Getting Off the Ground with SVN

This document is an add-on to the document titled “Setting Up Your UNIX Environment for Large Projects”. If you have not read that article yet, you should read it first. This article’s aim is to provide extra knowledge on how to use SVN effectively for Computer Science projects.

Table of Contents

Introduction to Version Control

Version control is any practice or software that allows programmers to keep track of changes to source code, and have control over what changes are used.

Version control is usually set up in such a way that there is a shared **repository** of files for many people to access. Each of these people has a local copy of the repository and is free to work on that local copy. When they reach a milestone, they are able to push their changes into the repository. This doesn't change others' local copies of the repository. When the others are ready, they are able to merge their current copy of the repository with the actual contents of the repository which are now edited.

Repositories also keep track of all of the changes to date, associating each change to their files with a number. This allows users to take a look at previous versions of the repository. This number is called a **revision number** of the repository.

SVN Commands

Subversion, or SVN, is a fairly simple form of version control. It has much fewer commands than some other forms of version control like git. The main commands for SVN are: add, cat, checkout, commit, diff, ls, revert, rm, and update.

svn checkout is what is used for creating a local copy of a shared repository. This is usually only done when first trying to get a copy of an online repository onto your Unix machine. It is called by passing in a url for a repository:

```
svn checkout svn+ssh://cs164-cb@torus.cs.berkeley.edu/staff
```

There is also a shortcut for the word "checkout", so this command can also be called like this:

```
svn co svn+ssh://cs164-cb@torus.cs.berkeley.edu/staff
```

"svn ls" is what is used for viewing all of the contents of a folder in the actual repository. If no argument is passed in, SVN displays the files in the actual repository's version of the current folder. A call to "svn ls" could look like this:

```
svn ls machine/newFolder
```

"svn cat" is what is used for displaying the contents of a file in the actual repository. It is called by passing in a filename/path for you to display. This would actually do the same thing as the Unix built in command "cat", but it displays the repository's version of the file (without your local edits). It would look like this:

```
svn cat fileName
```

You can also pass in a revision number to look at the contents of a file at a particular revision number:

```
svn cat -r 4 fileName
```

The above example would look at the contents of filename at revision number 4.

"svn diff" shows the differences between the local copy of a file and the copy of the file in the actual repository. It is called by passing in the file's name or a path to the file into the command:

```
svn diff fileName
```

"svn diff" can also take in a revision argument to compare the local copy of the file to a copy of the file in an older version of the actual repository:

```
svn diff -r 4 fileName
```

“svn commit” is what is used to push the local changes into the online repository. It takes every file that is affected by local changes, and overwrites the shared repository’s copies of those files by the local copies.

It is recommended that it is always called with the “-m” flag, so we will only explain its usage with this flag. If you accidentally call it without its “-m” flag, you will be redirected to a text editor (often Nano or Pico) to write a message explaining what the changes are. This can be slow and annoying.

svn commit’s “-m” flag requires the user to pass in a message after it (if you want to include spaces or special characters, you must enclose your message in quotation marks). This means that users’ commit commands tend to look like this:

```
svn commit -m “fixing the off-by-one error in priorityScheduler.java”
```

“svn commit” can also take an unbounded number of filenames after the call so that only certain files are committed (instead of every file affected by local changes). Such calls would look like this:

```
svn commit -m “fixing the testing harnesses” lib/setup machine/testingHarness.java
machine/testDriver.java
```

“svn add” is used to add a file to the shared repository. This is necessary because SVN only keeps track of changes to files that the repository already knows about. It doesn’t handle new files. “svn add” is used to alert the repository that the next commit should push the local copy of the file into the repository, and that from then on, SVN should watch for changes to that file. It is called like this:

```
svn add newFile
```

“svn remove” is used to remove a file from the shared repository. It will also remove the local copy of that file. It is passed a folder or file’s name, and looks like this:

```
svn remove uselessFile
```

It can also be called recursively on a folder by giving it a “-r” flag:

```
svn remove -r uselessFolder
```

“remove” can also be called with the words “delete” or “rm”, so all of the following calls do the same thing:

```
svn remove -r uselessFolder
svn delete -r uselessFolder
svn rm -r uselessFolder
```

“svn update” is used to push all changes from the shared repository onto your local copy. This means that all files that were changed since you last updated your repository will be overwritten. If you have

local changes on affected files, SVN will ask how you want to handle this kind of file conflict. In this case, you can choose to do one of the following:

1. Theirs-Full (tf) – Completely overwrite the local copy with the copy in the shared repository, disregarding local changes
2. Theirs-Conflict (tc) – Merges the local copy with the repository's copy in a smart way, using the contents of the repository's copy when there are different parts in the files. This usually works, but you should always check the outcome of using this option – sometimes SVN will not do it correctly.
3. Mine-Conflict (mc) – It is the same as "Theirs-Conflict", but prefers the local changes to contents of the repository's version of the file
4. Mine-Full (mf) – Ignores the copy in the repository, keeping the local version of the file.
5. Edit (e) – Opens an editor like Emacs (or if SVN is not configured to use your favorite editor, Nano or Pico), and allows the user to manually fix changes.
6. Resolve (r) – Tells SVN that the edited version of the file is the version the user wants to keep. This should only be called after calling "Edit" - otherwise, files will no longer compile.

Calling "svn update" is simple since it takes no arguments:

```
svn update
```

It can also be called as:

```
svn up
```

"svn revert" is used to undo all changes to a certain file. This can be important if you decide that some changes that have been implemented on a file cannot do what you had hoped they would do, and take a long time to undo. Calling "svn revert" will undo all changes to the local copy of that file. This does not mean that the local copy will now be the same as the version of the file in the actual repository if somebody committed that file into the repository after the last time you updated your local copy of the repository.

"svn revert" is called by passing in folders or files:

```
svn revert fileName1 fileName2 fileName3
```

"svn merge" is best used for reverting the repository to an older version. This is sometimes necessary one you realize that the latest commit actually didn't work and needs to be removed. In such cases, you call "svn merge" but also raise an "-r" flag, and provide the current revision number, and the revision number that you want to return to. After that, you pass in the files that you want changed, like this:

```
svn merge -r 102:101 fileName1 fileName2
```

Good SVN Usage Tips and Courtesy

SVN is not the most straightforward software to use from the start, but is very important. Here are a few tips to help you make sure that your SVN repository will not return errors when you try to make changes.

1. **Always update right before sending in a commit.** If some of the files you are trying to commit have been changed since you last updated, SVN will complain at you. Nothing happens and you can update and then commit anyway, but it saves time. This is also important so that you can see if your changes break anybody else's recent commits. **If any files are changed when you update, run all of your tests again so that you know that your changes are compatible with the most recent version of the repository.**
2. **Compile your code before sending in a commit** if you are coding in C, C++, Java, or any other language that uses a compiler. It is extremely frustrating for others if they are trying to work some on files, and your recent changes (which can't compile) make it impossible for them to run tests on the project.
3. **Communicate with project-mates before calling "svn rm"** – Don't scare your teammates by deleting files without their consent – they may be working on it, and be surprised when updating their local copies of the repository deletes their recent changes on those files. You can save them time and prevent panic by **letting them all know before deleting files**.
4. **Let people know when you have made important commits** – Project partners don't always know when to update their repositories, so let them know once you've committed so that they can continue to run tests with the most recent copy of the project
5. **Update often** – Always update your local copy of the repository just in case, so that you are always working with the most recent copy of the repository. Some versions of SVN will e-mail you when a change has been committed, but not all of them do that. Updating often allows you to always work with the most recent copy of the project
6. **Commit Modularly** – Change one kind of thing at a time with SVN – this is because changes sometimes have to be rolled back, and it is less painful if only one change was involved in each commit (this doesn't mean commit one file at a time. For example, when adding a new bit of functionality into a project that spans four files, commit them all at once. But don't fix those four files, and also refactor code in a fifth file that has nothing to do with that functionality in the same commit.
7. **Work in teams** – Never make a commit by yourself without somebody to code-review you. Everybody makes mistakes, and it is a lot better for your friends to find your mistakes early than for you to find it after hours of debugging because your software started screaming at you.

HELP MY SVN BROKE OMYGOODNESS I HATE YOU WHAT HAVE YOU DONE TO ME

Sometimes, SVN still breaks. In these more extreme cases, here are some tips on how to fix the repository:

1. Run “svn cleanup” – it may fix some issues
2. Try to run “svn up” – that may also fix some issues
3. Delete the conflicting local copy of a folder, and run “svn up” to get it back. (If you have local changes, copy them out to some other area of your computer so they’re not lost forever)

IF EVERYTHING BROKE AND YOUR LOCAL COPY OF THE REPOSITORY IS DESTROYED:
Just delete the entire repository folder, and checkout the repository again. Again, copy your local changes out, and copy them back in.