EECSpeaks, Volume 39, Issue 2, October 2007
http://hkn.eecs.umich.edu/pdfs/EECSpeaks/vol39_issue2.pdf

Using Subversion on your ITD AFS space

By Adam Goodman

For any sort of group coding project (for a class, for work, or just for fun), it is absolutely essential to use some sort of version control system. You've probably heard this before from you instructors, but believe it! Using one of these systems can prevent you from messing up in any number of ways: for example, you won't overwrite your groupmates' changes unless you really try, and if you make a huge mistake in coding, you can revert to a previous version...

If you've taken a group-coding course here at Michigan, it's likely that your instructors provided you with some quick instructions on setting up either CVS or Subversion. If they told you to use Subversion, then you might already know much (but maybe not all) of what is presented in this article. If they told you to use CVS, *don't*. I mean it. Of all the version control systems with widespread use, CVS is probably the worst. Linus Torvalds would rather send diff's back and forth through email than use CVS, and would probably laugh you out of whatever room you were in if you admitted to him that you used it. (Though, he's not that much of a fan of Subversion either.) I won't bother getting into the reasons why Subversion is better than CVS, but trust me when I say that they are numerous.

Instead of further expounding upon the virtues of version control, I'll just proceed by telling you how to get started using Subversion in the University of Michigan's computing environment. If you follow these instructions properly, then you'll be able to access the repository from *anywhere* – not just computers within the U-M network!

First, a quick bit of terminology: a *repository* is the central storage location for your code. A *working copy* is, well, a copy you can work out of, and periodically sync up with the central repository (in a few different ways; we'll get to that later). Note that you cannot work directly out of the repository: it is simply impossible, due to the storage format used by the repository, and it would be a bad idea anyway.

Creating a Subversion Repository

Since ITD now provides more storage space than CAEN, you'll probably want to create the repository in your ITD space. Because of some complicated version-incompatibility issues, you need to login to the ITD servers to create the repository initially:

First, use your favorite SSH client to connect to "sftp.itd.umich.edu" and in your home directory, create a directory to hold your repository:

$ mkdir *svn_repo*

Next, give your groupmates permission to access the directory (where *uniqname1*, *uniqname2*, etc. are the uniqnames of your groupmates):

$ fs setacl *svn_repo uniqname1* rlidwk

$ fs setacl *svn_repo uniqname2* rlidwk

Then, create the repository:

$ svnadmin create *svn_repo*

Getting a Working Copy

You can create as many working copies of a repository as you want, but you should probably make at least one per group member. That is, multiple people should not work out of the same working copy (unless you're doing pair-programming) as this defeats one of the major reasons to use version control.

To get a working copy on a CAEN linux machine, first make sure you've run "gettokens" to get access to your ITD space. Then, run:

$ svn checkout file:///afs/umich.edu/user/*letter1*/*letter2*/*uniqname*/*svn_repo*

Where *letter1* and *letter2* are the first and second letters of your uniqname, respectively. So, if your uniqname was "superman" this command would be:

$ svn checkout file:///afs/umich.edu/user/s/u/superman/*svn_repo*

To get a working copy on a non-U-M linux machine (first make sure you have subversion and ssh installed), you can run:

svn+ssh://uniqname@sftp.itd.umich.edu/afs/umich.edu/u/letter1/letter2/uniqname/svn_repo

You'll have to enter your ITD password a few times; that's just the way of it. Note that any of your group members should be able to run any of these commands under their own user accounts (but the repository path still needs to include *your* uniqname, since it's stored in *your* home directory).

Adding and Modifying Files

When you check out a working copy of a newly-created repository, it will be empty. Say you create a file called "hello.txt" in the working copy, and you want to add it to the repository. You can do so by running the following commands:

$ svn add hello.txt

$ svn commit –m "*commit message*"

Whenever you add a file, you need to explicitly tell Subversion that you have done so, as above. If you modify a file, you need only commit it.

The most important thing to keep in mind when using subversion is that no changes are applied to the repository unless you use the "svn commit" command. So if you add, delete, or modify files in your working copy (even with SVN commands), don't forget to commit. When you commit, you need to enter a message describing the changes you made; these messages appear in a log you can query later.

When and how often you commit changes is up to you. With real, production code, people often institute rules saying that you should only commit code that will compile and run successfully, but if you're in the middle of building a project from the ground up, such a restriction might not make much sense. Also, remember that more-frequent commits will give you more points to which you can revert later, should it become necessary (you could think of it a little like Windows' "System Restore" feature).

If you want to undo local, uncommitted changes, you can run:

$ svn revert *file*

Updating

To copy new changes from the repository to your working copy, you can run:

$ svn update

If you have made local changes to a file that was updated in the repository, then subversion will attempt to merge the changes. If this cannot be done easily, however, then it will place the

file in a "conflicted" state and you will have to edit it manually. Subversion will place markers around "conflicted areas" and show the state of your working copy, and the current state of the repository. It will be up to you to ensure the correct code ends up in that area; when you have done so, run:

$ svn resolved *file*

To tell subversion that you have fixed the problem (it will not let you commit otherwise).

Viewing Status

You often might want to view the current status of your working copy (what files have changed, etc. with regard to the working copy). To do this, run:

$ svn status

If you want to see what changes have been made to a particular file in your working copy, you can run:

$ svn diff *file*

If you do not provide a file to the command, it will show you all of the changes against the working copy.

Viewing History

To view the log of commit messages, run:

$ svn log *file*

Again, the *file* parameter is optional – if provided, the command will only show commits relevant to a certain file; otherwise, it will show everything pertaining to your current directory.

You can also view differences across revisions. To see what changed between two revisions:

$ svn diff –r *rev1:rev2 file*

You can specify "HEAD" as *rev2* to refer to the most recent revision, and *file* is optional yet again.

Reverting to a Previous Revision

If you really mess something up, you can revert your working copy to the state of a previous revision:

$ svn merge –r HEAD:*rev1* .

This will compute a backwards diff between the current revision and the specified one, and then apply it as a patch to the current working copy. You'll need to commit for this to take effect in the repository, as always.

If you merely want to see the contents of your repository at a previous version, you can checkout a new working copy at that revision (you probably shouldn't try to make any changes to it, however):

$ svn checkout –r *revision svn_repo_path*

File Management

If you want to copy, move, or delete a file, use SVN commands:

$ svn copy *source destination*

$ svn move *source destination*

$ svn delete *file*

These will preserve the revision history of the files involved.

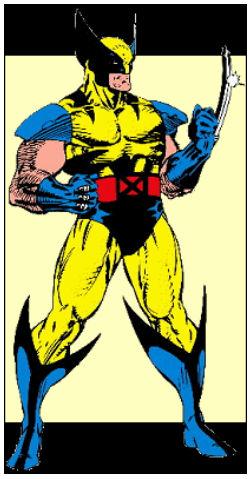
Further Resources

There's way more to Subversion than what I've mentioned here, though hopefully I've provided enough information to get started. If you want to learn more, you can check out the official documentation at:

http://svnbook.red-bean.com/

Also, if you use windows, you might want to check out TortoiseSVN, a really well-done graphical interface to subversion that integrates with Windows Explorer. (I won't try to explain how to use it here, but all the commands should line up reasonably well.) You can find it at:

http://tortoisesvn.tigris.org/



"Pretty much Wolverine is awesome and I wish I could be him"

Sushil Answers a Really Hard Question

By Sushil Gupta

This issue's hard question for which Sushil has an answer is:

Can Wolverine drown?

This question was heavily discussed in the old, old office by me and Zach for a while one time. It was awesome. I watched all the Xmen movies and all 5 seasons of the cartoon again just to get a good answer for this one.

Dictionary.com lists drowning as to die under water or other liquid of suffocation. One can assume that dying by drowning is caused by a lack of oxygen to the brain. But do Wolverine's cells even need oxygen? In X2, Professor X gets mad at Wolverine for smoking in Cerebro, and Wolverine extinguishes his cigar on his own hand, while *appearing* to hold his breath. I can imagine this would be painful, but the hand heals, so it can be assumed that Wolverine does not require oxygen, but does it so that there is less air for the rest of us, which fits perfectly with his history of badassery and nonsense. Even if he does need oxygen, what does the lack of oxygen do to cells? Whatever it is, I bet Wolverine could just heal it out. I mean in Season 4 of the cartoon, Wolverine heals out radiation. Radiation is way worse than a little water in the lung. More important to all of this is, can Wolverine swim? I can imagine he weighs a lot with all that metal on his bones. Pretty much Wolverine is awesome and I wish I could be him. Minus the haircut. I'm not sure how that got popular, but him and Beast need to get that garbage taken care of.