Introduction to Concurrent Versions System

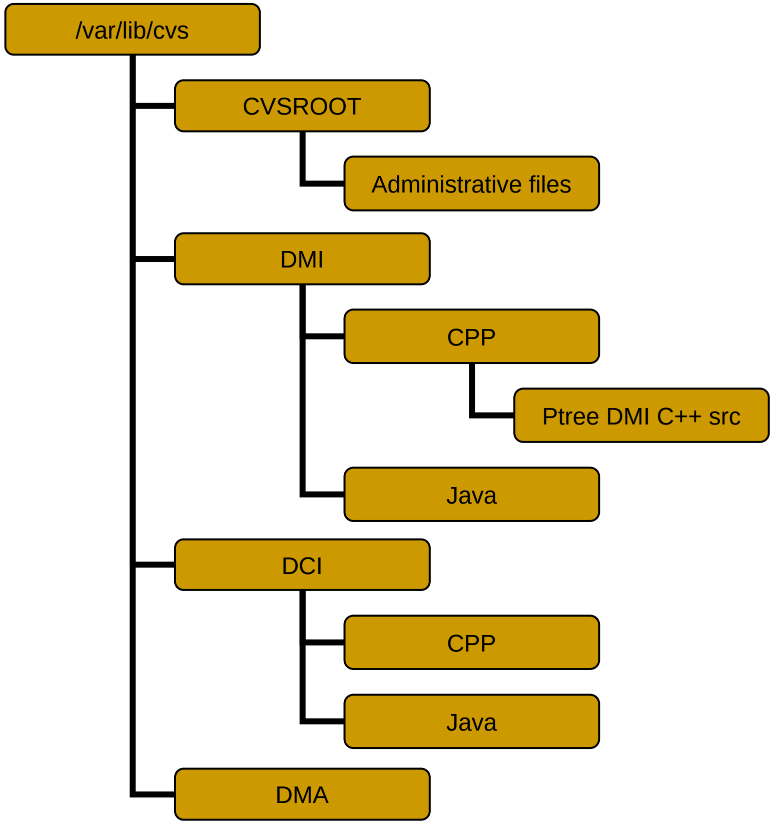
Overview

- Conceptual Overview
- A typical work session
- Revisions
- Branching and Merging
- Multiple developers
- How to start to use our CVS server
- CVS Resource

Conceptual Overview

- What is CVS?
  - CVS is a version control system. It is used to record the history of your source files.
  - CVS also helps you if you are part of a group of people working on the same project
- What is CVS not?
  - Not a build system
  - Not a substitute for management
  - Not a substitute for developer communication
- Why use CVS?
  - Bugs can creep in when software is modified, and may not be detected until a long time after the modification is made. With CVS, we can retrieve old versions to find which change caused the bug
  - CVS can also help when a project is being worked on by multiple people, where overwriting each others changes is easy to happen
    - CVS solves this problem by having each developer work in his/her own directory and then instructing CVS to merge the work when each developer is done.

Conceptual Overview (Contd.)



- CVS repository structure

Conceptual Overview (Cont'd.)

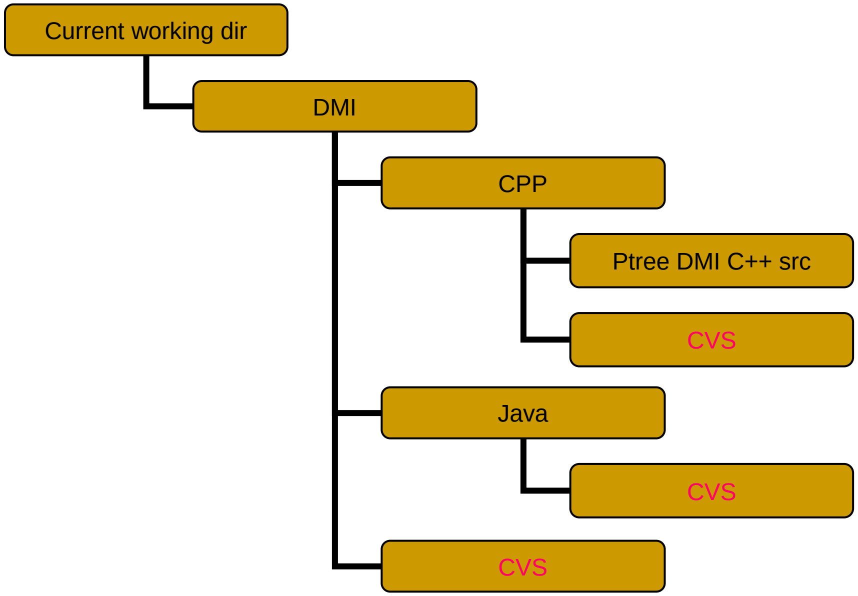
- CVS repository stores a complete copy of all the files and directories which are under version control.
- CVS can access a repository by a variety of means.
- Use cvs command to perform all the repository operations. Don’t operate repository directly!
- CVSROOT contains some administrative files
  - modules file is the most important one, which can be use to define all modules in the repository.
  - We can group out source files into modules
    - Module1 file1, file2, file3
    - Module2 file4, file5
    - Module-n file6, file7, file8, file9

A typical work session

- Some environment variables involved (BASH style)
  - CVSROOT (three ways to access CVS repository)
    - CVSROOT=/var/lib/cvs
    - CVSROOT=:pserver:user@hostname:/var/lib/cvs
      - CVS_AUTH_PORT
        - $CVS_AUTH_PORT=2401
    - CVSROOT=:ext:user@hostname:/var/lib/cvs
      - CVS_RSH
        - $CVS_RSH=ssh
  - CVSEDITOR
    - $CVSEDITOR=/usr/bin/vim
  - Don’t forget to run export!

A typical work session (Contd.)

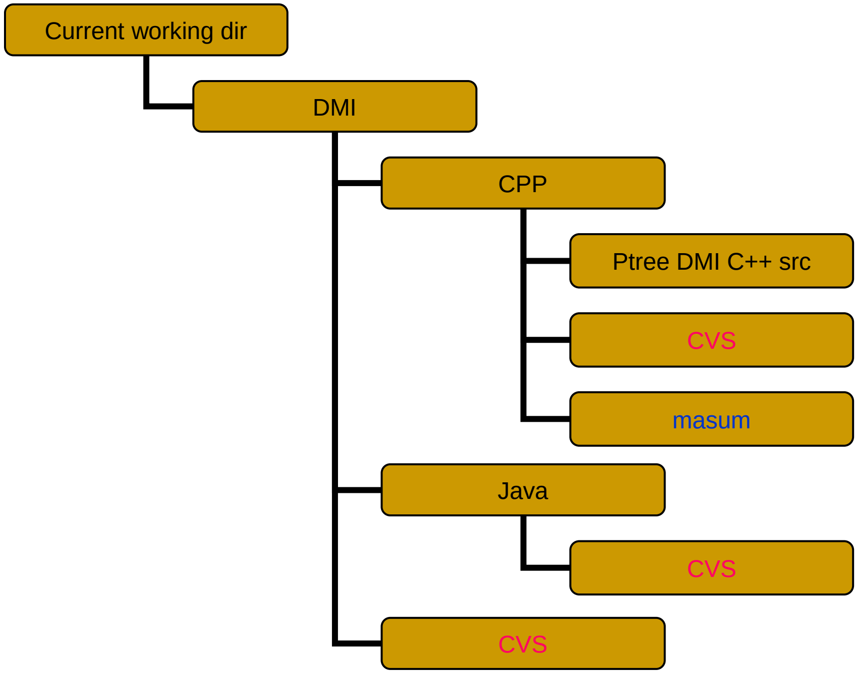
- Before start
  - Generally, using a remote repository is just like using a local one, except that the format of the repository name is different
  - Using “pserver”
    - $cvs login
- Get your own working copy
  - $cvs co DirName|ModuleName
  - $cvs co DMI



Working copy directory structure
The `CVS' directory is used internally by CVS.

A typical work session (Contd.)

- Add new file or dir
  - $cvs add DirName|FileName
  - $cvs commit DirName|FileName
  - $cvs commit –m “log info” DirName|FileName
  - Example:
    - cd DMI\CPP
    - mkdir masum
    - $cvs add masum
    - $cvs commit masum



A typical work session (Contd.)

- Clean up
  - Clean up working repository
    - $rm –rf dirName
    - $cvs release –d dirName|FileName

A typical work session (Contd.)

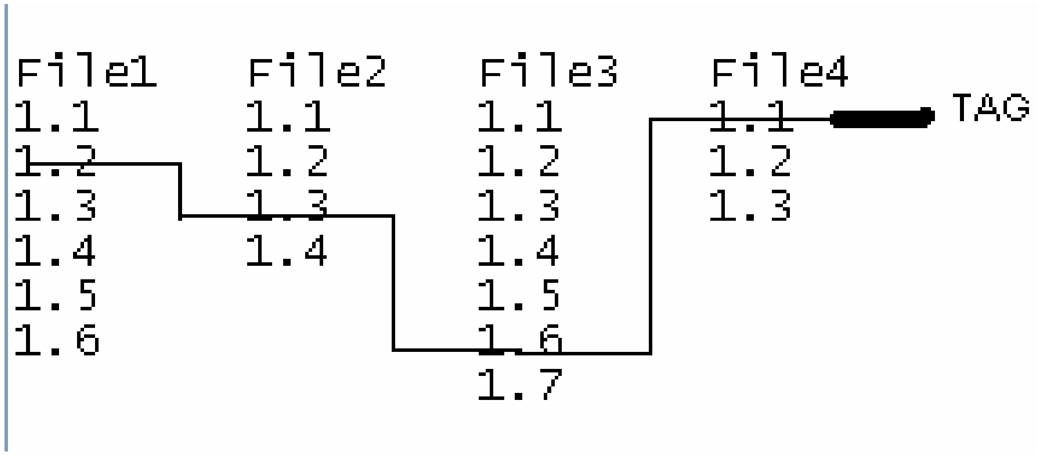
- View difference
  - $cvs diff –r ver1 –r ver2 fileName
- History browsing
  - $cvs log
  - $cvs history
- View modules
  - $cvs checkout –c
- View file status
  - $cvs status filename …

Revisions

- Revision numbers
  - Look like 1.1 -> 1.2 -> 1.3 -> 1.4
  - By default, CVS will assign numeric revisions by leaving the first number the same and incrementing the second number.
  - To bring all your files up to revision 3.0 (including those that haven't changed), you might invoke:
    - $ cvs commit -r 3.0
- Tags-symbolic revisions
  - A symbolic name to a certain revision number of a file
  - Example:
    - cd /DMI/C++
    - $cvs tag ptree-first-stage .
    - $cvs checkout -r ptree-first-stage

Revisions (Contd.)

- When we tag more than one file with the same tag, you can think about the tag as a handle.
- When you pull on the handle, you get all the tagged revisions.



File1	File2	File3	File4
1.1	1.1	1.1	1.1
1.2	1.2	1.2	1.2
1.3	1.3	1.3	1.3
1.4	1.4	1.4	
1.5		1.5	
1.6		1.6	
		1.7	

Branching and Merging

- Why branching? To maintain several versions at the same time, e.g. one developing version and one stable version.
- Create a branch, assuming you're in a working copy:
  - $ cvs tag -b rel-1-0-patches
- Create a branch without reference to any working copy, by using rtag:
  - $ cvs rtag -b -r rel-1-0 rel-1-0-patches tc

Branching and Merging (Contd.)

- You can merge changes made on a branch into your working copy by giving the `-j *branchname*' flag to the update subcommand.
  - $ cvs update -j R1fix m.c
  - $ cvs commit -m "Included R1fix"
- A conflict can result from a merge operation.

Multiple developers

- What’s the problem?
- Two solutions
  - Reserved checkouts
    - Allow ONLY one person to edit each file at a time
    - Very counter-productive
    - $cvs admin -l
  - Unreserved checkouts (default)
    - Allow more than one person to edit their working copy of a file simultaneously
    - What will happen using this solution?
    - CVS provides mechanisms to facilitate the communication without actually enforcing rules like reserved checkouts do

Multiple developers (Contd.)

- How to use unreserved checkouts?
  - Check file status before “commit” changes
  - When you want (need) to update or merge a file, use the update command.
  - Your modifications to a file are never lost when you use update. If no newer revision exists, running update has no effect. If you have edited the file, and a newer revision is available, CVS will merge all changes into your working copy.
    - All non-overlapping modifications are incorporated
    - And the overlapping section will cause conflict

Multiple developers (Contd.)

- You can resolve the conflict by editing the file, removing the markers and the erroneous line.
  - overlapping section is marked with `<<<<<<<', `=======' and `>>>>>>>'.
- Then go ahead and commit this file as a new revision into the repository again.

Multiple developers (Contd.)

- Mechanisms to track who is edition files
  - Tell CVS to watch certain files
    - $cvs watch on files
    - $cvs watch off files
  - Tell CVS to notify you
    - $cvs watch add [-a action] files
    - $cvs watch remove [-a action] files
  - How to edit a file which is being watched
    - $cvs edit files
    - $cvs unedit files

Multiple developers (Contd.)

- Information about who is watching and editing
  - $cvs watchers files …
  - $cvs editors files …

How to start to use our CVS server

- Remotely access:
  - pserver (Using RSH):
    - CVSROOT=:pserver:username@midas2.cs.ndsu.nodak.edu:/var/lib/cvs
  - ext (using an external rsh program)
    - CVSROOT=:ext:username@midas2.cs.ndsu.nodak.edu:/var/lib/cvs
    - CVS_RSH="ssh"
- export CVSROOT CVS_RSH

How to start to use our CVS server

- Suggest to use module name instead using the directory name directly.
  - I may need to know files you are working on
- Configure files for using our cvs server: http://www.cs.ndsu.nodak.edu/~datasurg/kddcup/darron/cvs_config_files/
  - .bashrc
  - .bash_profile
  - Download the file, merge them to your original .bashrc and .bash_profile using your favorite editor
  - Before try any cvs command, run appropriate alias

CVS Resource

- Get CVS manual
  - man cvs
- CVS Links
  - CVS Home: http://www.cvshome.org/
  - http://www.cvshome.org/new_users.html
  - http://www.loria.fr/~molli/cvs-index.html
  - http://cvsbook.red-bean.com/cvsbook.html
  - http://www.loria.fr/~molli/cvs/cvs-FAQ/cvsfaq0.html
  - http://sfsetup.sourceforge.net/tutorial_index.html
- Mailing List:
  - Info-cvs: info-cvs-requests@gnu.org