CVS – concurrent versions system

APRICOT 2008
Network Management
Taipei, Taiwan
February 20–24, 2008
Overview – what is CVS?

- CVS is a Version Control System (VCS)
• Part I
  – version control and change managements
  – introduction to CVS – principles, commands
  – examples
  – setting up a repository
  – accessing the repository
  – importing a project
  – creating modules
Contents – cont'd

● Part II
  – the CVSROOT/ directory and its files
  – pre- and post- jobs
  – the big picture: mail notifications, cvsweb, and lists
  – putting it all together
  – automated scenarios
Overview – what is version control

• Version control, and change management
  – Keep track of changes (revisions)
  – Share changes with others (public repository)
  – Maintain multiple versions of a same set of data (branches)

• What kind of data?
  – Source code
  – Documentation
  – Configuration files
  – Binary data as well (less efficient)
CVS terminology

• repository
  – Central, master copy containing all files being versioned. Directory structured

• working copy
  – Local copy of a project, checked out from a repository. Contains special directories (CVS) with information about which files are under CVS control, where they files come from and where they should be committed.

• module
  – A set of directories, files or other modules under a common “shortcut” name
CVS principles

- CVS uses a centralized “master copy”: the repository
- All work is done in a working copy
- Changes are committed back to the repository
- Special directory, CVS
CVS – the repository

- CVS is a centralized VCS (1 repository)
- The repository contains files in the RCS format, all ending in ' .v '
- Each RCS file contains a complete history, with changelog, of the file being versioned
- Well adapted to text files
- The repository is NEVER edited by hand
- A number of tools exist to analyze or browse the repository
  - cvsweb/webcvs
CVS – the repository

- Clients can access the repository locally or over the network.

- The repository is indicated (UNIX) using the CVSROOT environment variable:
  
  CVSROOT=
  - /cvs/myprojects # local disk
  - :pserver:myserver.com:/cvs/myprojects # via pserver
  - :ext:user@myserver.com:/cvs/myprojects # via SSH

- Allows for distributed work over LAN/WAN
CVS – example workflow

- **Initial checkout**
  - `cvs co projectname` initial checkout
  - `vi filename` ... work ...
  - `cvs commit [filename]` record changes

- **Later:**
  - `cvs up` update working copy from repository
  - `vi filename` ... work ...
  - `cvs commit [filename]` record changes
CVS – example workflow – cont'd
CVS clients

- Exist for most operating systems
  - cvs command line (UNIX, Win32)
  - TortoiseCVS – embeds in Explorer (Win32)
  - WinCVS (Win32)
  - ...

- Access the repository over the network or locally
CVS commands – action commands

- **import**
  - import a new project into an existing repository

- **checkout (co)**
  - check out a working copy of a project/file/module from the repository

- **update (up)**
  - update a working copy from the CVS version

- **commit**
  - commit changes back to the repository (incl. new files)
CVS commands – action commands cont'd

- add
  - add a new file in the working copy, ready to commit

- delete (del)
  - remove a file from the working copy, ready to commit
CVS command – status commands

- **status**
  - see the status and version of a given file or by default all files

- **diff**
  - show the difference between a given revision (by default: the last one) of the named file and the file in the working repository

- **log**
  - show revision history for one or more files
A working example

% CVSROOT=:ext:server.name:/data/cvs
% export CVSROOT
% cvs co someproject
Password: ******
cvs server: Updating someproject
U dir/file1
U dir/file2
...
% ls -l dir/
-rwxr-xr-x 2 regnauld staff  512 Dec 20 15:44  CVS/
-rw-r--r-- 1 regnauld staff 1244 Nov 17 14:21  file1
-rw-r--r-- 1 regnauld staff  341 Dec  3 21:04  file2
...
% vi file1
...
% cvs commit file1
A working example – cont'd

................. editor .................
/ Bugfix -- Modified file1 to fix bug
/
/ CVS: Enter Log. Lines beginning with `CVS:' are
/ CVS: removed automatically
/ CVS:
/ CVS: Modified Files:
/ CVS:  file1
/ CVS:----------------------------------------------
/
/tmp/cvsUABnYm: 8 lines, 290 characters
Checking in file1;
/data/cvs/dir/file1,v <-- file1
new revision: 1.2; previous revision: 1.1
done
%
What's in the CVS/ directory?

- **Entries**
  - existing files, and newly added files

- **Root**
  - where is the repository located

- **Repository**
  - name of module or path in the repository
The CVS $Id$ directive

- In an existing file, we add the following line
  
  $Id$

- Now cvs commit the file, and look at the file again
Anyone can create a repository, anywhere

Done using the `cvs init` command

Example:
- `mkdir /data/cvsrepo`
- `export CVSROOT=/data/cvsrepo`
- `cvs [-d /data/cvsrepo] init`
- `ls -l /data/cvsrepo`

```
drwxrwxr-x 3 pr staff 1024 Dec 20 15:45 CVSROOT/
```
Accessing the new repository

- **Locally**
  - `cvs -d /data/cvsrepo ...
    - Not necessary to specify -d if CVSROOT is defined

- **Remotely**
  - `cvs -d :ext:servername:/data/cvsrepo ...
  - SSH must be available!

- Ready for import!
Importing a new project...

% CVSROOT=/data/cvs; export CVSROOT
% cd someplace/myproject/
% cvs import my/new/project before_cvs start

..................... editor ........................
/ Import pre-CVS version of my new project           /
\                                                    \
/ CVS:---------------------------------------------- /
\ CVS: Enter Log. Lines beginning with `CVS:' are   \
/ CVS: removed automatically                         \
\....................................................
N my/new/project/file1
N my/new/project/file2
...
No conflicts created by this import
%

The location for this project in the repository is now my/new/project, under the /data/cvs repository i.e.:
  - /data/cvs/my/new/project

Let's test that we can check out the project:

% cvs co new/project
U my/new/project/file1
U my/new/project/file2
% cd my/new/project
% ls -l
...

• my/new/project is maybe too long as a project name
• solution: modules, which are shorter names for directories or groups of directories and other modules.
• For example:
  
  project  my/new/project
• With such a module defined, it will be possible to checkout, commit, etc... using the simple name “project”
  
  cvs -d :ext:/data/cvs co project
• We'll see how to define modules later.
The CVSROOT/ directory

- A default module is always created when one inits a repository: CVSROOT

  ```
  % cvs co CVSROOT
  U CVSROOT/checkoutlist
  U CVSROOT/commitinfo
  U CVSROOT/config
  U CVSROOT/cvswrappers
  U CVSROOT/editinfo
  U CVSROOT/loginfo
  U CVSROOT/modules
  U CVSROOT/notify
  U CVSROOT/rcsinfo
  U CVSROOT/taginfo
  U CVSROOT/verifymsg
  ```
The CVSROOT/ directory – cont'd

- Files described in cvs(5)
  - man 5 cvs

- Most relevant:
  - modules define modules
  - commitinfo pre-commit scripts
  - cvswrappers handle special files
  - loginfo post-commit scripts
Pre- and post- jobs

- Using commitinfo and loginfo, it is possible to have automatic jobs run before and after each commit, for instance:
  
  - **pre-commit stage (commitinfo)**
    - verify that a user is allowed to modify a given file
    - check syntax for a file
    - ...
  
  - **post-commit stage (loginfo)**
    - send update as a mail
    - append it to a log
    - ...
The big picture: mail, cvsweb, lists
Putting it all together...
CVS shortcomings

- symlinks and ownership of files are not recorded
- no renaming of files (copy + delete)
- no changesets
  - each file has 1 version, need postprocessing work to figure out “all files for this commit”
- no disconnected operation
  - add, remove, commit, ... all need access to the server
- branching/merging is quite complicated
Automated scenarios

• Idea: automatize configuration management tasks so that configuration files are automatically versioned using CVS...

• ... even when the sysadmin forgets :)

• Implementation – cron job
  – look at all files in a given directory
  – if they exist in the repository already -> commit
  – if they don't, add, then commit
Automated scenarios – cont'd

- Already exists for network equipment: RANCID
  - http://www.shrubbery.net/rancid/

- Simple concept to implement for all relevant files in /etc

- Subscribe all admins to the alias / mailing list, so everyone receives a notification when a change takes place – whether planned or not!
References

- http://www.nongnu.org/cvs/
- http://cvsbook.red-bean.com/
- http://www.tortoisecvs.org/