Intro to CVS

Concurrent Versions System
Overview

- What CVS is and why we need it
- Using CVS in the department
- Basic Commands to get you started
- Remote use and front-ends
- Integration with other software
The Problem

- We all have work we don’t want to lose
- Backup?
- Increasing number of computers available to us
- Ubiquitous network connections
- Copying is dangerous, especially in UNIX and variants
- Version System
Why use version control?

- Clear case with multiple developers
- Case for single person:
  - You work on some code in lab
  - Then you take a copy and work on that at home
  - Then you do some more work in the lab
  - Then you realise what you’ve done and have to combine both changes. Only, you can’t remember which came first…
What is this CVS thing?

- Version Control System
- Client/Server Architecture
- Supports multiple people working on same code
- Supports remote and distributed working
- Software Development
- Website Management
- Documentations
- Anything based on plain text
How it works (overview)

• Central Repository and Working Copy
• Server works in all types of UNIX, clients available for most OSs
• Work loop:
  – Check-out
  – Edit
  – (Merge if necessary)
  – Commit changes
Example CVS Session

• `cd workingdir`
• `cvs checkout modname`
• `emacs modname/file1.pl`
• `cvs commit`

• *(assumes that \$CVSROOT env var is set)*
Commands

• `cvs [options] <command> [cmd-options] [files]`

• **Command**: one of
  – import, export
  – checkout, update, commit
  – add, remove
  – status, diff, log
  – tag, rtag, admin
Setting up CVS in the department

- Make a directory called `cvsroot` e.g.
  - `mkdir ~/cvsroot`

- Initialise Repository
  - `cvs -d ~/cvsroot init`

- Add the following line to your `.bashrc` or equivalent file:
  - `export CVSROOT="~/cvsroot"`
Really Getting Started

• Importing a source = generating a new module
  – cd into the directory containing the source
  – `cvs import <new-module> <vendor-branch> <release-tag> -m "<log message>"`
  – `cvs import goodstuff justme start -m "imported sources"

• Remove the old directory (of course, Backup is always a good idea)

• Get a fresh checkout from the new module:
  `cvs checkout <module-name>`
  `cvs checkout goodstuff`

• Work with it
Basic Operations

- Edit ...
- Adding files: add
- Removing files: remove
- Patching repository changes in: update
- Submitting changes to repository: commit
- View additional information status, diff, log
- Commands apply usually recursively
Adding, Removing files

- add the new file with
  - `cvs add <filename>`
- remove a file: first delete it in the filesystem, then say
  - `cvs remove <filename>`
- The action is reflected in the repository after a `cvs commit`
Updating

• `cvs update` patches the changes, which have taken place in the repository since your last update, to your current working file.

• It only patches in the differences, so your local changes are not lost

• If a conflict occurs, the offending lines are both shown in your files - for manual resolution of the conflict.
Committing

- `cvs commit`
- does not commit, if the repository's revision of the file(s) you changed is newer than the working revision
  - In that case, you've got to do an update first
  - This allows you to perform necessary merging
Revision Numbers

- The history of each file is tracked with an incrementing revision number
- For each revision there is a log entry
- Revision numbers have the format 1.25 if they're on the main trunk, branches have something like 1.33.2.16
- use in commands like `cvs diff -r1.3 -r1.5 <file>`
Keyword Substitution

• Keywords are filled on checkout/update
• These are most useful in the header of documents
• Some useful keywords are
  - $Id$
  - $Revision$
  - $Date$
• e.g. this file; $Id$ expands to:
  $Id: cvstut.slides,v...$
Connecting to the Repository

- CVSROOT is like a URL: Location of the actual Repository
- There are the following typical forms
  - Local System /usr/local/cvsroot
  - Remote Shell :ext:username@server:/cvs/root
  - Client/Server :pserver:username@server:/cvs/root
  - krb5 Server :gserver:username@server:/cvs/root
- Remote shell and pserver are insecure, but…
Connecting to the Repository

• ...Remote Shell access can use any rsh-like program (rsh, ssh, kerberos rsh, etc.) by setting \texttt{CVS\_RSH}

• In UNIX, this is easy:
  - \texttt{export CVS\_RSH=ssh}
  - \texttt{export CVSSROOT=egginton@ice.cs.bris.ac.uk:/home/pgrad/egginton/cvsroot}

• Then use as normal. GUI available for X called \textit{gCvs} (part of CvsGui project)
Connecting to the Repository

• Windows:
  – We will use PuTTY/Paegent to provide us with a secure connection to the server
  – Download TortoiseCVS rather than WinCVS, because it’s just better
  – Setting up PuTTY/Paegent is rather involved, but see http://mongers.org/cvs
TourtoiseCVS Screenshot
Connecting to the Repository

• OSX
  – Dunno
  – But given its UNIX backend and built-in ssh, I assume that it is easy to get working
  – There is an OSX front-end as part of the CvsGui project
Further Information

• Latest CVS manual

• Simple introduction and step-by-step instructions for Windows setup:
  – http://mongers.org/cvs

• GUIs:
  – http://www.tortoisecvs.org/
  – http://sourceforge.net/projects/cvsgui/