## 6.035 CVS, Tools, JLex Lecture

- JLex
- The 6.035 Locker
- Make
- Java on Athena
- CVS

## JLex: Reg-Exps (1)

- JLex uses grep-style notation, which is different from what was used in lecture.
- See Section 2.3 of the JLex manual (linked to from the 6.035 homepage) for details.
- The following a brief overview, with clarifications for some potential pitfalls.

## JLex: Reg-Exps (2)

- [abc] => a | b | c
- [^abc] => any single character (including newlines) *except* a, b, and c
- [a^bc] => a | ^ | b | c
- [a-d] == [abcd] => a | b | c | d
- [ad-] == [-ad] => a | d | -

## JLex: Reg-Exps (3)

- () groups things, and * and + are as in lecture
- abc+ matches abc, abcc, abccc, etc.
- (abc) + matchesabc, abcabc, abcabcabc, etc.
- [a(bc)] => a | bc

## JLex: Reg-Exps (4)

- Both backslash and doublequotes can be used to signify literals.
- “\”n == “\n” == \\n != \n
- . is any one character *except* newline.
- . == [^\n]

## JLex: Reg-Exps (5)

- a.*b matches afoob, but not afoo\nb
- a[^b]*b matches afoob and afoo\nb but not afobob
- Double quotes do *not* group:
- [a“\n”] == a | \ | n
- [a(“\n”)] == a | “\n”

## The 6.035 Locker

- % add 6.035
- % cd /mit/6.035
- Put “add 6.035” in ~/.environment
- % cat README
- You’ll worry mostly about “groups”, “classes” and “provided”.

## Make

- See /mit/6.035/provided/scanner/Makefile for a well-commented Makefile ready to go for the scanner.
- Beware of relying on javac’s dependencies.

## Java on Athena (1)

- % add -f java_v1.1.6
- 1.1.6 is the only supported version.
- You might want that in ~/.environment
- You might want to use 1.2 for javadocs. That should be fine. Just don’t forget to switch back.

## Java on Athena (2)

- You’ll need to add “/mit/6.035/classes/” to your CLASSPATH in addition to the CLASSPATH for your own code.
- Another wonderful thing for ~/.environment! :)

## CVS! (Concurrent Version System)

- Everything you really need to know is at http://www.loria.fr/~molli/cvs/doc/cvs_toc.html linked to from the 6.035 website.

## Why CVS?

- Multiple users want to edit the same files.
- CVS handles merging changes, even to the same file.
- CVS is no replacement for management!
- CVS keeps track of older versions
- CVS is no replacement for management!

## CVS: Basic Concept

- Repository: Where condensed versions of every version of your files are kept.
- You don't work from the repository. Rather, you work from a "checked out" copy.
- In fact, you should never manually touch anything in the repository.

Fall 2000 13

## CVS: What's in a repository?

- .java source files
- Makefiles
- maybe some text documentation
- NOT class files, tar and jar files, and other binaries or executables.
- Not even .java files generated by JLex and CUP

Fall 2000 14

## CVS: Sample Usage

1) Create a repository
2) Do a checkout
3) Add/Edit a file
4) "Update" your checked out files
5) "Commit", aka "Check In", a file
6) Go away for a while
7) Update again and return to step 3

Fall 2000 15

## How CVS Works (Roughly)

- Uses "diff" to find difference between files. Thus, it only works well on line-based text files, *not* binary files.
- Uses various algorithms to figure out how to merge differences.
- Sometimes fails! You'll have to merge changes manually in those cases.

Fall 2000 16

## CVS: Creating a Repository

- % setenv CVSROOT /mit/6.035/groups/leNN/repository
- % cvs init
- cd to your working directory. We suggest that you create a set of /mit/6.035/groups/leNN/<username> directories to work from.
- % mkdir Compiler; cd Compiler
  (You're now in <workingdir>/Compiler)
- % cvs import -m "Creation" Compiler leNN start
- % cd ..
- Do a checkout

Fall 2000 17

## CVS: Note on repository creation

- The Compiler directory is called a "module".
- The "leNN" and "start" parameters to "import" are just info tags and can be arbitrary, but must be there.
- When you use the import command, any files in that directory will automatically be imported into the module after it is created. If your directory is empty, the module will simply be created.
- If you choose to import your previous files *en masse* in this way, be sure to make clean, i.e., clear your directory of all binary/class files and tool-generated .java files, before running import.

Fall 2000 18

## CVS: Checking Out a Repository

- cd to your working directory
- % cvs co Compiler
- % cd Compiler
- Work from within this directory

Fall 2000 19

## CVS: Updating your files

- First cd to the directory with your files.
- % cvs update
- To get new subdirectories, use
  % cvs update -d
- If you get a merge conflict, you'll have to resolve it manually. Delete the version you don't want.s

Fall 2000 20

## CVS: Adding and Committing

- Create the file or subdirectory.
- % cvs add <filename>
- % cvs ci -m "message" <filename>
- Use commit after adding or editing a file.
- Omitting the filename commits everything (recursively).
- Not using -m prompts you for a message.

Fall 2000 21

## CVS: When to Commit?

- No hard and fast rule.
- General rule of thumb:
  Make sure everything compiles before you check in. There's nothing more annoying than having your code cease to compile after checking out other someone else's changes.
- There are special exceptions.

Fall 2000 22

## CVS: Removing a file

- % rm <file>
- % cvs rm <file>
- % cvs commit <file>
- Note the sort of odd ordering (which prevents you from using tab-completion)

Fall 2000 23

## CVS: Removing a subdir

- Remove everything (and from CVS) inside the subdir. Now, from its parent directory,
  % cvs update -P
- That will get rid of any empty subdirs.

Fall 2000 24

## CVS: Diffing

- Show log:
  % cvs log <file>
- Differences from repository:
  % cvs diff <file>
- Differences from some version:
  % cvs diff -r<version> <file>
- A different display style:
  % cvs diff -u <file>

## CVS in Emacs

- C-x,v, v = Commit
- C-c, C-c after entering a comment.
- See C-x,v,C-h for a list of relevant commands.

## CVS: And More

- Tagging and branching allows you to set checkpoints and such. See the French page for more information.
- We'll also send out mail about a script that'll let you automatically get sent email upon each commit.

## How to use CVS *well*!

- CVS is no replacement for management!
- Minimize working on the same file at the same time.
- If you do, work on different portions.
- Try to use lots of auxiliary files for different functions instead of one mondo file. This will be especially useful for "parser.cup" you'll see later.

## CVS Pitfalls

- Never muck with the repository manually.
- Try not to make drastic changes at once.
- Always update before editing. Managing conflicts earlier is easier.
- Watch out for emacs auto-tabbing. Different versions sometimes tab differently. CVS doesn't like to merge a retabbed file.