CSE 331 Software Design and Implementation

Handout T3: *Version Control Reference*

Contents:

Introduction

SVN (Subversion) provides the following functionality:

- It allows multiple users to edit the same files independently, working on their own copies and synchronizing their work. In the case of CSE 331 assignments, it permits you to work from both the instructional lab and from other computers (say, at your home), synchronizing your work without having to copy files back and forth.
- It provides backups, permitting you to recreate the state of your files at any moment in the past. This is valuable if your home computer crashes (or is stolen). It is also valuable if you make some edits, then decide they were a bad idea and you want to recover a previous version.
- CSE 331 uses SVN to turn in assignments. At the deadline, we collect the current version of your files from your repository.

SVN works as follows. There is a "repository" containing the master version of the files, including a history of all previous versions. Each user "checks out" a working copy of the

files.

Each user can edit his or her copy of the files arbitrarily, without affecting other users or the master version.

- To store your version of the files in the repository, you "check in" (a.k.a. "commit") your files.
- To incorporate others' changes into your working copy, you "update" your copy.
- You can perform many other operations, such as comparing versions of the files or reverting to a previous version if a change introduces a bug.

SVN is insurance against computer crashes

Over many years of teaching courses like CSE 331, we have observed that on average about one student's computer will either die or be stolen during the quarter. Therefore, you should either work on computers in the instructional lab, or commit your work to the repository *often*. Committing often

- guarantees that as much as possible of your work is saved if your computer dies,
- will convince us that you had been working on the hw when this happens (otherwise, you should expect *no mercy* if your computer dies), and
- is good practice for industry anyway.

Setup: Checking out the cse331 project from SVN

You always edit your own personal copy of files that are under SVN control. Before you can make such edits, you must "check out" your own copies of the repository's master files. You only need to do this step once at the beginning of the term. If you are tempted to use this command at other times, you most likely want SVN's `"update"` command. In the case that you plan to work at both UW CSE and home, you will need to do these setup steps for both your IWS account and your home computer.

The following instructions assume you wish to check out a directory named `cse331` from a repository that is located at `/projects/instr/13sp/cse331/`*`YourCSENetID`*`/REPOS/`.

Throughout this document, you only need to do the work on the command line *or* in Eclipse, not both. If you plan to use Eclipse (as almost every 331 student does), you should follow the Eclipse instructions. However, you should be aware of the command-line versions, which you will find helpful because Eclipse lacks certain functionality and sometimes gets wedged. If you have trouble with Eclipse, then you should use the more reliable and versatile command-line tools. They work on all operating systems. Another option, which is also more reliable than Subclipse, is TortoiseSVN, which only works on Windows computers.

- Command Line
- Eclipse

Command Line

Execute the following commands at the Linux prompt to check out the project to `~/cse331`:

```
cd ~/
svn checkout svn+ssh://attu.cs.washington.edu/projects/instr/13sp/cse331/$USER/REPOS/cse331
cse331
```

NOTE (for those who are new to the command line): When you try to type passwords in the command line, you may be alarmed that you can't see any text entered. Don't Panic. In order to protect your password your typing simply isn't being shown. Just type your password as normal and press enter.

Eclipse

The following instructions will show you how to use Eclipse to check out SVN projects. Before doing so, please check to make sure that you have properly set up the Eclipse environment for CSE 331. We will assume that you will check out a project called **cse331**. When you check out **cse331**, Eclipse will automatically set up an Eclipse **project** named `cse331` for you.

- Eclipse on Linux Instructional Machines
- Eclipse on Windows Instructional Machines

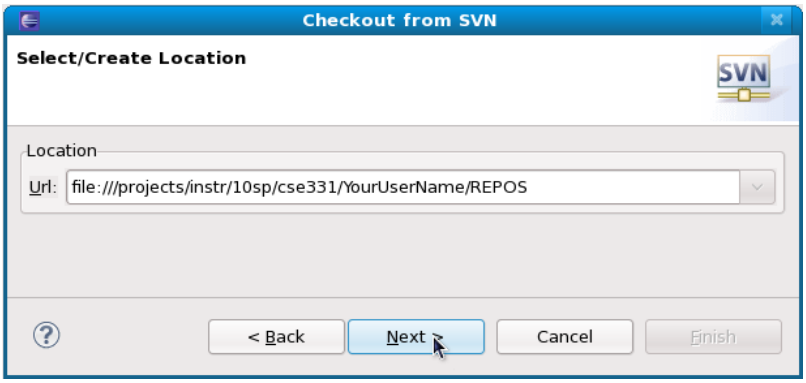
Eclipse on Linux

a. Goto **File » Import**

b. Select **SVN » Checkout Projects from SVN**
If that option does not exist, then you may not have installed SVN.

c. Select "Create a new repository location" a click "Next"

d. Enter the URL
`svn+ssh://attu.cs.washington.edu/projects/instr/13sp/cse331/`*`YourCSENetID`*`/REPOS` as shown below:

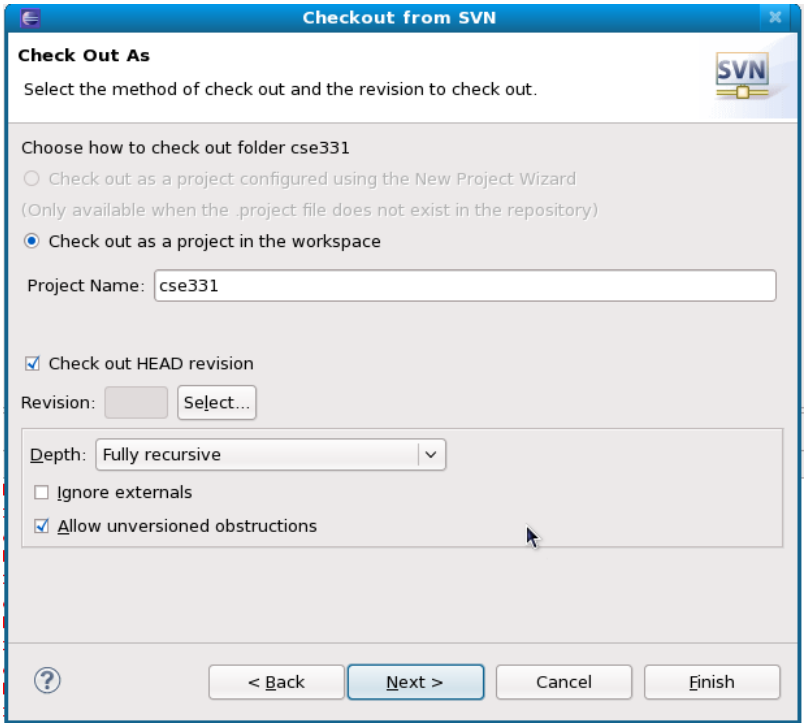


e. Click **Next**.

f. If you are working on your own machine rather than an Allen Center instructional machine, enter your CSENetID and password in the "Enter SSH credentials" dialog (see the Windows instructions below) and click **OK**.

g. In the "Select Folder" dialog, select the `cse331` directory and click **Next**

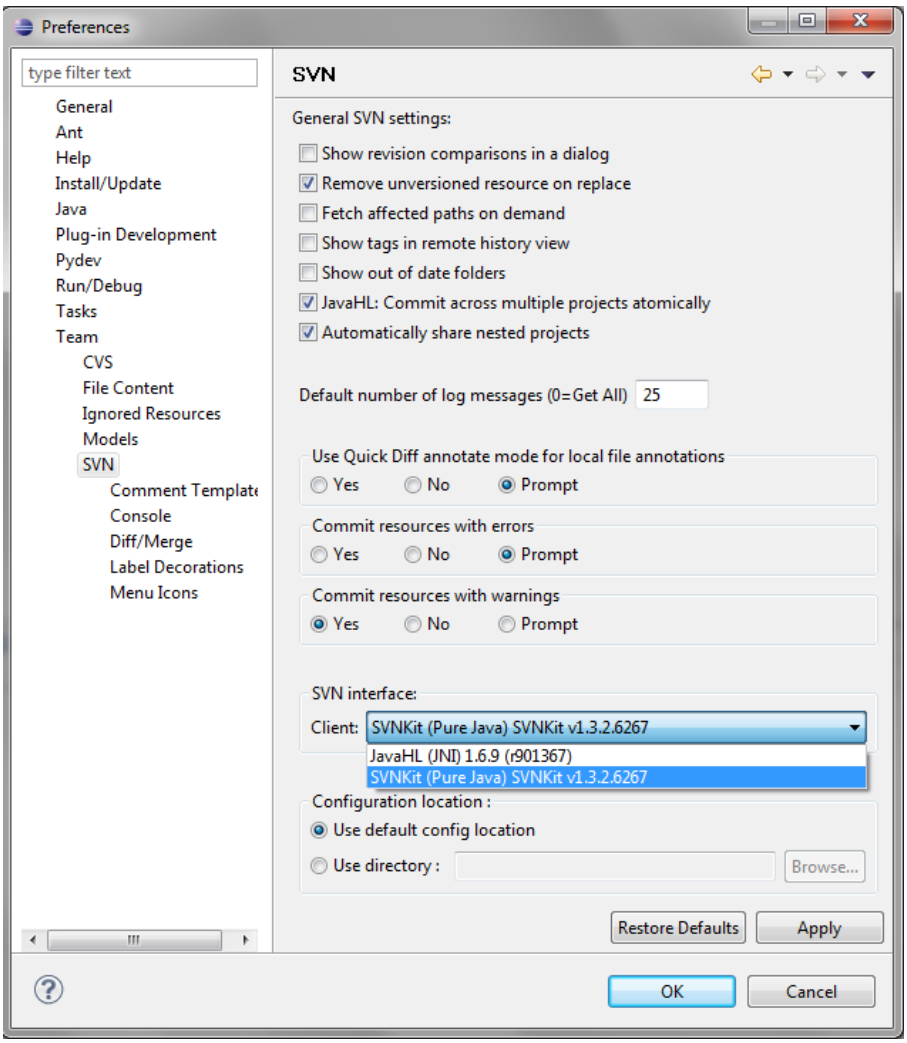
h. In the "Check Out As" dialog that appears (shown below), make sure "Check out as a project in the workspace" is selected and click **Finish**



Eclipse on Windows

a. Goto **Window » Preferences**

b. On the left of the Preferences dialog select **Team » SVN** . In the SVN pane that appears, choose SVNKit (not JavaHL) from the Client pull-down as shown below:



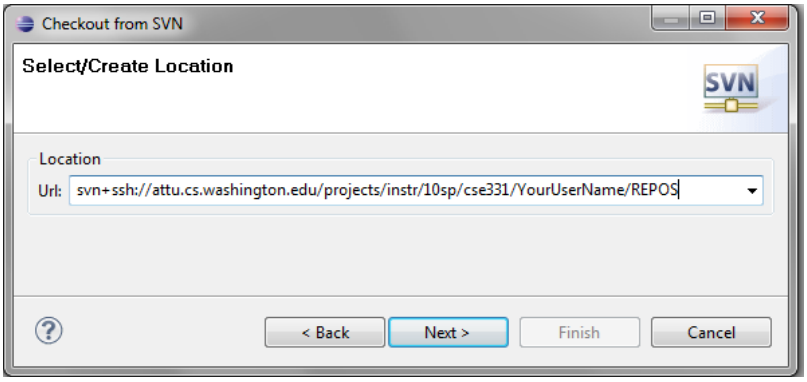
c. Click **OK**

d. Goto **File » Import**

e. In the dialog that appears, select **SVN » Checkout Projects from SVN**
If that option does not exist, then you may not have installed SVN.

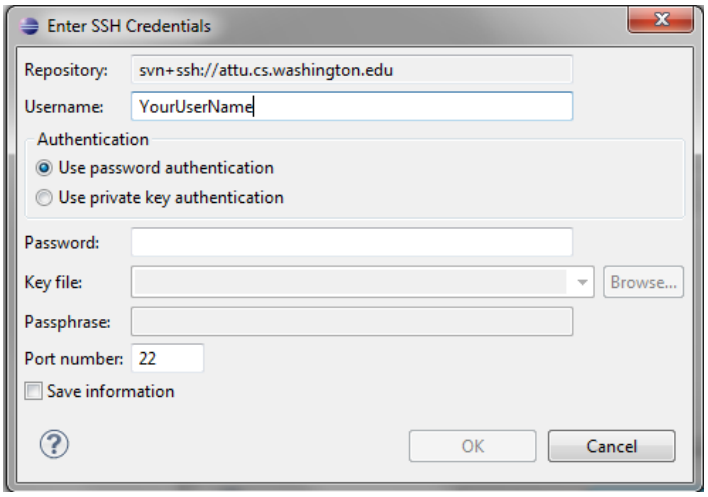
f. Select "Create a new repository location" a click "Next"

g. Enter the URL
`svn+ssh://attu.cs.washington.edu/projects/instr/13sp/cse331/`*`YourCSENetID`*`/REPOS` as shown below:



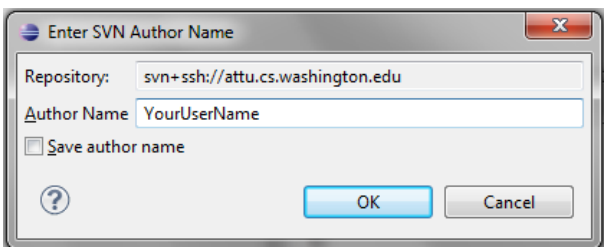
h. Click **Next**

i. Enter your CSENetID and password in the "Enter SSH Credentials" dialog that appears (shown below) and click **OK**. Use the same password you use to login to the Linux machines in the Allen Center software labs, which may differ from the password you use to login to the Windows machines.



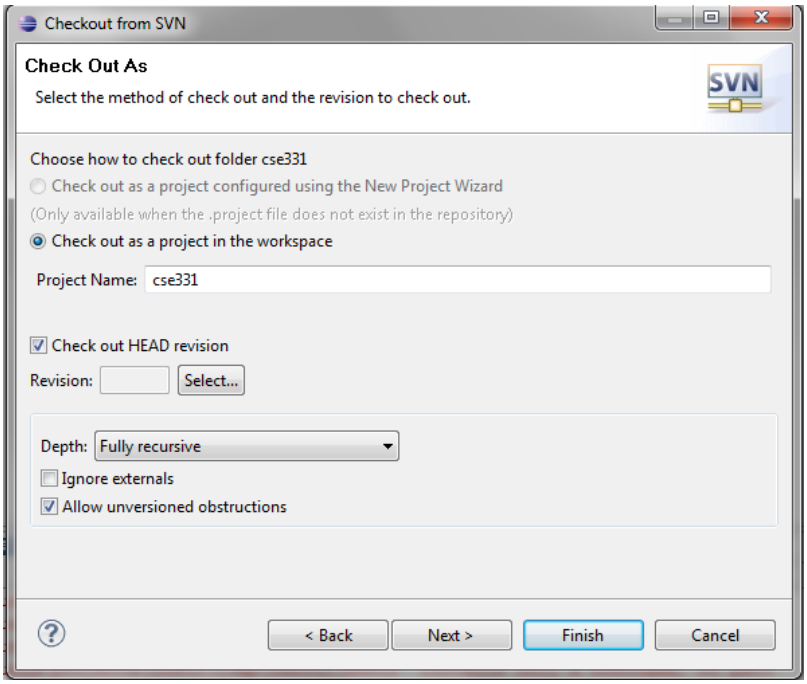
j. In the "Select Folder" dialog, select the `cse331` directory and click **Next**

k. If the "Enter SVN Author Name" dialog appears (shown below), enter your CSENetID and click **OK**



l. In the "Check Out As" dialog, make sure "Check out as a project in the workspace" is

selected and click **Finish**



Adding Files to SVN Control

If you create a new file in a directory which is under SVN control, SVN will **not** automatically add it to the repository. This behavior is to avoid extraneous files from being added to SVN control. (You can tell when a directory is under SVN control when it contains a `.svn` sub-directory with files such as `entries` and `dir-prop-base`, none of which you should directly modify.)

You should not place compiler-generated or other auto-generated files under version control. This includes `.class` files and Javadoc files. You can tell SVN to ignore specific filenames or patterns (e.g., `*.class`) by running `svn propedit svn:ignore .` in the directory that contains the files to be ignored. Unfortunately, you have to re-run this in every directory that contains `.class` files. This simplifies SVN's `update` output in the presence of generated files (see this section).

To make SVN place a file under its control, first follow these instructions, then commit the file.

- Eclipse
- Command Line

Eclipse

Right click on the file to bring up the context menu and select **Team » Add to Version Control`**

Note: if you create a file or folder in an Eclipse project but via the command line (or any mechanism outside Eclipse), then you will need to "refresh" the package explorer for Eclipse to recognize the change. To do this, right click on the project name in the package explorer and select the "refresh" item.

Command Line

If you're are attempting to add a file to the directory `~/cse331/src/hwN/` directory, first you should `cd` to that directory:

```
cd ~/cse331/src/hwN
```

Now you can add a file (or directory) as follows:

```
svn add RandomHello.java
```

Updating Files

SVN's `"update"` command updates your local copy of files to reflect changes made to the repository by other people (or by you when working on a different computer system). The only changes made by people other than you will be made by the CSE 331 staff when we are adding new hws to your repositories. If you work at home and at UW CSE, you will need to use `update` to propagate your changes between the two locations.

SVN usually does a good job of merging changes made to multiple checkouts (say, by different people or by you on your home computer and you at UW CSE), even if those changes are to different parts of the same file. However, if both people change the same line of a file, then SVN cannot decide which version should take precedence. In this case, SVN will signal a conflict during SVN update, and you must resolve the conflict manually. This can be an unpleasant task.

To minimize the possibility of conflicting changes being made simultaneously, you should **update frequently** and **commit frequently**.

Eclipse

In the Package Explorer window, right-click on a file or directory, and select **Team » Update**. If the selected item is not a directory, just that file will be updated; otherwise, everything inside the directory will be updated.

Command Line

To update your local copy, go to the root of your repository (e.g. `~/cse331`) and run:

```
svn update
```

This will display a list of files that have been updated. You should pay attention to the character that shows up in front of each file, as it indicates what sort of change was made to the file:

A	Indicates that this file has been **added** successfully.
D	Indicates that this file has been **deleted** successfully.
U	Indicates that this file has been **updated** successfully to match the latest copy in the repository.
G	Indicates that your working copy of the file is different from the repository's latest

	version, but that **merging** the repository's copy with yours is successful.
c	Indicates that there was a **conflict** when trying to merge your local copy of the file with the one in the repository. (See Resolving Conflicts for more information.)

Committing Changes

After making changes to, adding, or removing files, you must "commit" your changes to SVN. This step will cause SVN to record your changes to the repository, so that your changes are backed-up and available to other people working on the repository, or to you when working on a different computer system.

In general, you should "update" your files before committing them. (And, if the update results in any conflicts, you should resolve them before committing.) If you forget to update, SVN will abort and remind you to update first.

It is a good idea to commit your changes frequently. It backs up your work, thus enabling you to revert to an earlier version of your code if you find yourself going down a wrong path. Also, when you are working with others, it minimizes conflicts.

- Command Line
- Eclipse

Command Line

First, enter the directory in which the file(s) you wish to commit are located. For example:

```
cd ~/cse331/src/hwN
```

Then, you can commit a group of *filename(s)* to SVN by running:

```
svn commit -m "a descriptive log message" filename(s)
```

If you omit the message flag `-m "a descriptive log message"` from your `commit` command, SVN will throw you into an editor where you can enter a message.

If you omit the *filename(s)* from your `commit` command, then SVN will recursively commit everything in the current directory. Be careful not to inadvertently commit files that you don't want to commit.

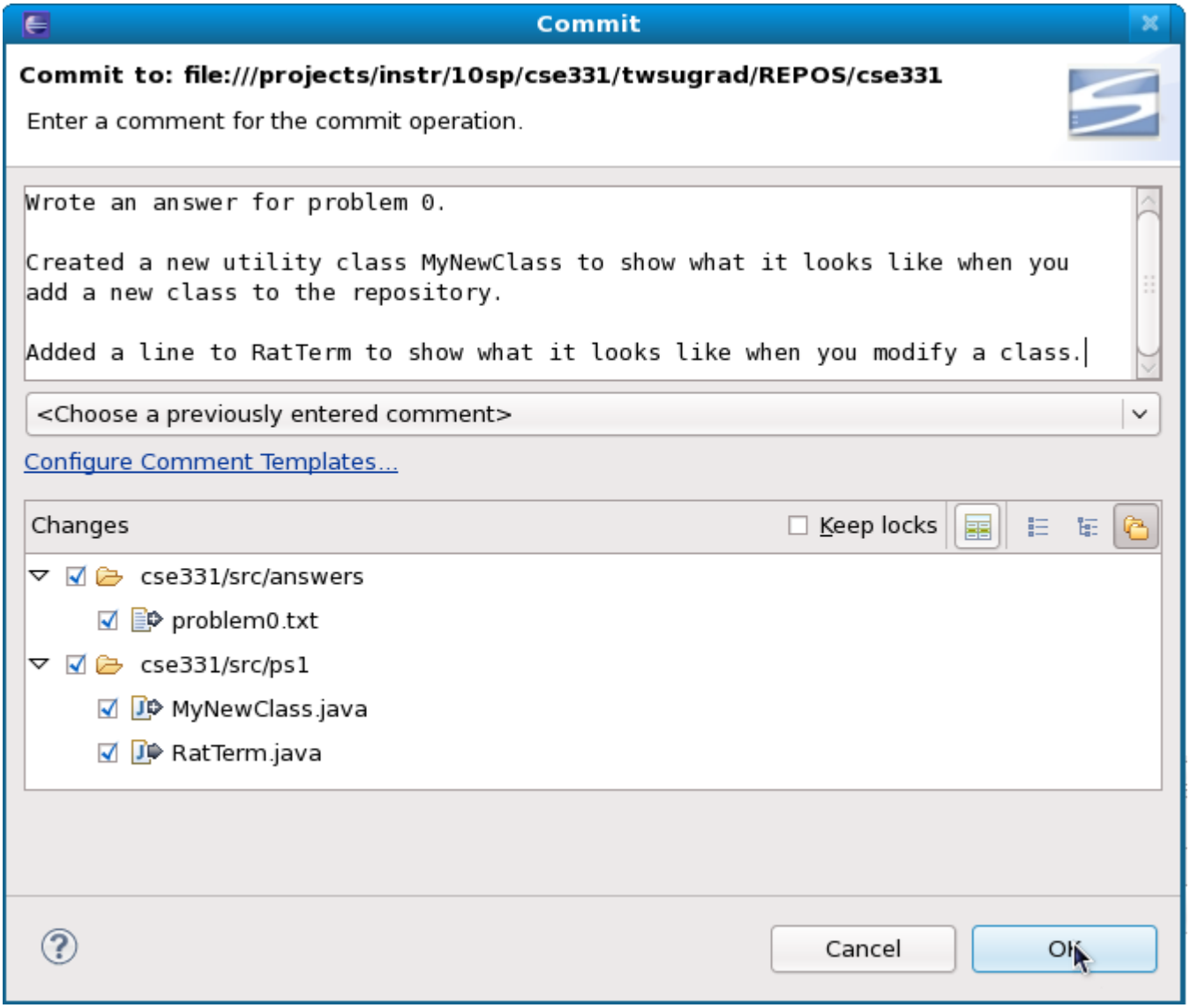
Eclipse

a. Eclipse allows you to add files and commit changes in one step. Open the **Java perspective** (**Window » Open Perspective » Other... » Java**) if you're not already in it. Open the **Package Explorer** view if you don't already see it (**Window » Open View » Package Explorer**). Make sure before you commit that you've refreshed the current Package Explorer view by pressing F5, this will take into account all changes that did not occur in eclipse. In the Package Explorer window, right-click on `cse331` in the directory structure (the one representing the entire project). Choose **Team » Commit...**.

b. Depending on your settings, you may be prompted to add or ignore new files on your local file system that are not in the repository. If you choose to add these files, they will show up in the summary in the **Commit files** dialog with a small + sign next to

their name. With every commit to SVN, you are recommended to append a comment describing changes you made since the last commit. If you ever need to look at old versions of your code, your commit messages will remind you of what happened between versions.

Enter a descriptive commit comment, and click **Finish**. The lower panel shows you the list of files that you have modified and that will be committed to the SVN repository.



c. If a dialog appears asking for an author name, enter your CSENetID

Resolving Conflicts

When multiple people (or the same person on multiple machines, such as the lab machines and your own computer) are working on the same file concurrently, SVN tries to merge the changes made by each person together as each person calls SVN update. Usually, SVN succeeds. However, sometimes SVN is unable to merge the files together when there are two different changes to the same line of a file. In this case, SVN will signal a conflict during the update; `svn update` will produce output such as `Conflict discovered`, and `svn commit` will produce output such as `Commit failed (details follow): ... Aborting commit: 'myfile' remains in conflict.`

SVN conflicts are rare - most students will never encounter one - but if you do get a SVN conflict, you need to resolve it. This is a very brief primer about resolving conflicts; you

can read the SVN documentation to get the full story. Also, it's better to prevent a merge conflict than to have to resolve it later on.

You can resolve conflicts within Eclipse; see its documentation for instructions. However, we've had some reports of problems with Eclipse's SVN support. If you have trouble with it, then you can work from the command line. Again, see the SVN documentation for full details.

To see the status of all your files, run `svn status`. This will tell you, for each file and directory, whether it is currently in a conflicted state or not.

Subversion distinguishes two types of conflicts. A *file conflict* results from simultaneous overlapping edits to a file, and is the more common type. A *tree conflict* occurs when a file is moved or deleted in one checkout, and is edited or differently moved in the other checkout.

When SVN detects a **file conflict**, it changes the file to include *both* versions of any conflicting portions (yours and the one from the repository), in this format:

```
<<<<<<< filename
YOUR VERSION
=======
REPOSITORY'S VERSION
>>>>>>> .r128 -- repository version's revision number
```

For each conflicting file, edit it to choose one of the versions (or to merge them by hand). Be sure to remove the `<<<<<<<`, `=======`, and `>>>>>>>` lines. (Searching for "`<<<`" until you've resolved all the conflicts is generally a good idea.)

Once you have made these edits, then you can tell SVN that you have resolved the conflicts in that file by typing a command like:

```
svn resolve --accept=working src/hw2/test/SpecificationTests.java
```

Resolving a **tree conflict** is similar, but instead of or in addition to editing the file, you must manually decide which of the versions you prefer and move it to the checkout you are working with. Then, you can run a `svn resolve --accept=working` *`myfile`* command, just as for a file conflict.

Preventing merge conflicts

The text above showed how to fix a merge conflict if one occurs. It's better to prevent them in the first place. Conflicts are possible even when you are working by yourself.

- Always commit when you finish work on one computer, and update before you start work on another computer.
- View old versions in your repository as read-only: you can view them, and even copy parts of them into the current version of your code, but you should not edit the old versions. Never revert your repository to a previous version and then begin editing it.

The remainder of this section gives tips for preventing merge conflicts when working with teammates.

SVN is no replacement for management! Coordination of work is important, even if you're working separately. You should minimize working on the same file at the same time if possible. If you do work on the same file, work on different portions. Modularizing code into multiple files often makes parallelizing work more efficient. You should always pass

major design decisions by your teammates before implementing them, particularly if they involve interfaces that will affect their code.

When and how often should you commit? If you commit too often without sufficient testing, you may introduce bugs into the repository that will affect your teammates' work. However, if you commit too rarely, your teammates will be using outdated code, which may cause wasted effort and merge conflicts later.

There is no hard and fast rule, but one good rule of thumb is to make sure everything at least ***compiles*** before you check in. If you check in non-compiling code, your teammates will be very annoyed when they update (which is good practice) and they cannot compile the code any longer.

Another good rule of thumb (though this one is far more malleable) is that you should minimize leaving something uncommitted when you quit for the day. A lot can happen while you're not coding, and it's generally better to get your changes in working order and commit it before you leave. Large amounts of uncommitted code being committed all at once will result in much more conflicts than small amounts of code being committed often. Since the previous rule (of never checking in non-working code) is more important, this can be hard to accomplish if you're making big changes. Thus, it's often good to tackle one feature at a time, so you can finish each piece quickly and keep the repository up-to-date.

Coordinating your efforts with your teammates is, of course, the true key to minimizing merging hassles. Again, SVN is no replacement for management!

svn:ignore

Command Line

There will often be files that you do not wish to check into SVN. These will include `.class` files and Javadoc files. However, SVN will, annoyingly, point out the existence of these files whenever you update. A solution is to tell SVN to ignore these files by setting the `svn:ignore` property:

```
svn prohw svn:ignore ON file(s)
```

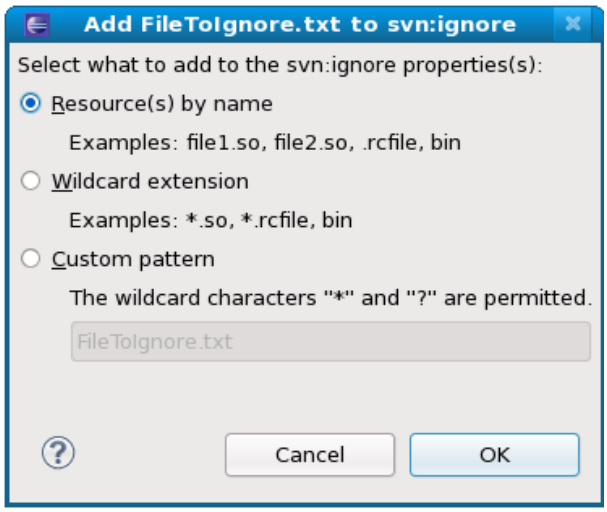
Alternatively, you can modify the ignore policy for an entire directory with the command:

```
svn propedit svn:ignore dirname
```

This will open an editor where you can enter the names of the files and directories to be ignored. Each file name should go on a separate line. You can also use `*` to match any file. For example, to ignore all the class files in the directory, add `*.class`.

Eclipse

To ignore a file that is not under version control, right click on the file to bring up the context menu and select **Team » Add to svn:ignore**. This will bring up the dialog:

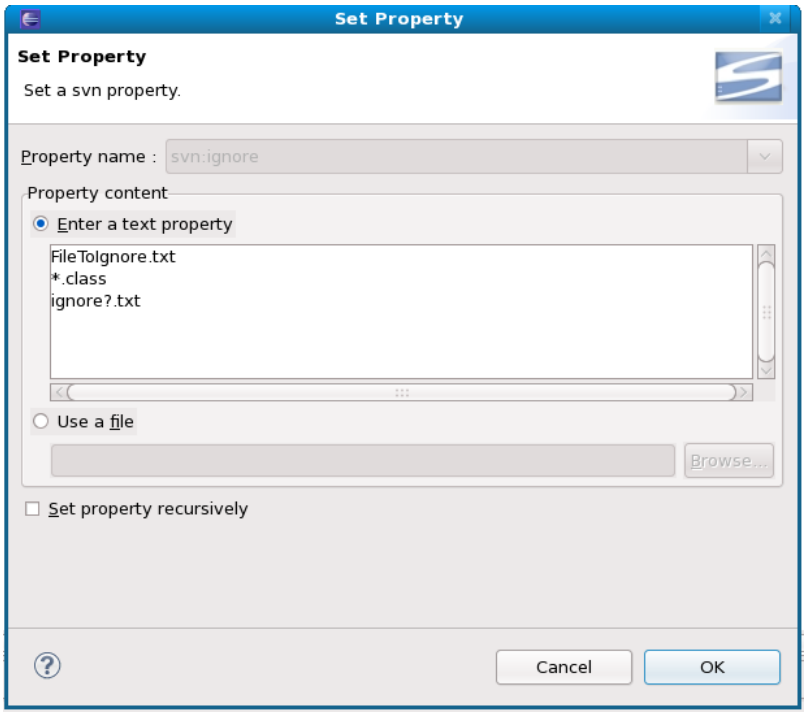


You can ignore just the selected file by choosing "Resource(s) by name" and clicking OK. Alternatively, you can specify a pattern used to ignore files in the directory containing the file you had originally selected.

You can edit the set of files being ignored in a directory by following these steps:

a. Right click on the directory in the Package Browser and select **Team » Show Properties** from the context menu. This will open an "SVN Properties Pane" showing a list of properties. If svn:ignore is not included, use the previous method to first ignore a file (or set of files).

b. Right click on the svn:ignore property and select **Modify Property**. This will bring up the following dialog:



c. Select "Enter a text property" and enter a list of file patterns to ignore.

d. Click **OK**

Adding and Removing Directories

Command Line

You can add a sub-directory with:

```
svn mkdir dirname
```

To delete a directory or file from the repository, use the `svn delete` command:

```
svn delete dirname
```

After adding or deleting a directory, you must perform a commit for the change to be reflected in the repository.

Tracking Changes

Eclipse

Eclipse conveniently marks files and directories that have changed since the last SVN update by adding an asterisk to the file's icon the **Package Explorer**. In addition, there are two features that allow you to track changes between your working copy and the repository's latest copy: **Compare** and **Synchronize**.

To compare a file with its latest version, right-click it in the **Package Explorer** and select **Compare With » Latest from repository**. If the files are different, a window will appear showing a side-by-side comparison of the two files.

To see a summary of differences between the local copy and the repository, right-click a file or directory in the **Package Explorer** and select **Team » Synchronize with Repository**. A window will appear that summarizes which files (if you selected a directory) have outgoing changes (changes you've made after updating), which have incoming changes (new revisions committed by others to SVN), and which have conflicts. Double-clicking one of these summarized items will bring up a Compare window for that file.

To view SVN commit logs and previous revisions to a file, right click it in the **Package Explorer** and select **Team » Show History**. You will see a **History** window in the bottom panel. Double-clicking a row will allow you to read the corresponding revision.

Command Line

To see the change log, which is a list of the messages used when checking in changes:

```
svn log filename
```

To see differences between the working copy and the repository's latest copy:

```
svn diff filename
```

Omit *filename* to see differences for all files. Use the `--revision N` flag to compare with revision `N`, and use `--revision N:M` to compare versions `N` and `M` with each other.

Viewing an old version of a file

You can retrieve an old version of a file under Subversion control with commands like the following:

```
svn update -r 140 MyFile.java
svn update -r {2013-03-14} MyFile.java
```

The first command used a revision number and the second used a date.

This command changes your working copy - that is, your local directory - but it *does not* change the repository. **Do not** attempt to edit the old version of the file in your working copy. Doing so will result in nasty merge conflicts and confusion.

You should save a copy of the file somewhere, then `svn update` your working copy to the current version of the file. Now, edit the current version in whatever way you like, possibly copying some or all of the differences from the old version that you saved. You can discard the old version when you are done; there is no need to check it in into SVN.

SVN Pitfalls

Some tips on avoiding common problems while using SVN:

- Do not edit the repository manually. It wasn't designed for modifications by humans.
- Try not to make many drastic changes at once. Instead, make multiple commits, each of which has a single logical purpose. This will minimize merge conflicts. This is good coding practice in general.
- Always `svn update` before editing a file. It's easy to forget this. If you forget, you may end up editing an outdated version, which can cause nasty merge conflicts.