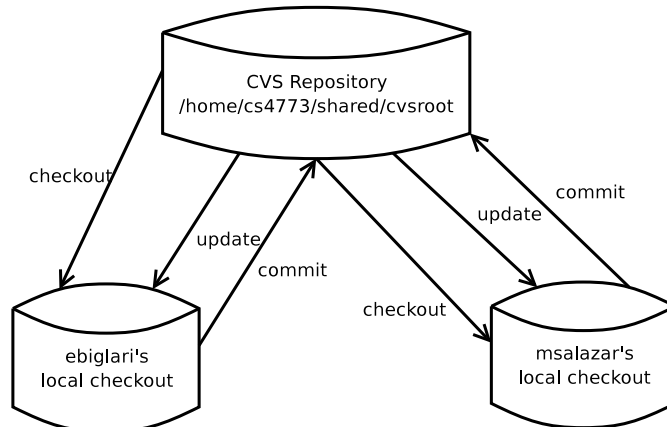


Using Eclipse

CS 4773, Fall 2007, Dr. Jeffery von Ronne

1 Definitions Background on CVS

Think of the CVS repository as a shared folder that everyone has access to but nobody reads from and writes to directly. Instead you *checkout* your own local copy, occasionally *update* your local copy from the shared copy, and occasionally *commit* your local changes to the shared copy. You only have to *checkout* the repository once in order to set things up, and after that you can get the latest changes from the shared version by *updating* your local copy.



The root of the repository is at `/home/cs4773/shared/cvsroot`. Normally you do not want to checkout the entire repository, but merely a subdirectory of it.

Most CVS repository may contain multiple modules associated with different projects. A CVS module is basically just a directory with all of its subdirectories (recursively) and all of the files therein. Normally, with CVS you checkout a complete module to create a working copy, but you can also checkout arbitrary directories and their files (with or without their subdirectories). A CVS module will often include the source files, documentation, a build system (e.g., Makefiles, or Eclipse project metadata), and test harnesses / test cases.

In our repository, you will want to check out the two top-level directories "tst1" and "chat". When you checkout either of these, you will automatically get all of the subdirectories¹ These are setup to be checked as Eclipse packages. "chat" will be the main package for our class project. "tst1" is there for you to experiment with; feel free to add and delete classes, experiment with modifying your neighbor's classes, etc.

In addition to storing shared files, CVS also keeps a history of what files looked like at previous points in time. You can think of this as CVS automatically making a backup of the old shared copy of each file, every time someone commits a new version to it. You can just ignore this feature for now.²

Normally, although you can look at and use any of the old versions stored in CVS, if you want to make changes, you need to update to the newest version, and commit changes to that. It is possible to make modifications to other versions, by creating a new "branch" to which changes can be committed. We're not going to make use of this feature; we'll only be using a single main branch (a.k.a. "trunk", a.k.a. HEAD) of the CVS repository.

Background on Eclipse

Eclipse has multiple "Perspectives", which are different collections of windows that provide user interfaces for different activities. The two perspectives we'll mostly be dealing with are "Java" and "Team Synchronization". You can switch between open perspectives with the ">>" in the upper right corner of. You can open a new one

¹At least, all the ones that contain files; CVS is file-based and sometimes ignores empty directories. The file-based architecture of CVS causes some other quirks as well.

²but this is what the "Versions" and "Dates" stuff in Eclipse's "CVS Repository Exploring" perspective is for.

by clicking on the little window icon to the left. In between, there should be a box showing the name of current perspective.

In class last Friday, when we first connected to the CVS repository, we used the "CVS Repository Explorer". We don't need this,³ and you can delete this perspective by selecting the perspective using ">>" and then using "Close Perspective" from the "Window Menu".

Initial Checkout of an a Project from CVS

1. open the Java perspective
2. menu: File/New/Project, CVS, Projects from CVS
3. If there is an existing repository location for:

```
:extssh: vonronne@pandora.cs.utsa.edu:/home/cs4773/shared/cvsroot
```

use it, otherwise create a new repository location:

Host: pandora.cs.utsa.edu

Repository: /home/cs4773/shared/cvsroot

User: <username>

Connection type: extssh

Port: *default*

4. use an existing module, log in, select the appropriate module (i.e., "tst1" or "chat")
5. check out as a project in the workspace with the default Eclipse project name (which is just the CVS module name)
6. use the default workspace location (i.e., the default)
7. use the "HEAD" tag to checkout from (i.e., the default)
8. Your new project should appear in the "Package" panel on the left-hand side

Creating a Class

1. Open Java Perspective
2. File, New, Class
 - Source folder:** tst1/src or chat/src
 - Package:** *e.g.*, edu.utsa.vonronne
 - Name:** *e.g.*, Adder
3. right click on new class in left panel, select Team, Add to Version Control
4. add methods and fields to the source file in the center pane
5. commit to CVS (see below)

³at least not until we have old versions of our chat client to look at

Synchronizing Eclipse Project with CVS module

1. open Java perspective
2. right click on the project you wish to synchronize
3. select Team, Synchronize with CVS Repository.

This will open up the "Team Synchronization" perspective. On The left "Synchronize" pane allows you to view the list of differences between the local working copy and the shared CVS repository, and has several buttons on the toolbar for navigating through the changes and doing the actual synchronization..

- The left-most item (with a window, a disk cylinder, and an arrow between) in the Synchronize pane tool bar has the tool tip "Synchronize CVS"; think of this as a "refresh button".
- The right most button is the "commit" button that will commit your local changes to the CVS repository.
- Just to the left of the commit button is the "update" button that will update the local copy with the latest changes from the CVS repository.
- The next section of the toolbar to the left contains buttons for filtering the changes.
 - The window next to the left arrow, causes the pane to show only the new changes others have made to the CVS repository that haven't yet been downloaded to the local working directory.
 - The disk next to the right arrow, causes those changes which have been made in the local working directory, but haven't yet been committed to the repository.
 - The icon with two arrows, a window, and a disk causes changes in both directions to be shown.
 - The red double-headed arrow, shows those changes that 'conflict'. This is occurs occasionally, and is usually because somebody else has modified the same line you've modified, and they committed first. So the same line is modified in the CVS repository and in the local working directory. (You have to manually resolve the conflict by editing the text in the windows to the right.)

4. "Synchronize CVS" so that Eclipse knows about the latest changes to the CVS repository
5. check for conflicts using the red double arrow, and resolve them if necessary.
6. check the incoming changes with "Incoming mode", if they look reasonable, go ahead and "update".
7. switch back to the Java perspective
8. make sure everything still compiles,
9. rerun the JUnit test cases.
10. switch back to the Team Synchronization perspective.
11. "Commit" the outgoing changes.
12. write a log message describing what you did and why you did it⁴

⁴PS Don't forget to put a log message in your file in your log directory and commit that; the log message can be the same as the commit log message but should include your start and end times for the logged activity.

Creating a JUnit Test Case class

1. open the Java perspective
2. find and the class you want to test using the "Package Explorer" in the left pane
3. menu: File, New, JUnit Test Case
4. change the source folder to "chat/test" or "tst1/test"
5. check that the "class under test" is, in fact, the class you want to test
6. check that the package is the same as that of the "class under test"
7. check that the default name is the one you want for the JUnit Test Case class
8. "Next", and select the methods in the "class under test" that you want to test
9. edit the new test case class to taste
 - (a) the code needs to initialize an instance of the class under test
 - (b) the code needs to make some method calls on it
 - (c) the code needs to check that the method calls did the right thing
 - (d) if they didn't, the code should cause an exception (use the inherited methods from <http://junit.sourceforge.net/javadoc/junit/framework/Assert.html>).
 - (e) right click on the new class in the package explorer, select Team, 'Add to Version Control'

Running JUnit Test Cases

1. open the Java perspective
2. right click on a package (i.e., "chat" or "tst1") to run all the test cases in the package; right click on an individual JUnit Class to run the test cases in that class
3. select 'Run As', JUnit Test Case
4. if the test fails, try using 'Debug As', JUnitTest