

Introduction to CVS

Sivan Toledo
Tel-Aviv University

Goals of Source Management

- Ability to roll a project back if a bug was introduced
- Release tagging
- Multiple developers
 - Locking
 - Or concurrent updates with merging
- Branches to fix bugs in old releases

Non Goals

- Not a build/configuration system (make, autoconf/automake, MSVC projects, etc)
- Not a substitute for human management
 - Who does what (concurrent updates hurt)
 - When to commit changes, who gets notified
 - when to release, etc
- Not a substitute for testing, QA, bug tracking
- A defect in CVS: we want to track changes to several sources, but can't

The CVS Approach

- CVS=Concurrent Versions System
- Sources and their history are stored in a **repository**
- Modules are **checked out** by developers
- Developers work on their local copies and then **commit** changes to the repository
- Other developers can then **update** their local copies

The Repository

Creating a Repository

```
cv$ -d /home/stoledo/.../cvsroot init
```

- Creates an empty repository (will not delete things in an existing repository)
- -d argument is the root; can specify a default using CVSROOT environment variable

More on Repositories

- You can access a repository using
 - Direct access to files
`/home/stoledo/.../cvsroot`
`w:\cvsroot`
 - A remote shell
`setenv CVS_RSH ssh`
`:ext:dan@zoot.tau.ac.il/home/dan/cvsroot`
 - A CVS server (must be running)
`:pserver:anoncvs@anoncvs.us.lyx.org/cvs/lyx`
`cvs login` (type password, lyx in this case)

Basic CVS Usage

Starting a Project

- Create a directory with just the files you want to source-manage

```
images/ main.cpp richedit.pro richedit.ui richedit.ui.h  
richedit_he.ts
```

but not `richedit.cpp` since it was generated by designer

- `cd` to this directory & `setenv CVSROOT`

- Import the files and directories

```
cvs import -m "initial import" richedit sivan start
```

"initial import" is the log message for the operation

`richedit` is the path in the repository

`sivan` is a vendor tag (not really important)

`start` is a release tag

Checking Out a Project

- Move to a working directory
- `cvs checkout richedit`
- This creates a subdirectory `richedit` which contains a copy of the project
- Each project directory also contains a CVS subdirectory with CVS information; don't touch these files
- The CVS directories contains a pointer to the repository, so you don't need to specify `CVSR00T` any more

Build and Test

- Now `cd` to the project's working directory
- Try to build: `qmake richedit.pro; make` or whatever the build command is
- Hopefully this will work
- Run `cvs update` to update your sources
CVS will list files that are not up-to-date with respect to the repository, and in particular all the generated files (objects, etc) with a `?` prefix: it knows nothing about them

Modify and Commit

- Add some great feature to the program
- Run `cvcs update` to update your sources
 - Files that were updated by others but not by you will be marked with a **U** prefix
 - Files that you updated and were not touched in the repository, or were updated by others and the changes were successfully merged (changed to different lines) are marked **M**
 - Failed merges are marked **C**; will discuss these later

Modify and Commit (Cont.)

- let's assume there were not conflicts (C's)
- Build and test again if there were merges; perhaps the merges resulted in sources that don't compile or in bugs
- If everything works, `cvsc` commit
- Like most CVS commands, `commit` works recursively and will commit all your changes; You can commit one by one
- You'll need to supply a log message using `-m` or using a text editor

Revisions and Releases

Revisions of Files

- CVS takes a snapshot of each version of a file that is committed or added. These are called **revisions** and have numeric values, starting at 1.1 (next comes 1.2, etc)
- You can retrieve any old revision using
`cv update -r 1.4 main.cpp`
- CVS does not know about intermediate states in your working directory
- CVS stores revisions compactly

Releases and Tags

- You can assign a symbolic name, called a **tag**, to a particular revision of a file
- Particularly useful for tagging the current revisions of all files of a project that participate in a release of the product
- E.g., tag everything as belonging to release 2.0
`cvs tag -c release-2-0`
This tags everything and ensures (-c) that the version we tag (in the repository) is identical to the files in the working directory

More on Tags

- A single tag will correspond to a different revision of each file, e.g., to 1.4 of `main.cpp` and to 1.1 of `richedit.pro`
- You can checkout an old release
`cvs checkout -r release-2-0 richedit`
for example, to fix a bug in an old release of your software

Branches

- The revisions of a file need not be a simple chain; they can form a tree
- Suppose we are working on release 2.0 but a customer discovered a bug in release 1.0 that must be fixed
- We create a branch at the release 2.0 tag
- Release 2.0 revisions have two children
 - revisions in the chain leading to the 2.0 release
 - revisions that are 1.0 bug fixes

Branches and Merging

- CVS allows you to merge branches
- E.g., to port a new feature of 2.0 back into release 1.0 (perhaps a customer needs the feature but cannot upgrade for some technical reason)
- Don't count on this in your project planning: you'll have to resolve conflicts
- More on branches and merging in the manual; this is an advanced feature

Adding and Removing Files

Adding Files to the Repository

- `cvs add newclass.cpp`
- Not recursive! you can't `cvs add src/newclass.cpp`, you must be in that directory
- Still not in the repository
- `cvs commit newclass.cpp`
(`cvs commit src/newclass.cpp` will work)

Removing Files from the Repository

- `rm oldclass.cpp`
`cvs remove newclass.cpp`
- Will only remove non-existing files, or
`cvs remove -f oldclass.cpp`
- Still not in the repository
- `cvs commit`
- To rename, remove then add and commit

Removing Files from the Working Directory

- Use `cv s update -dP` to ensure that files that were deleted in the repository are deleted in your working directory
- and that empty directories are removed

Conflicts and How to Resolve Them

Conflicts

- If CVS discovers during update that your changes overlap changes made in the repository (relative to your revision), it will put both changes in the file and ask you to resolve

...

```
<<<<<<< main.cpp
```

```
    exit(error==0 ? SUCCESS : FAILURE);
```

```
=====
```

```
    exit(!!error);
```

```
>>>>>>> 1.5
```

file name

your code

new code

repository rev

Resolving Conflicts

- Select one version or the other, or write new code to replace the conflicting code
`exit(error==0 ? SUCCESS : FAILURE);`
- CVS will not commit the file until all the conflict markers are gone (<<<<<<< etc)
- As in other situations, it's best to avoid conflicts altogether
- CVS helps, but you still have to do the hard work

Text Files, Binary Files

CVS assumes that files contain text

- it automatically converts between line separators (`\n` in linux and unix, `\r\n` in windows, `\r` in MacOS)
- it assumes changes don't conflict if they are on different lines
- it performs keyword substitution `$Author$`, `$Revision$`, `Id`, etc
- it modifies your local copy to show conflicts and to allow you to resolve them

This Can Cause Trouble

- A binary file (e.g. an image or sound file) may become corrupted by line-separator substitution
- A binary file may become corrupted by keyword substitution; e.g., PDF files have pointers in them

Or May be Useless

- Some files are text, but are not meant to be edited by a text editor
 - Qt Designer .ui files; these are xml files (text), but are meant to be edited in Qt Designer
 - you probably won't be able to resolve the conflict in a text editor

Dealing with “Binary” Files

- Specify `-kb` flag to commands to prevent substitution and merging
- Better yet, specify as a sticky tag: a tag that gets propagated through revisions, and that gets used as a command option
`cv$ admin -kb images/textleft.png`
or
`cv$ add -kb images/textleft.png`

Other CVS Tools

Additional CVS Commands

- `cvadmin`
- `diff`
- `export`
- `history/annotate/log/status/
editors/watchers`
- `edit/unedit/watch`
- `rdiff`
- `rtag`
- `release`

.cvsrc

- A `.cvsrc` file in your home directory allows you to specify default options to `cv`s and its subcommands
- `cv`s `-z6`
update `-dP`
diff `-u`

Reservations and Notifications

- CVS supports reserved checkouts (locking) but not very well
`cvadmin -l`
- CVS supports notification about who is editing a file, when it is checked in, etc
`cvswatch on`
`cvsexit`
`cvsunedit`

CVS and other Source Management Systems

Other Systems

- SourceSafe: MicroSoft product, integrated with Visual Studio
- ClearCase: transparent, takes over file-system functionality, commercial
- BitKeeper: new, now used to manage Linux, both free and commercial licenses
- Aegis: free, better change control than CVS
- ...

CVS's Advantages

- Already installed on most Linux/Unix systems; means you can always grab a copy, fix it, and commit, even at a customer's site
- Available for Windows and MacOS
- Free
- Mature; used since approximately 1986
- GUI interfaces
- Web interfaces

That's it, folks