

Introduction to CVS

Lena Bayeva

CS Tutors – Portland State University

October 22, 2005

1 About CVS

CVS is a client/server system. It is a very useful tool for programmers and software developers because it helps to backup various versions of projects and files, handles updates made by various users at different times, helps to manage the development cycle by allowing revisions' comments and viewing files' history or status. The usefulness of CVS is defined by its numerous features. It allows users to:

- keep everything in one place
- access projects' files from any account
- store all the versions of files' backups with minimum space usage
- document changes made to projects and files
- easily access older versions of source files
- view logs of changes

2 Setting CVS environment

First thing that needs to be done by CVS users is adding environment variables to their shells, so that CVS does it's job properly and doesn't cause any system conflicts. The following must be added by the users to their shells' rc files:

CVS_RSH - remote shell should be a secure shell - ssh:

- csh/tcsh → setenv CVS_RSH ssh
- sh family → export CVS_RSH=ssh

Note: These commands must be used because pserver necessary for working with CVS remotely isn't ran on our systems for security reasons.

3 Creating a repository

CVS repository is a "centralized area that stores projects' files, managed by the version control system and the repository administrator(user)". It contains information about projects and their versions.

The user first have to choose a location of the CVS repository, e.g. host - cs.pdx.edu and location on the file system - /u/your_login_name. Multiple repositories could be created in multiple locations.

1. Create a directory for your CVS repository:
`'mkdir /u/your_login_name/cvs_files/cvsrep'`
2. Set a \$CVSROOT environmental variable:
`'export CVSROOT=/u/your_login_name/cvs_files/cvsrep'`
3. Initialize your repository: `'cvs init'`

Or instead of steps 2,3 use -d option to specify repository location and initialize it: `'cvs -d $CVSROOT init'`

4 Working with CVS

Now it is possible to work with the repository (add and delete projects and revisions).

1. To be able to use CVS efficiently it is necessary to organize files appropriately: create a directory that will hold all the files related to the project. That directory then could be added to the repository. The command "import" does exactly that: it imports the entire directory into your source repository as a separate entity - module. You should run this command from the directory that holds a source for a new project:

```
'cvs import project name vendor name release name'
```

For example, a user with a login id user1 and a directory named Class1 will do the following:

```
'cd /u/user1/Class1'
```

```
'cvs import Class1 user1 alpha', where Class1 is a directory(project) name, user1 is a user name, alpha is a release tag.
```

All the files underneath Class1 will be added to the repository.

2. In order to use files and log in changes to the repository one needs to checked out a project from the repository. If you already have the original copy of a project, move it aside to backup your work (`'mv Class1 Class1.orig'`) and run this command:
`'cvs co Class1'` (co is short for checkout) in the directory you want Class1 to appear under. After this command CVS directory should appear inside of Class1.

The CVS directory inside of Class1 is a sandbox (working directory) that contains copies of versions of files from the repository. Users can create sandboxes on different hosts and in different locations of the file systems and check-in (commit) changes made to files from different locations and at different times. CVS takes care of

Before:

```
/u/user1/projects
|
|- Class1.orig
```

After:

```
/u/user1/projects
|
|- Class1.orig
|
|- Class
    |
    |-CVS
```

Figure 1: Directory hierarchies before and after checkout

conflicts and merges when users commit changes and update the same versions of files.

Sandboxes talk to the main repository (CVSROOT directory) when commits and updates happen and save and retrieve various versions of files from it when user specifies.

If the parent directory of your project is already a part of CVS, meaning it was previously checked out of CVS via the checkout command, then after importing the project into CVS repository, giving a full project's name (path to the new directory from the first directory in the hierarchy that's in CVS) and running the import command from the directory that needs to be imported (e.g. `cvs import Class1/Class2 user1 alpha`), run the update command to update the repository (it is recommended to move current files aside beforehand): `'cvs update -d'`.

Set of Commands for Adding Class1 to CVS

- `'cd /u/user1/projects/Class1/Class2'`
- `'cvs import Class1/Class2 user1 alpha'`
- `'cd /u/user1/projects/Class1'` (or `../` if your current directory is Class2)
- `'mv Class2 Class2.orig'`
- `'cvs update -d'` - this will checkout Class2 from the repository along with a sand box (you'll see CVS directory inside of Class2):

3. If the CVS repository is external and could be accessed via a remote shell utility, to accesses its content CVSROOT variable could be added to the local environment

Before:

```
/u/user1/projects
  |
  |- Class1
      |
      |-CVS
      |-Class2
          |
          |-
```

After:

```
/u/user1/projects
  |
  |- Class1
      |
      |-CVS
      |-Class2.orig
      |-Class2
          |
          |-CVS
```

Figure 2: Repository before and after adding

then commands on the repository can be run:

```
'setenv CVSROOT :ext:user1@cs.pdx.edu:/u/user2/cvsrep' (or /u/user1/cvsrep)
'cvs checkout Class1'
```

or to specify the CVSROOT for a onetime use, use the -d option with cvs: 'cvs -d :ext:user1@cs.pdx.edu:/u/user2/cvsrep co Class1'

4. To add files to the project that is already in CVS and has a sandbox in its directory, use the add command:
'cvs add file(s)'

Note: Files that you're trying to add with this command should already exist in the current directory that have to be created with the checkout command. To add a directory hierarchy to CVS use the import command described above.

5. To include changes that you have made to the current project to the source repository use the commit command:
'cvs commit'

If you want to commit to a particular revision -r option could be used:

```
'cvs commit -r4.0', where 4.0 is a revision number you are adding changes to.
```

Note: CVS will commit only files to which you've actually made some changes, it'll examine all files and directory hierarchies in the current project.

6. To merge your work with other revisions that have been done since the last update or checkout the update command should be used:

`'cvs update'`

It'll provide you with the most recent version composed of all the changes made to the project and saved (committed) to the repository.

7. To display information about the use of CVS commands on existing projects run the history command:

`'cvs history'`

Note: This command could be used with various options to display various information in different formats. For example, `-c` option displays a report on each time commit was used, and `-o` reports check-out modules. See man pages for more options.

8. To remove files from the repository use the remove command in the following way:

- a. erase files from the working directory only if you want to completely get rid of all your files (`'rm file(s)'`)

- b. `'cvs remove file(s)'` - this will remove files from the source repository

- c. `'cvs commit'` - to save changes (confirm the removal)

Note: The remove command is recursive by default. To avoid recursion specify `-l` option or specify just the files you actually want to remove.

9. Other commands:

`diff` - to show differences between files in the working directory and source repository.

`log` - to display log information

`release` - to delete working directory (this command doesn't affect the repository)

`rtag` - to specify symbolic tags for particular revisions

`status` - to display status of files in the working directory (e.g. latest version)

`tag` - to specify symbolic tag for files in the repository

10. For more information use the following resources:

- a. man pages (`'man cvs'`)

- b. CVS Pocket Reference, O'Reilly & Associates, Inc., copyright@2000

- c. <http://www-106.ibm.com/developerworks/linux/edu/l-dw-linuxcs-i.html> (you would have to register for a tutorial) or find a tutorial at www.ibm.com/developerWorks

- d. other links:

www.cvshome.org/docs

www.loria.fr/~molli/cvs/cvs-tut/cvs_tutorial_toc.html