, note that the import operation is recursive and that executing it without care may result in uploading trash (i.e., unwanted material) into the repository: ensure that the command is given *inside* the directory to be imported (as shown in the example).

If the module/project is already in the repository (even if it is empty), no import operation is needed (see the "add" operation below).

Checking out

The check out command creates a local copy of a module in the repository:

```
cvs co module-name
```

Updating a local copy

The update operation synchronizes a local copy with the contents of the module on the repository. The synchronization process may cause new files to be downloaded into the local copy, obsolete files to be removed and existing files to be changed. Of course, the process may run into difficulties if the local copy and the information coming from the repository cannot be merged without conflicts.

```
cvs update
```

CVS tells the user about the state of the update process by prefixing each file name (in the log of its actions) with a single character: U (new file coming from the repository), P (local file patched to reflect changes in the repository), R (local file scheduled for removal, but still exists in the corresponding repository revision), A (local file scheduled for addition, but not yet commited), C (a conflict was detected and marked: manual intervention is needed), ? (unknown local file). Other messages may also be given (for instance, telling about the removal of local files).

When solving a conflict, the user must first select the version to keep in the local copy and only then proceed to committing local changes.

Adding a local file

If, during the life of a module, a new file is added to the project, then the local file must be added to the repository. This is done by the following command:

```
cvs add name-of-local-file
```

Note that the command does not change the repository (it only marks the file for addition at the next commit operation).

Local directories must be added before their contents may be added. Adding a directory creates a local CVS subdirectory for keeping information about its contents.

Removing a local file

First and foremost: the CVS repository NEVER removes any files. This is because, even if a file becomes obsolete, there were past versions of the project in which the file exists and it must be possible to recreate it. Nevertheless, it is possible to mark it as deleted, so that it no longer appears in newer versions.

The command to schedule a file removal is:

```
cvs remove name-of-local-file
```

Note that CVS complains if the file still exists (this means that, for the previous command to work, the file must first be removed).

Committing local changes

The commit operation is, arguably, the most important, since it is the one that changes the repository's contents and makes the project evolve:

```
cvs commit -m "Message describing commit operation"
```

The commit operation allows the user to input a message that will be inserted in the log of changes made to the project. The commit operation is, by default, recursive and allows a different log message to be registered for each local directory in the project tree.

Other CVS commands

In addition to the above basic commands, other are available, for instance, for tagging revisions and inspecting the repository. Further information may be obtained in the manual:

```
man cvs
info cvs
```

Category: **Pages with broken file links**