CSC207 Lab 1 — Unix Shell and Subversion

To earn lab marks, you must arrive on time and actively participate for the entire session.

1 Overview

This week, you are going to work with Subversion using a UNIX command line shell.

2 Choose a driver and navigator

In these labs, you are encouraged to work in pairs. You and your partner together will be able to figure out problems better than you would individually. Find yourself a partner. If you have trouble finding one, let your TA help you. This partnership is only for today's lab.

We strongly advise you to form a partnership with a classmate who has a similar level of background. For example, if this is your first time working in a Linux environment, we suggest that you partner with someone who is also new to Linux.

In all the labs, we will use the terms *driver* and *navigator*. Here are the definitions of the two roles:

- **Driver:** Types at the keyboard. Focuses on the immediate task at hand.
- **Navigator:** Thinks ahead and watches for mistakes.

In lab handouts, we'll often refer to you as `s1` and `s2`, and `s1` will be the first driver.
Tip: If you are new to the CDF labs, you should be `s1` to get practice logging in while `s2` helps.

3 Log in and get things set up

`s1` drives and `s2` navigates.

You can't both be logged in at once, so let `s1` be the person to log in. If necessary, `s2` should offer assistance with this, but here is a very important rule for this and all other labs: **Make sure neither of you learns the other's password while you work through this process!**

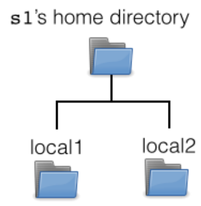
Also, during this lab, `s2` does *not* log in and does *not* check out a repository. Both of you work *in `s1`'s account on `s1`'s SVN repository.* **Note**: If you are `s2` and wish to try this exercise in your own account, you can do so after the lab.

After logging in, open a new terminal window.

You need to use some basic Unix commands in order to complete this lab. These commands are listed on the second handout and the navigator should look them up.

1. Change to `s1`'s home directory.
2. Create two directories named `local1` and `local2` in `s1`'s home directory. These two directories will be used later for checking out local copies of `s1`'s repository.

At this point, the directory structure should be as follows (there are other files and directories in s1's home directory that are not shown below):



4 Subversion (svn)

We will begin by telling Subversion which editor to use use, by issuing the following command:

```
setenv SVN_EDITOR nedit
```

Each of you has a subversion repository for use in this course. It can be accessed by you, by the instructor, and by the TAs, but not by other students. `s1` should login to `https://markus.cdf.toronto.edu/csc207-2015-01` and click on "lab1: Unix Shell and Subversion". In the subsequent steps, when the instructions mention checking out the repository, use the SVN repository URL given under "URL to your group's repository" on MarkUs. It will be something like:

```
https://markus.cdf.toronto.edu/svn/csc207-2015-01/your_CDF_username
```

1. Change directories to `local1` and check out a working copy from s1's repository.

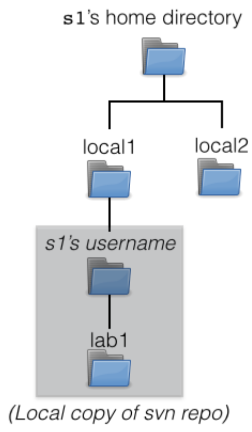
   If `s1` has never checked out a repository from `markus.cdf.toronto.edu`, `s1`'s CDF password will be required.

   Subversion will warn you that it is about to store your password unecrypted. Agree by typing **yes** at the prompt.

   At the end of this process, you should see something like "Checked out revision 1".

2. There will be a directory in your repository called `lab1` that was created by your instructor. Change directory to the directory `lab1` that is inside your local copy of your repository.

   The directory structure should now be:

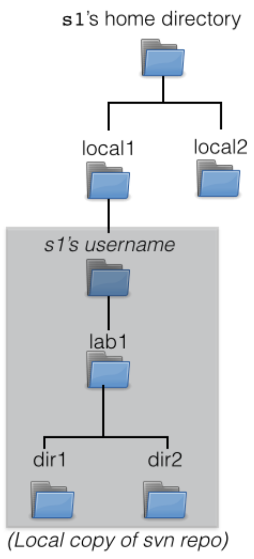
3. Create two directories, `dir1` and `dir2`, inside the directory `lab1`, and add them to the repository. Recall that the Subversion's `add` command marks a file for addition to the repository, but does not modify the repository. Run the `commit` command too.

Log messages should be meaningful. An empty message or a message like “made some changes” is not useful. A helpful message might be “Renamed method foo to bar”, or “Removed the 3-argument constructor”.

The repository maintains version numbers on a per-commit basis. When a new version of a file (or several files) is committed, the version number of all files is incremented.

Note: If you don’t specify a log message, the editor that we specified above pops up to allow you to create this log message.

The directory structure should now be:



4. Inside the directory `dir1`, create a file named `myfile.txt` with a text editor (e.g., by running `nedit myfile.txt`), save it in the `lab1` directory, and add it to the repository. Commit your changes.

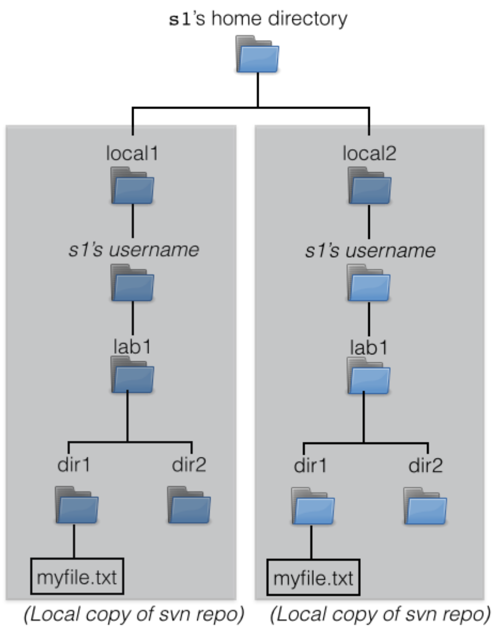
5. Edit `myfile.txt`: put the names of `s1` and `s2` (and anyone else you are working with) in the file and *at least two blank lines* after the last name. Commit the changes to the repository.

   Switch roles: `s2` drives and `s1` navigates.

6. Open another terminal window. Change to `s1`’s home directory, then change to the `local2` directory, and check out `s1`’s SVN project.

   Note: we are still in `s1`’s account.

   The directory structure should now be:



7. Make some changes to `myfile.txt` in `local2`, but *leave the group member names alone*, and commit the changes.

   Switch roles: `s1` drives and `s2` navigates.

8. Update the local copy in `local1`. Since you just changed `myfile.txt` in `local2` and committed the new version to the repository, you should see the changes in `local1`.

9. Now, you will create a set of conflicting changes. To do so, edit `myfile.txt` again in `local1`, *changing the first line*, and commit the changes.

10. Before updating `local2`, **switch roles (`s2` drives and `s1` navigates)** and have `s2` change *the same line* in `myfile.txt` inside `local2`. Make different changes this time, so that there will be a conflict.

11. Try to commit the changes. This should fail, since the file `myfile.txt` in `local2` has not been updated.

12. Issue an `svn update` command. Read the feedback from the command carefully.

    Choose to edit the file `myfile.txt` to resolve the conflict and finish the update.

    Switch roles: `s1` drives and `s2` navigates.

13. In the terminal window containing `local2`, display the status of working copy files and directories. With `--show-updates`, the `status` command adds working revision and server out-of-date information. With `--verbose`, it prints full revision information on every item.

14. Look at the revision history of `myfile.txt` using the `log` command.

15. What needs to be done now to make sure `local1` and `local2` both contain the latest version of `myfile.txt`? Complete the necessary steps. Show the TA your work.