6.170 Laboratory in Software Engineering
Fall 2005
Problem Set Procedure for 6.170

Contents:

Overview

This document explains the procedure for how to do problem sets for 6.170 this semester which is considerably different than how they were done in the past. We are introducing some new tools, so this document explains what they are and how to use them for 6.170.

Basically, the steps for completing a problem set are as follows:

1. Start early.
2. Find a computer on which you are going to work and configure your environment. (Also, get comfy because you might be there awhile.)
3. Checkout the problem set from your CVS repository.
4. Think, write, code, compile, debug, commit. Repeat as necessary.
5. Turn in your problem set by doing a final commit to CVS.

The rest of this document explains these steps in greater detail...

Step 1: Start Early

As you know, 6.170 problem sets often take a long time to complete. Even if you cannot start a problem set as soon as it is released, you should read it over right away to try to gauge how much time you will need to complete it. Remember:

> *Everything will take twice as long as you think*
> *even if you know that it will take twice as long as you think.*

Don't deceive yourself into believing that you can put off your problem sets until the last minute – you know what you signed up for!

For additional hints on successfully completing 6.170 problem sets, see the Problem Set hints handout in the assignments section.

Step 2: Find Someplace to Work

No matter where you work, you should configure your environment on Athena first by running the following from the command line *once* (although nothing harmful will happen if you run it multiple times):

```
add 6.170
student-setup.pl
```

Then you have to logout and log back in for the changes to take effect. Now everything should be set up so that you can start working on your problem set. Be aware that the software that you will be using tends to run faster on the Athena Linux workstations than they do on the Athena Solaris boxes, so you should try to use the Linux machines whenever possible.

Working at Home

But maybe working in Athena isn't for you. Sometimes it smells bad, and now that the 24-hour-coffeeshop is gone, w20 may not have the "resources" you need to keep you going all night. Thus, you can try to setup your own machine for 6.170.

Take a look at the Working at Home document for more instructions.

Setting up CVS in Your Environment

This semester, you will manage versions of your code with a program called CVS which stands for *Concurrent Versions System*. This system keeps track of previous versions of your code so that you have a backup copy that you can revert back to, if necessary. More importantly, CVS enables multiple developers to work on a codebase at the same time by synchronizing different versions of files checked in by multiple developers.

Although you will be the only person working on the code for your problem sets, you may want to switch back and forth between working in Athena and at home, so you can still take advantage of CVS's support for synchronizing multiple copies of code. Further, it is good practice to start using CVS now so that you'll be familiar with it by the time you start working on your team final project. More importantly, you will encounter CVS (or some other type of versioning-control software) in industry and elsewhere, so it is an important tool to learn how to use.

So how does it work? On Athena, we have created an individual **CVS repository** for you that has the code you need to get started for each problem set. Each problem set is its own **module** in CVS which you must **checkout** to your local machine. When you create a new file for the problem set, you must **add** it to CVS. Every time you make a change to a file worth that is worth recording, you **commit** that file to CVS. If someone else makes a change to the repository, you must **update** your local copy to get the other person's changes. CVS will try to **merge** the copy in the repository with your local copy, but if it can't, then it will notify you that there is a **conflict** which you must address.

You will need to setup CVS before working on a particular machine. If you are not using Eclipse, you can do this from the command line.

Setting up CVS in Eclipse

Setting up CVS from the command line

Step 3: Checkout the Problem Set from CVS

You will begin each problem set by checking out a module from your CVS repository. This repository has been set up for you by the 6.170 staff.

Checking out a pset in Eclipse

Checking out a pset in Emacs or from the command line

Step 4: Do the Problem Set

Step 4 has a number of subtasks:

- Creating a new file
- Adding a file to the CVS repository
- Editing and compiling code
- Running unit tests
- Committing changes to CVS
- Updating your local copy of the CVS repository
- Resolving CVS conflicts

Creating a New File

Every problem set will require you to create new java source files in addition to the ones we give you. You may also have to create new textfiles that answer questions asked on problem sets. These textfiles should be created in the `doc/` directory.

Creating a New Java File

Regardless of whether you are creating a new Java class or a new Java interface, the file should have the same name (and capitalization) of the class or interface and end with a `.java` extension. Relevant textfiles, screen shots, and diagrams should also be created in your `psN` directory.

Create a new file in Eclipse

Create a new file in Emacs

Adding a File to the CVS Repository

When you create a new file for your problem set, you must **add** it to your CVS repository.

Which files should be added to CVS?

For 6.170, you should expect to add the following files to CVS:

- Java source files
- Text files that answer questions asked by the problem set
- Image files that depict diagrams of your design
- Any file that the problem set dictates should be added to CVS

Which files should NOT be added to CVS?

By comparision, the following files should **NEVER** be added to CVS:

- Generated .class files
- Files generated by javadoc
- Any file that is generated by some script of yours

These files should not be added to CVS because they should be cleanly generated by whoever checks out the module from CVS.

You may have noticed that CVS acknowledges when you have a file that has not been added to the repository. To tell CVS to ignore these files, you should add (and commit) a file named `.cvsignore` that has a list of files in the same directory that should be ignored by CVS. For example, in Problem Set 0, the file `src/ps0/.cvsignore` contains the line `*.class` so that CVS will ignore all `.class` files in that directory. Note that this does not recurse down the directory tree, so `src/ps0/optional/` has its own (identical) `.cvsignore` file.

How Do I Add a File to CVS?

How to add a file to CVS in Eclipse

How to add a file to CVS from Emacs or the command line

Editing and Compiling Code

Traditionally, programmers switch back and forth between editing code and compiling it. However, Eclipse users perform these two tasks concurrently, as Eclipse takes advantage of **partial recompilation** and compiles code every time that you save it.

Editing Java code in Eclipse

Eding Java code in Emacs

When you compile Java code, `.class` files are generated from your `.java` files. Eclipse is set up to place these `.class` files in your `~/6.170/psN/bin/` directory, we recommend mimicing this structure if you do not use Eclipse.

If one or more of your files does not compile, you will receive an error message from the compiler and no `.class` files will be generated for the `.java` file that does not compile.

Compiling code in Eclipse

Compiling code in Emacs and from the command line

Running Unit Tests

JUnit is a Java package that allows you to test your code. It is integrated into Eclipse but it can also be run from the command line.

Using JUnit in Eclipse

Using JUnit from the command line

Committing Changes to CVS

When you make a change to a file that should be saved in the repository, you must **commit** the change to CVS. For whatever reason, `cvs add` does not actually add the file to your CVS repository; instead, it *indicates* that the file should be added to your repository upon your next CVS commit. Thus, you should always commit a file to CVS after adding it because you obviously meant to commit the new file if you decided to add it.

When you commit a file to CVS, you should always add a message when you commit so that you can quickly find an old version of a file if you need to revert to it. (You can see a log of your commits by running `cvs log` *`filename`*.) Also, when your repository is shared among a team of developers, your team mates may want to read your messages to get a summary of what you have committed.

When Should I Commit a File to CVS?

First, you should commit a file to CVS immediately after you add it so that you do not forget to do so later. After that, you should commit a file every time that you make an important change to it, so commit often! As a general rule, you should not commit files to CVS that do not compile because your team mates will be upset if you "break the build." Finally, you should also remember to commit all of your code to CVS before the problem set is due as that is how you will be turning in problem sets for this course.

Committing to CVS in Eclipse

Committing to CVS in Emacs and from the command line

Pitfalls

Some tips on avoiding common problems while using CVS:

- Do not edit the repository manually. It wasn't designed for human use.
- Try not to make many drastic changes at once. This will minimize merge conflicts. This is good coding practice in general.
- Always `cvs update` before editing a file. It's easy to forget this. If you forget this, you may end up editing an outdated version, which can cause nasty merge conflicts.
- Watch out for tabbing. Your text editor's auto-tabbing can be a different width for different people, and other text editors also vary in the way they treat tabbing, both in terms of width and in terms of whether tabs are true tabs, spaces, or both. **All text editors that your groupmates use must use the same tabbing**

convention. This is because CVS compares on a line-by-line basis, so tabbing differences can often cause nasty merge conflicts. This is just a specific case of "You should agree on a code formatting convention."

Updating Your Local Copy of the CVS Repository

Normally, you **update** your local copy of your repository before working on it so that you get others' changes to the code before you start editing it. Although you will be the only person working on the code for your problem set, you should get in the habit of updating your code before you start working and committing when you are done. If you work on your code on multiple machines (such as on an Athena machine and on your own machine), then it is imperative that you commit your changes when you finish working in one place so you can use `cvs update` to get them when you sit down to work in another.

Updating from the CVS repository in Eclipse

Updating from the CVS repository in Emacs and from the command line

Resolving CVS Conflicts

CVS will try to merge any changes made since your last `cvs update` by both yourself and others. If some of your changes conflict with others' changes, `cvs update` will tell you so, and the source file will be changed to include both versions of any conflicting portions (yours and the one from the repository), in this format:

```
<<<<<<< filename
YOUR VERSION
=======
REPOSITORY'S VERSION
>>>>>>> repository version's revision number
```

You must resolve the conflict by editing the file, removing the markers, and leaving whichever version of the code you prefer (or merging them by hand). (Searching for "`<<<`" until you've resolved all the conflicts is generally a good idea.) Once you've resolved any conflicts, you can safely commit the file to the repository.

Note that CVS works on a line-by-line basis. That is, it only knows whether an entire line has been changed, added, or deleted.

Step 5: Turn in Your Problem Set

If all of your work is committed to CVS, then consider your problem set submitted as your problem set is automatically collected from your CVS repository. There is no

specific step for printing or handing in a problem set. You may want to do a fresh checkout on another machine to make sure that all of your files were committed to CVS. Eclipse uses icons to indicate which files have not been added to Eclipse, and command-line CVS precedes the names of files with question marks if they have not been added. Use these cues to avoid forgetting to add files to CVS because you will not get credit for work that is not in CVS.