CS395T: Introduction to Scientific and Technical Computing

CVS

Instructors

Dr. Karl W. Schulz, TACC
Dr. Victor Eijkhout, TACC

Outline

- HW #1 Update
  - Expect to return on Thursday
- HW#2 is now Posted
  - You will need to login to Lonestar to complete
  - Submit results electronically on Blackboard
  - Due in 1 Week, 10/9/2007
- Source Code Control

Ranger Update

Source Control and Versioning

- Codes evolve over time
  - sometimes bugs creep in
  - sometimes the old way was right
  - sometimes it's nice to look back at the evolution
- How can you get back to an old version?
  - keep a copy of every version of every file
    - disk is cheap, but this could get out of hand quickly
    - is a huge pain to maintain
  - use a tool

Some Tools

- Free
  - RCS – Revision Control System
  - CVS – Concurrent Versions System
  - SVN – Subversion
  - ...
- Commercial
  - MS Visual SourceSafe
  - ClearCase
  - MKS
  - ...

Two Models of Source Control

- A central repository holds the files in both models
- Read/Write Local Workspaces and Merging
  - every developer has a local copy of the source files
  - everybody can read and write files in their local copy
  - conflicts between simultaneous edits handled with sophisticated merging algorithms or manually when files are synched against the repository or committed to it
- Read-only Local Workspaces and Locks
  - every developer has a read-only local copy of the source files
  - individual files are checked-out as needed and locked in the repo in order to gain write access
  - unlocking the file commits the changes to the repo and makes the file read-only again

CVS

- Started with some shell scripts in 1986
- Recoded in ’89
- Evolving ever since (though mostly unchanging now)
- Uses r/w local workspaces and merging
- Only stores differences between versions
  - saves space
  - basically uses *diff(1)* and *diff3(1)*
- Works with local repositories or over the net with rsh/ssh

CVS and the Environment

- *$CVSROOT* gives the path to the repository
  - *\/path/to/repo* for local repositories
  - *username@host:/path/to/repo* for remote
- *$CVS_RSH* tells CVS which remote shell command to use to talk to the remote machine
  - should almost always be set to *ssh*
- Both of these should generally be set before using *cvs*

Repositories

- CVS stores files in a central location called a repository
- You can create a repository anywhere you want
- Generally, this should be
  - accessible to your collaborators
  - backed up daily
  - sufficiently large
- You don’t usually need more than one repo
- If you’re lucky, your site or IT staff will provide a CVS repository already
- Repos are collections of modules, and modules contain your source code

Invoking CVS

```
cvs [cvs-options] command [command-options-and-arguments]
```

- The first set of options are common to all *cvs* commands
- The second set are command specific
- Some commands
  - init
  - checkout
  - update
  - commit
  - diff
  - add
  - log
  - tag
  - status
- Commands are usually recursive
  - current dir or its immediate subdirs must be CVS local working directories, or CVS will stop recursing

Creating a Repository

- Set *$CVSROOT* and *$CVS_RSH*

```
lslogin1$ cvs init
```

- You can also specify the repository on the command line

```
lslogin1$ cvs –d
  istc00@lonestar.tacc.utexas.edu:/home
  /utexas/istc/istc00/cvs import
```

Preparing to Import a New Module

- Clean up your source directory
- CVS isn’t really designed for object files, exectuables, binary files, etc.
  - though it can deal with them if it has to
- It’s designed for text files
  - source
  - scripts
  - *Makefiles*
  - text documents
  - etc.

Importing

```
lslogin1$ cd mydir
lslogin1$ cvs import myrepodir vendor_tag release_tag
```

- Creates a directory in *$CVSROOT/myrepodir*
- Pulls in all of the files in *mydir*
- Applies two tags that can be used to get back to this exact version later
- Does not modify anything in *mydir*
  - Note: you have to check out a CVSified working copy to begin working on this source under CVS control

Checking Out a Module

```
lslogin1$ cd ..
lslogin1$ cvs checkout myrepodir
```

- Creates a directory called *myrepodir*
- Pulls down a copy of all the files in the repository in the *myrepodir* module
- Adds a directory for CVS's use called *myrepodir/CVS*
  - no reason to modify these files in most cases
  - ... but there are occasions when you will want to do so

```
lslogin1$ cvs co myrepodir
```

- Most CVS commands have a shorter name *'co'* is short for *'checkout'*

Ignoring Certain Files

- In any given module, for each directory in that module, *.cvsignore* (notice the dot in the name) lists files and glob(7) patterns of files to be ignored
  - one file or pattern per line
  - every directory should get one that ignores tilde files, object files, and a.out

.cvsignore:

```
*~
.*~
*.o
a.out
```

Updating to the Latest

- Usually, before adding anything, or checking in your changes, you should synchronize the state of your local copy with things that may have changed in the repository version

```
lslogin1$ cvs update
lslogin1$ cvs up
```

Update

- Downloads new files
  - use *cvs up –d* to get new directories as well
- Downloads any differences in files between repo and the local copy
  - merges changes where possible
  - marks files that can’t be merged
    - these need manual conflict resolution
- Marks locally modified files
- Marks locally added files
- Lists unknown files and directories
  - these probably need to be added or listed in .cvsignore

Update Codes

- U FILE
  - FILE was updated, whole file was downloaded
- P FILE
  - FILE was updated, only a “patch” was downloaded
- A FILE
  - FILE locally added with *cvs add*
  - this change needs to be committed
- M FILE
  - FILE has been locally modified
  - differences may have been successfully merged
    - if so, a backup of your local file is made
    - a message is printed telling you about it and giving the name of the backup

Update Codes

- C FILE
  - FILE has conflicts that require human intervention
  - CVS makes a backup of your original: .#FILE.REVISION
- ? FILE
  - CVS doesn’t know anything about this file
    - add it
    - add it to .cvsignore
- R FILE
  - FILE has been removed with *cvs remove FILE*
  - this change needs to be committed

Adding Files

```
lslogin1$ cvs add somefile
```

- Tells CVS to start tracking this file
- Does not actually upload to the repo
  - *somefile* is simply marked for addition to the repo
  - *cvs commit* needed to actually upload the file

Committing Changes

- Local modifications, added files, and removed files have to be committed to the repo

```
lslogin1$ cvs commit
lslogin1$ cvs ci
```

- Tries to update first
- If it finds conflicts, it will error and ask you to fix the conflicts before trying to continue

Commit

- CVS uses the *$EDITOR* environment variable to invoke your favorite editor to allow you to enter a log message
  - if *$EDITOR* is empty, it invokes *vi(1)*
  - if you don't like *vi(1)*, you should set this variable to something else

    ```
    lslogin1$ export EDITOR=emacs
    ```

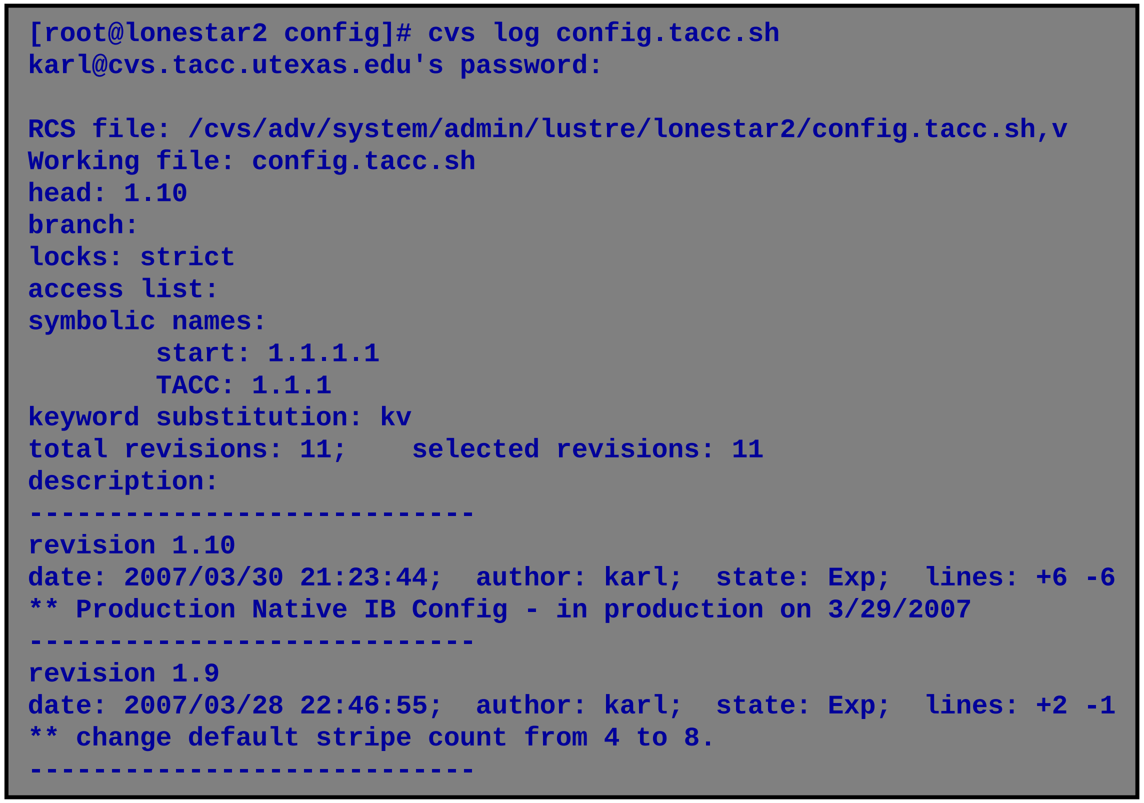
- You can specify the log message on the command line with the *–m* option to *commit*

```
lslogin1$ cvs ci –m “some message”
```

Viewing the Log Messages

```
lslogin1$ cvs log somefile
```

- Shows you
  - all the past log messages
  - who made the commit
  - version numbers
  - total revisions
  - etc.
- Doesn’t change anything



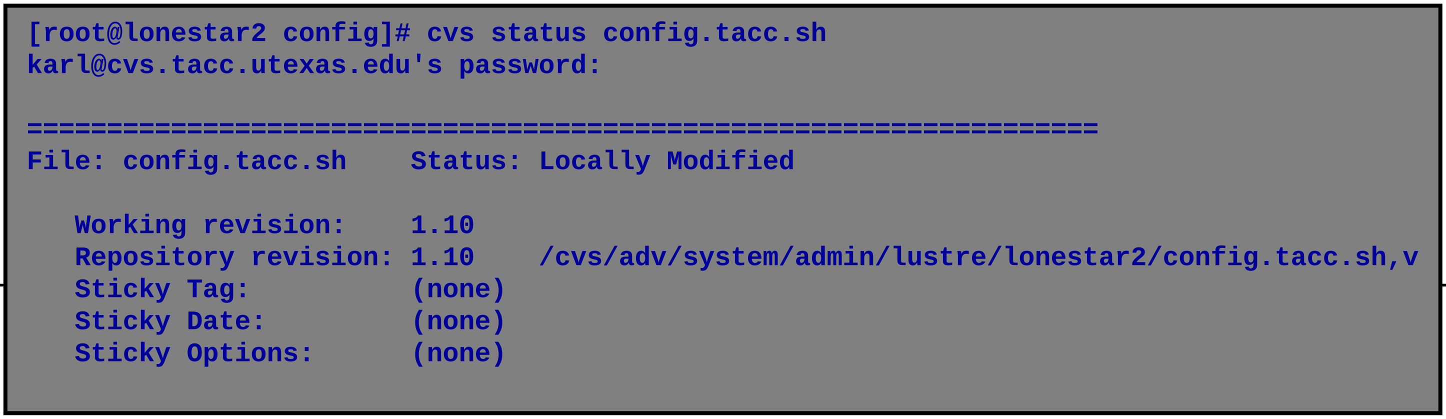
```
[root@lonestar2 config]# cvs log config.tacc.sh
karl@cvs.tacc.utexas.edu's password:

RCS file: /cvs/adv/system/admin/lustre/lonestar2/config.tacc.sh,v
Working file: config.tacc.sh
head: 1.10
branch:
locks: strict
access list:
symbolic names:
        start: 1.1.1.1
        TACC: 1.1.1
keyword substitution: kv
total revisions: 11;    selected revisions: 11
description:
----------------------------
revision 1.10
date: 2007/03/30 21:23:44;  author: karl;  state: Exp;  lines: +6 -6
** Production Native IB Config - in production on 3/29/2007
----------------------------
revision 1.9
date: 2007/03/28 22:46:55;  author: karl;  state: Exp;  lines: +2 -1
** change default stripe count from 4 to 8.
----------------------------
```

Checking Status

```
lslogin1$ cvs status somefile
```

- Gives you
  - the repo version number
  - the local version number
  - state of the file
  - any current “sticky” tags
- Changes nothing



```
[root@lonestar2 config]# cvs status config.tacc.sh
karl@cvs.tacc.utexas.edu's password:

===================================================================
File: config.tacc.sh    Status: Locally Modified

   Working revision:    1.10
   Repository revision: 1.10    /cvs/adv/system/admin/lustre/lonestar2/config.tacc.sh,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Statuses

- Up-to-date
- Locally Modified
- Locally Added
- Locally Removed
- Needs Checkout
- Needs Patch
- Needs Merge
- Unresolved Conflict (two files w/ the same name)
- File had conflicts on merge
- Unknown

Resolving Conflicts

- Open the file that is listed as having a conflict
- Search for the line that starts with “<<<<<<< thisfile”
  - lists the local file name after the “<”s
  - following lines are the local version of the code
- A line that starts with “=======” marks the transition
  - the repo code follows
- A line with lots of “>>>>>>> x.y” ends this conflict section
  - lists the repo version number after the “>”s

Resolving Conflicts

- Decide what code actually needs to appear in this section
- Remove the “<”, “=”, and “>” lines
- Rerun update and commit
- CAVEAT CVS will accept anything that’s in the file once you change the date of the file
  - conflict markers might be legit in your code, so be careful
  - it will warn you if you leave the conflict markers in, but will do the commit anyway

Removing Files

- If you remove a file, and then do an update, CVS will get you a fresh copy from the repo
- To remove a file from the repo, you must
  - remove the file locally
  - then ask CVS to remove it with the remove command
  - then commit the removal

```
lslogin1$ rm somefile; cvs remove somefile
lslogin1$ rm somefile; cvs rm somefile
```

- The file isn’t actually removed, you can get it back by asking for an older version of the repo
  - CVS remembers the name, so if you add a file back, it pulls in the old history

Showing Differences

`lslogin1$ cvs diff somefile`

- Shows the difference between the local copy and the repo version
  - differences are shown in the format used by *diff(1)*
  - many *diff(1)* options can be applied to *cvs diff* to affect the formatting

`lslogin1$ cvs diff –r tag somefile`

- Shows the difference between the local copy and some other version specified by the *tag*

```
[root@lonestar2 config]# cvs diff config.tacc.sh
karl@cvs.tacc.utexas.edu's password:
Index: config.tacc.sh
===================================================================
RCS file: /cvs/adv/system/admin/lustre/lonestar2/config.tacc.sh,v
retrieving revision 1.10
diff -r1.10 config.tacc.sh
51a52,54
> ${LMC} -o $config --add net --node ls2-mds  --nid ls2-mds  --nettype $LOCALNET
> ${LMC} -m $config --add net --node ls2-mds1 --nid ls2-mds1 --nettype $LOCALNET
>
60a64
>     ${LMC} -m $config --add net --node data-0-$i --nid data-0-$i --nettype $LOCALNET
```

Revision Numbering and Tags

- CVS numbers every version of every file you commit
  - each file gets its own revision number history
  - numbering starts at 1.1
  - usually goes 1.1, 1.2, 1.3, ...
  - ... but other things are possible
  - generally advisable not to change CVS's numbering strategy
- You can use *cvs ci –r x.y* to specify the revision number
  - but I wouldn't unless you know what you're doing
  - *-r* can be used with other commands to get access to a particular file's revision (like before with *cvs diff)*

Tags

```
lslogin1$ cvs tag sometag
```

- Applies the symbolic tag *sometag* to every file
  - modifies the repo directly
  - associates revision numbers with *sometag*
  - uses the current revision number
  - becareful if there are locally modified files
    - CVS won’t warn you
    - the repo version is the thing that gets tagged
  - cannot use ‘$,.:;@’ in your tag names
    - I use: vX_Y for version X.Y
    - i.e. if this is version 3.4 of my code, I tag it with v3_4
- Symbolic tags can be used as arguments to *-r*

Using Tags

```
lslogin1$ cvs co –r sometag –d
  someotherdir somemodule
```

- Checks out a fresh copy of an older version of the module tagged with *sometag*
- This tag is “sticky”
  - CVS commands will all operate under this tag
  - *cvs up –A* will clear all sticky tags and merge the local versions with the most current repo versions

When to Commit?

- Committing too often leads may leave the repo in a state where the current version doesn’t compile
- Committing too infrequently means that collaborators are wait for your important changes, bugfixes, etc. to show up
  - makes conflicts much more likely
- Common policies
  - committed files must compile and link
  - committed files must pass some test
- Come to some agreement with your collaborators about the state of the repo

Cute Tricks in Your Code

- You can keep track of the CVS revision automatically in all your source code files
- Just add `“$Id: $”` into a comment in your file
- This gets expanded when you check in/out the relevant file to include the revision, last update date, and the user making the changes
- For example:

```
/* $Id: lsf_showq.c,v 1.26 2007/10/02 18:00:52 karl Exp $ */
```

Cute Tricks in Your Code

- You can also include a record of the log history if you want
- Add `“$Log: $”` into the desired location
- Note that you only get log entries from the point at which you add the CVS Log entry

- Here is an example from a shell script

```
# $Log: mvapich-intel.spec,v $
# Revision 1.1  2007/06/22 09:35:06  karl
# ** Initial standard build of mvapich for Intel
#
```

Subversion

- Subversion is a Functional superset of CVS (so if you learn cvs, you can also function in subversion)
  - Directory versioning (rename and moves)
  - Atomic Commits
  - File meta-data
  - True client-server model
  - Cross-platform, open-source

Subversion Architecture

- Each working copy has a .svn directory
  - *Similar to the CVS's CVS directory*
  - *Stores metadata about each file*
- Local or Network Repository
- Network access over HTTP or SSH
- Encrypted authentication
  - *Cleartext password stored in ~/*.subversion
- Fine-grained authorization
- Command line client is svn

CVS vs. Subversion

- Most CVS commands exist in SVN
  - *Checkout, commit, update, status…*

- Arguments position matters in CVS

```
$ cvs -d /cvsrootupdate –d
```

- Not so in SVN

```
$ svn log -r 123 foo.c
$ svn log foo.c -r 123
```

Revisions

- Revision numbers are applied to an object to identify a unique version of that object.
- CVS
  - Revision numbers are per file.
  - No connection between two files with the same revision number.
  - A commit only modifies the version number of the files that were modified.

    foo.c rev 1.2 and bar.c rev 1.10.

  - After commit of bar.c:

    foo.c rev 1.2 and bar.c rev 1.11.

Revisions (cont)

- Revision numbers are applied to an object to identify a unique version of that object.
- SVN
  - *Revision numbers are global across the whole repository.*
  - *Snapshot in time of the whole repository.*
  - *A commit modifies the version number of all the files.*

    foo.c rev 25 and bar.c rev 25.

  - *After commit of bar.c:*

    foo.c rev 26 and bar.c rev 26.
    Subtle Note: *foo.c rev 25 and 26 are identical.*

References

- CVS FAQ: http://ximbiot.com/cvs/wiki/index.php?title=CVS_FAQ
- CVS Manual http://ximbiot.com/cvs/manual/