Using CVS

Concurrent **V**ersions **S**ystem - http://www.cvshome.org/

Jeremy Mates <jmates@sial.org>

The Road Map

1. What is CVS?
2. Anatomy of a Repository
3. Getting the Goodies
4. Working with Stuff
5. Issues & Caveats
6. Resources

What is CVS?

- CVS tracks document evolution in a hierarchical archive.
- Evolved from RCS.
- Leading Open Source Version Control (VC) system.
- Relatively Operating System agnostic.

Why do I need Version Control?

- Who broke **`foo.pl`**?
- When was **`foo.pl`** broken?
- Can we revert **`foo.pl`** to a working version?
- I want to develop my own **`foo.pl`** ...
- ... and merge my changes back in ...
- There's a bug in a old **`foo.pl`**, and the new **`foo.pl`** is still experimental... can we fix the old one for them?
- And more!

Anatomy of a Repository

- All CVS documents stored in a repository.
- Just a bunch of files and directories.
- Preferences in the repository, too:

```
$ ls -R /tmp/archive
/tmp/archive:
CVSROOT/

/tmp/archive/CVSROOT:
Emptydir/       config          editinfo,v  modules,v  taginfo
checkoutlist    config,v        history     notify     taginfo,v
checkoutlist,v  cvswrappers     loginfo     notify,v   val-tags
commitinfo      cvswrappers,v   loginfo,v   rcsinfo    verifymsg
commitinfo,v    editinfo        modules     rcsinfo,v  verifymsg,v
```

Creating a Repository

- A directory and a **`cvs`** command latter...

```
$ setenv CVSROOT /tmp/archive
$ mkdir -p $CVSROOT
$ cvs init
```

- CVS commands all begin with **`cvs`**, followed by a sub command to do something, e.g. '**`init`**' or '**`checkout`**'

- The **`init`** creates an empty, default repository in the specified path, given by **`$CVSROOT`**, or the **`-d`** option:

```
$ mkdir -p /tmp/archive
$ cvs -d /tmp/archive init
```

Connecting to a CVS Repository

- Use **`checkout`** (**`co`**) to obtain a working copy ("sandbox") of a "module" in the repository:

```
$ cd /tmp
$ cvs -Q checkout CVSROOT
$ ls CVSROOT
CVS/            loginfo
checkoutlist    modules
commitinfo      notify
config          rcsinfo
cvswrappers     taginfo
editinfo        verifymsg
```

- Modules are directories in the repository, or more...

More ways to get stuff:

- Can also obtain a module across a network:

```
client$ setenv CVS_RSH /usr/bin/ssh
client$ setenv CVSROOT \
     :ext:user@server:/tmp/archive
client$ cd /tmp
client$ cvs checkout CVSROOT
```

- Or via the CVS **pserver**, run from inetd(8):

```
client$ cvs -d \
     :pserver:user@server:/tmp/archive \
     checkout CVSROOT
```

Creating Initial Modules

- Use the CVS **`import`** command (annoying) to import existing sources.
- Or, checkout the entire repository, and use **`cvs add`** to create new modules as needed.
- Modules are just directories in the repository.
- Modules can also be groups of modules/files if one hacks up the **`CVSROOT/modules`** file.

Adding files

- The **`cvs add`** command also does files:

```
$ cd /tmp
$ cvs -Q checkout perl-scripts
$ cd perl-scripts
$ ls
CVS/
$ touch foo.pl
$ cvs add foo.pl
cvs add: scheduling file `foo.pl' for addition
cvs add: use 'cvs commit' to add this file permanently
$ cvs commit -m “Added empty foo perl script.” foo.pl
RCS file: /tmp/archive/perl-scripts/foo.pl,v
done
Checking in foo.pl;
/tmp/archive/perl-scripts/foo.pl,v  <--  foo.pl
initial revision: 1.1
done
```

Common CVS commands

- Use **`cvs commit`** to submit changed local files to the central repository.
- Use **`cvs update`** to synchronize local copy (“sandbox”) to repository.
- Use **`cvs diff`** to view differences between file versions.
- Certain utilities support the common cvs commands internally, e.g. the emacs VC Mode.

Issues & Caveats

- File & directory structure hard to change (plan well before adding new modules/files).
- Text orientation. Binary files are supported via '**`cvs add -kb logo.gif`**' but there is no "diff" support.
- Line orientation. Moved code is a delete from the source and an add elsewhere.
- Syntax oblivious. White space changes for formatting will be treated as a sweeping change.

Multiple Developers

- CVS uses an optimistic *merging model* to allow concurrent development.
- Can use **`edit`** and **`watch`** for more restrictive use.
- Communication is the key.

Advanced Stuff

- Blank templates can be created to base new development off of:

```
$ touch blank.pl; cvs add -kk blank.pl
$ cvs commit -m “Default perl script template added.”
```

- CVS can keep track of “tags” on files, to associate symbolic names (like “release-2001-02-27”) with a group of files.

- Branches off the main line of development can be done with tags, e.g. to apply a bugfix to a past release, or to develop off in an experimental direction.

Scripting Stuff

- CVS has good support for scripting, through various administrative files found under the CVSROOT module.
- CVS comes with some sample contrib scripts.
- Makefiles can also be inserted into the directory structure to automate various testing, building, and CVS commands:

```
TAGNAME = release
tag:
        @cvs tag -cfF $(TAGNAME)
```

Resources

- CVS Homepage: http://www.cvshome.org/
- Documentation central: http://www.cvshome.org/docs/
- Open Source Development with CVS: http://cvsbook.red-bean.com/
- CVS Pocket Reference: http://www.oreilly.com/catalog/cvspr/

Something broke!

- To revert a file to a previous revision, one must run log on the file to figure out which version was the last working one. This may involve committing a currently broken file first:

```
$ cvs log foo.pl | less

$ cvs diff -r 1.1 -r 1.2 foo.pl

$ cvs update -j 1.2 -j 1.1 foo.pl

$ cvs commit -m “Reverted bad 1.2 to 1.1.”
```

A few random commands...

- The cvs tag command can be used to mark a project that has just shipped:

```
$ cvs tag -fFc foo-project-2001-02-27
```

- To “tag” a modified file with a new revision, clearing the sticky bit that gets set afterwards:

```
$ cvs commit -r 2.0 foo.pl
$ cvs update -Ad
```