Programming in Groups

What's different about groups?

- Multiple developers on a project
  - Can divide the work!
  - Can benefit from everyone's ideas & skills!
- Challenges
  - Coordinate changes & extensions to shared source code base
    - CVS
    - Pair programming
  - Communication, organization, management of people

CVS

- Coordinate changes by multiple people sharing one source base
- Support multiple versions of software
- Allow remote development

Main concepts

- There's one central *repository* of all the stuff being managed by CVS
  - Source files, makefiles, documentation, even binaries
- Each user has a checked-out *working copy* of the repository
  - Can check out all or just part of the repository

Coordinating multiple users

- Users freely edit their own working copy, independent of all other users
  - Don't (need to) care if someone else has modified the same file!
  - Never bothered by someone else's buggy code!
- When happy with changes, a user *commits* their changes to the central repository
- When want to get other users' changes into local working copy, a user *updates* any changes from the repository to the copy

Managing changes

- What if two people have changed the same file?
  - One commits to the repository
  - Then, the other wants to update from the repository
- CVS update will automatically integrate changes
  - If not to same lines, then all's dandy
  - If overlapping lines, then CVS will report a merge conflict
    - User can then edit the file by hand to resolve conflicts

Observing changes

- Don't wait till update to see changes!
- Can use CVS's diff command to compare repository's version to working copy's version
  - See what changes have happened to the repository since you last updated
  - See what changes will happen if you try to update the repository

Versions

- Each commit creates a new version of the updated files
- But all old versions are still there!
- Can easily check out a copy of an older version of any part of the repository
  - To look at different versions of a file over time
  - To revert back to an older, maybe more stable version of the software
- Can use CVS even by a single user, to get version management

Starting a repository

- Pick a directory to be the CVS repository: *cvsDir*
  - Must be editable by all who will be sharing the repository
- cvs –d *cvsDir* init

Creating a CVS project

- Assume you have some existing directory tree you'd like to put under CVS control: *myDir*
  - If not, then create an empty directory
- Pick a name for the software: *myProject*
- cd *myDir*
- cvs –d *cvsDir* import –m "adding *myProject*" \ *myProject myName* start
- Remove *myDir*, after verifying that later commands work

Checking out a working copy

- cd someplace where you want the working copy created
  - Different from the initial imported sources
- cvs –d *cvsDir* checkout *myProject*
  - Creates a directory named *myProject* containing all sources imported under this name
- cd *myProject*
  - Then go ahead and edit away!
- Every user does this (and all later commands)

Adding and removing files

- Must tell CVS if you want to change what files are under CVS control
- cvs add *fileName*…
  - Add file(s) to CVS control
- cvs remove –f *fileName*…
  - Remove file(s), and from CVS control
- Neither affects the repository (yet)

Committing changes

- Once you're happy with your changes, commit them to the repository
- cd *myProject*
- cvs commit
  - Will create an editor window to let you describe the changes, in a permanent log
  - –m "*message*" option to skip editor
  - Does any adds and removes to the repository
    - Remove keeps older versions!
  - Bumps version numbers of changed files

Updating changes

- If someone else changes the repository, eventually you'll want to get those changes integrated into your working copy
- cd *myProject*
- cvs update
  - Reports updated files, and conflicts if any
- Do update before commit

Looking at differences

- What have I changed in my working copy since I last updated?
  - cvs diff
- What has changed in the repository since I last updated?
  - cvs diff –rBASE –rHEAD
- Do these before update or commit!

More in CVS

- Remote repositories, ssh
- Symbolic tags, e.g. RELEASE_1_0
- Version history
- Multiple branches of development
- Handling third-party software
  - "vendor branches"
- Actions upon commit, etc.
  - E.g. sending mail
- Tracking who's editing what files

My wish: nested CVS

- The scenario:
  - I want to check out a working copy of some shared sources
  - I want to then manage my own edits using CVS
    - Multiple internal versions
    - Copies at home & at work
- I want to treat my working copy as if it were a repository, recursively

What more do groups need?

- CVS is a mechanism, not a policy or a management plan
- Groups need to communicate!
  - CVS can help a very little bit
- Groups need to have a management plan!
  - Who's responsible for what?
    - Who's responsible for group management?
  - How to divide up work?
  - What are the policies for testing, committing, debugging?

Pair programming

- One interesting idea: two programmers sitting together at one computer working together (well) is more productive than those two programmers working separately
  - Productivity over the long run, including avoiding design flaws and implementation bugs
- Some advanced development organizations use pair programming
- Try it!