Lab 12: Collaboration with SVN

Introduction

Writing software is no longer the domain of the lone ranger programmer shut up in their office with their computer, emerging for a shower every week or so. Indeed, most projects in industry now require the programmer to work collaboratively in a team, sharing code with other programmers and managers, sometimes globally. Today, programmers generally work on just a small part of a much, much larger codebase, whether that codebase be a library, API, or custom codebase.

Tools have developed over the years to help programmers work together. One tool, Subversion (SVN), allows programmers to work collaboratively and also allows users to keep versions of their work in order so that previous versions may be restored in case of errors.

Lab 12 Preparatory Work

Introduction to VCS

Subversion (SVN) is a version (or versioning) control system (VCS, sometimes called revision control system). Version control systems allow you to keep track of changes to groups of files over time in discrete steps called revisions.

Programmers use VCS to keep track of changes to their codebase over time, but VCS can usually be used to keep track of any kind of files. For example, authors have been known to use VCS to keep track of versions of a book they are writing.

Introduction to Subversion

Subversion (SVN) is one of the most popular VCS available today. SVN is released by Apache as free software. There are other VCS available on the market, including commercial versions, each with their own strengths and weaknesses. The Wikipedia article on revision control systems in When you are done with the lab, you can delete the svn, john, and mary directories.

Lab 12 References is a good starting point for seeing what is available.

In addition to keeping track of changes, SVN can help many people work together on a project, by allowing people to work together on a group of files, uploading their changes to a central repository. Most importantly, SVN keeps track of the files to make sure that one user's changes does not overwrite changes from another user.

There are many approaches to implementing a VCS. For example, some approaches allow the exclusive checkout of files, permitting only one user to work on a file at a time. One of the limitations of this approach is that if two users need to work on the same file at the same time it cannot be done, which may slow development.

SVN uses the copy-modify-merge approach. This approach has each user checkout their own working copy of a file (or repository), and many users can check out the same version. As each user makes changes, the changes are committed back to the central repository. This approach requires users to regularly update their local version of the repository to make sure they are working with the current version. The major drawback of the copy-modify-merge approach is that conflicts can arise when users try to change the same file at the same time. There are many good approaches to resolving these conflicts, a process called *merging*. In this lab you will see one such approach that does not rely on automation to merge files.

Lab 12 In-Class Work

In this section you will be creating and using a SVN repository. Whenever you see *username*, this means type *your* thecity system username, not the word "username".

Create A Repository Directory

You do not necessarily need an external server to use Subversion, you can create and work with your own SVN repository.

Login to your account on thecity. If you have a directory named svn in your home directory, please rename it during this lab. Then, create a repository by typing:

```
svnadmin create svn↵
```

Now, look at what the svn directory contains by typing

```
ls svn -al↵
```

You'll notice that there are several directories and files residing in this directory. These are all the files that SVN uses for keeping track of the shared (main) repository.

Simulating Multiple Users

During this lab, you will be pretending to be two users named John and Mary. John and Mary are working together, sharing their code in the repository that you will create. From your home directory, create a directory for each of these users:

```
mkdir john↵
mkdir mary↵
```

When you're in directory john, you will be pretending to be user John. When you're in directory mary, you'll be pretending to be user Mary.

Check Out Working Copies

As previously mentioned, you must first check out a working version of a repository in order to begin working with the repository.

Working copy for John

Start by pretending that you are John. Change directories to john (`cd john`).

Check out a working copy of the repository by typing:

```
svn co file:///home/student/username/svn↵
```

Remember, *username* is *your* username on thecity. Note that the `co` in the above command is an abbreviation for `checkout`.

After a moment you should see the message "Checked out revision 0". This is SVN telling you that everything is OK. Each time someone commits files to the repository, the repository revision number increases by one.

See what the john directory now contains. You should see an svn directory. This svn directory contains a working copy of the svn repository, but it is a local copy in John's directory.

Change directories to svn and do a long directory listing including hidden files (`ls –al`). Notice that the directory is empty except for a hidden directory named .svn. In unix, directories whose name starts with a period are hidden from standard directory listings. The .svn directory contains information about the working copy of the repository, and generally should just be left alone.

Working copy for user Mary

Now pretend to be user Mary by changing directories to /home/student/*username*/mary. Follow the same steps as you did for John to checkout a second working copy of the repository for Mary. Note that the revision number is still 0, the revision number hasn't has not changed. This makes sense because you have not yet committed any files to the repository.

Using the Repository

Add the first file

Return to pretending to be John. Go to John's working copy of the repository by typing

```
cd ~/john/svn↵
```

Using your favorite command line editor (see Lab 2: IDEs and Editors), create a file named first.txt. In the file, add a line that says "first line - John". Save the file and exit the editor.

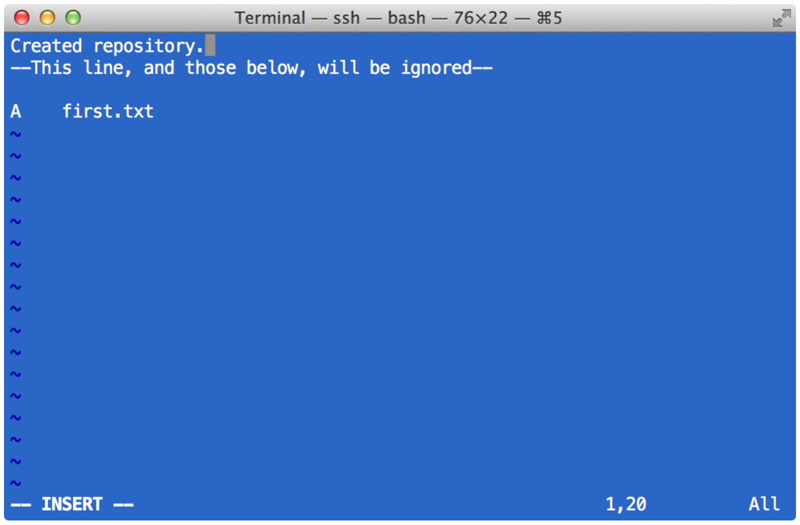
Now, add this file to the local working copy of the repository. From the command line type:

```
svn add first.txt↵
```

SVN will respond with a line that contains a capital A and the name of the file. The A stands for "Added". You have just added first.txt to John's local working copy of the repository. In order to commit this to the shared repository, from the command line type:

```
svn ci↵
```

The `commit` command is abbreviated to `ci`.



```
Created repository.
--This line, and those below, will be ignored--

A    first.txt
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                          1,20          All
```

Figure 41. Writing an SVN commit message in an editor.

After you press enter, an editor will automatically be opened containing some text, and a line below which you should not type (Figure 41). Each time you commit files to the repository, you must provide a *commit message*. This message is a summary of why this commit was made. Commit messages are an important way to keep track of what is happening during software development by tracking each group of changes to the codebase. Before SVN commits your files to the shared repository, an editor will open, and wait for you to enter a commit message. In this case, you added the first file to the repository, so a good message might be "created repository". Type the commit message above the line, then save the file and exit the editor. You should then see a message "Committed revision 1". So, John has now committed his local changes to the

Updating Mary's working copy

Now, pretend to be Mary. Go to Mary's working copy of the repository by typing

```
cd ~/mary/svn↵
```

Do a directory listing. Notice that there are no files in the repository directory, even though you just added a file to the repository as John. In order to get synchronized—get the changes from the shared repository to Mary's working copy—you must update Mary's local working version. Type:

```
svn up↵
```

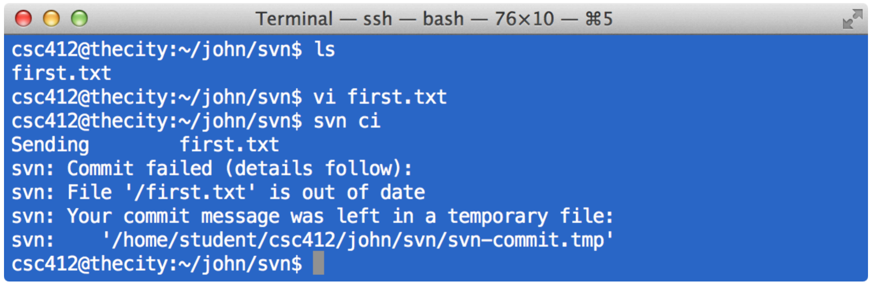
where `up` is an abbreviation for `update`. You should see the message "Updated to revision 1". Do a directory listing and notice that the first.txt file is now there. Mary's local copy is now in sync with the shared repository.

This modify-upload cycle continues during development. John or Mary can modify their files, and upload changes step-by-step with each commit to the shared repository.

When conflict happens

Edit the first.txt file. Add a second line to the file that says "second line - Mary", and save the file. Commit the file to the repository (adding an appropriate commit message!). You should now be at revision 2.

Change to John's svn directory. Note that John has an older version of the first.txt file, which does not include the changes that Mary just made. Without updating the repository (which is a mistake), edit John's version of the first.txt file, adding a second line that says "second line – John", and save the file. Commit the file to the repository (`svn ci`).



```
csc412@thecity:~/john/svn$ ls
first.txt
csc412@thecity:~/john/svn$ vi first.txt
csc412@thecity:~/john/svn$ svn ci
Sending        first.txt
svn: Commit failed (details follow):
svn: File '/first.txt' is out of date
svn: Your commit message was left in a temporary file:
svn:    '/home/student/csc412/john/svn/svn-commit.tmp'
csc412@thecity:~/john/svn$
```

Figure 42. SVN failing due to a file conflict.

You will see an error message from svn that the commit failed, because "first.txt is out of date (Figure 42). This means that the file that is in the shared repository has changed, and that if your file was uploaded, the change would overwrite the changes in the file in the shared repository.

To resolve this conflict, Mary's changes and John's changes need to be merged into a single file.

Resolving Conflict

SVN provides tools for automatically merging files. You are encouraged to carefully study how to use these powerful tools before using them (see Lab 12 References). Presented here is a manual approach for merging a file that has a conflict.

To recap, Mary and John have both modified a file in their local version of the repository. Mary committed her change, and John cannot commit his change because Mary's changes would be lost. What you will do is rename John's local version of the file, and download Mary's version of the file, then manually combine the changes.

Make sure you are in the /home/student/*username*/john/svn directory. Then, rename the first.txt file to first_john.txt by typing:

```
mv first.txt first_john.txt↵
```

Delete or rename the svn-commit.tmp file:

```
rm svn-commit.tmp↵
```

When you now update the repository, Mary's version of the file will be downloaded from the shared repository. To update your local copy type:

```
svn up↵
```

You should see some messages, including "Restored 'first.txt'" and "Updated to revision 2."

Now using your favorite editor, edit the first.txt file and add the changes from the first_john.txt, so that the new file contains the following lines:

```
first line – john
second line – mary
third line – john
```

Save the file and exit the editor. Now, commit the change to the repository (svn ci). A good commit message in this case would be something like "Merged my changes to second line with Mary's changes in first.txt".

Remember!

You should write a unique, descriptive commit message every time you commit!

When you are done with the lab, you can delete the svn, john, and mary directories.

Lab 12 References

Revision control overview: http://en.wikipedia.org/wiki/Revision_control

Subversion: http://en.wikipedia.org/wiki/Apache_Subversion

Comparison of VCS:

http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

Version Control with Subversion, a.k.a the Red Bean Book, http://svnbook.red-bean.com/