ECE 364: Software Engineering Tools Lab

Lab 00: Subversion Source Control & Introduction to Bash

Name: ___________________________________ Class Login: ee364___

Part 0: Account Setup

1. Your TA will begin the lab with a brief lecture on lab policies and rules followed by an introduction to the lab's computers. Listen carefully.

2. Log into your course account and open a terminal window (Go to "Applications" > "System Tools" > "Terminal"). If you have difficulty, ask the TA. At the shell prompt, execute the command:

   ```
   ~ee364/bin/bash_init364
   ```

   This will set up your course account.

3. Now change your password with the `passwd` command (Be careful with this step because we cannot view or reset your new password if you forget it).

> **IMPORTANT**
>
> Course staff *DO NOT HAVE ACCESS* to your password. If you forget it, you must submit a request to esite@ecn.purdue.edu with your login account to have it reset. If this happens at the beginning of the lab, odds are good that the password will not be reset in a timely manner. Forgetting your password is not considered an excuse for missing a lab. As such, regular grading policies apply.

4. Once you have changed your password, run the `exit` command to close the terminal window.

5. Make your terminal window a login shell with the following steps:
   - Open a terminal window as in step 2.
   - Click "Edit."
   - Select "Profile Preferences..."
   - Go to the "Title and Command" tab.
   - Check "Run command as a login shell."
   - Click "Close."
   - Close your terminal window and log out of the computer ("System" > Log out).
   - Log back into the computer with your ee364 account and new password and open a terminal window as before.

You should see the following at the top:

```
Terminal is <something>
<Your login> on <hostname> -- <arch> <date>
SVN repository is <svn repository URL>
Set-up environment for ee364
```

You should now be able to use tab-completion as well as the up arrow to access previous commands.

6. Raise your hand and wait for the TA to sign below BEFORE continuing:

____

Part 1: Subversion (A Version Control System)

Subversion (svn) is a software tool that helps maintain and control the proper accessing of revisions of an ASCII text file (i.e, source code). The advantages of Subversion include:

- The ability to recover any previous revision of a file.
- Maintenance of a complete history of a file and file security, in terms of both user access and simultaneous access.

These attributes are particularly useful in group project work, where many people may need to access and modify the same file. It is also useful for maintaining "snapshots" of work in progress even if you are working alone, as you will be in this course. Keep in mind that Subversion may be used in future courses as well, so it would be prudent to learn how to properly use it now instead of later.

IMPORTANT

The grading scripts in this course rely on your code being checked into SVN. Failure to use SVN in this course will result in failure of the course!

Using SVN:

1. Type "which svn" on your command line. It should display one of the following paths:

```
/usr/local/bin/svn
```
or
```
/usr/bin/svn
```

This is the location of the actual binary program "svn". If you do not see the above path, CONSULT YOUR TA!

2. SVN comes with built-in documentation. Execute the command below to see the possible uses:

```
svn help
```

Similar to svn's help feature, manual pages are provided for most commands on a UNIX based system and may be accessed with the command 'man <command>', i.e.:

```
man ps
man wget
man grep
```

Many commands also have a help option that is invoked by adding `-h`, `--help`, or occasionally `-help` after the command. Which one exactly, depends on the specific command. This help message is typically less in-depth than the man page.

```
ls --help
wget -h
ping –h
```

You are encouraged to read the manual pages for UNIX commands unfamiliar to you before asking the TA questions.

3. Now, make sure you are in your home directory by entering the command:

```
cd ~
```

4. In order to gain access to the SVN repository, you must create and upload a public key. A public/private key pair is a more secure alternative to the username/password method of authentication you are probably most familiar with. To generate the key run the command:

```
ssh-keygen
```

It will ask you where to save the file. **Use the default value by pressing enter without typing anything**. It will then prompt you for a password. Do not leave this blank. When you are finished, type:

```
ls .ssh
```

You should see `id_rsa` and `id_rsa.pub` listed in the directory. The first file, `id_rsa` is the private key file protected by the password you entered in. This file must remain private and should not be distributed. The second file, `id_rsa.pub` is the public key and can be distributed to any system you would like to have access to (the repository, in this case.)

IMPORTANT

The password used for this file is in no way connected to the password used to access your course account. They may be different and changing one will not change the other. If you forget this password, it cannot be recovered or reset by anyone. In this case, you will have to make and upload a new key pair by repeating steps 3-5.

5. To upload the public key, run the command:

```
sendkey
```

You will be prompted for your password. This is the password for your account, not the one for your key, if they are different. If this command reports an error, please contact a TA.

You now have access to the repository. A template of your home directory has already been created for you in the repository. Run the following command **(Note: There is a " ./ " at the end of the command)**:

```
svn checkout $SVNREPO ./
```

This command creates a working copy of your section of the repository in your home directory. It will ask you for the password for your private key. If you get an error message, please contact your TA.

Run the command "`ls`". You should see directories such as "Lab01", "Lab02", etc. Raise your hand and wait for the TA to verify your directory structure.

TA: ____

6. Go to your "Lab00" directory by typing:

```
cd Lab00
```

Create a file named "svntest" using any editor that you prefer (e.g. vi, emacs, gedit, nano...). Don't forget that if you want to create an empty file quickly, the "`touch`" command will be very helpful.

7. Mark the file you just created for inclusion in the repository by running:

```
svn add svntest
```

You should see:

```
A    svntest
```

This indicates that "svntest" is now marked for addition, but it is important to remember that it *HASN'T ACTUALLY BEEN ADDED YET*. To do so you need to run:

```
svn commit svntest
```

You will see some editor (like "vi") opened for you to enter a comment. This is generally used to describe, at a high level, any changes made between the previously checked in version of the file and the current one.

In case you are not familiar with "vi", press "`i`" to enter insert mode. Enter something interesting. When you are done, press "ESC" to enter command mode and type **"**`:wq`**"** to save and quit. You will be prompted for your key password.

If you see the message "`Committed revision <revisionNumber>`", then your file has been successfully added to the repository, and may be graded.

8. Create two new files called "data1" and "data2". Open "svntest" with a text editor and make some changes to it. Then enter the command:

```
svn status
```

What do you see and what do you think it means? (Hint: use svn's manual by running "`man svn`")

____

____

Now execute the following commands:

```
svn commit svntest
svn status
```

What is the output of "`svn status`" and what do you think it means?

____

____

In each lab you will likely have many files that are not very important, such as files we provide for sample input and output, or test cases you create yourself. There is no reason to add these to the repository. In general, you only need to add the source files we specifically ask for.

9. Now add keyword/identification markers to your file, e.g. "`$Author$`". Refer to lecture notes for other keywords that can be used. To view the lecture notes, execute the command "`lecture N&`" where N is the lecture number you wish to view. The ampersand at the end is not strictly necessary, but it allows you to continue running commands from the same terminal window while viewing the lecture slides.

   DO NOT USE THE WEBSITE TO ACCESS LECTURE NOTES. INTERNET ACCESS IS STRICTLY PROHIBITED DURING REGULAR LAB.

10. Modify and commit the file using the "`svn commit`" command. Open the file in an editor, and note any changes in the markers. For example, the marker "`$Author$`" should have been replaced by your username after the commit.

11. At this point you may have noticed that using SVN can involve typing your password a lot of times. As a convenience you may cache your password using a program called "ssh-agent" so that you only have to enter it once. To use "ssh-agent", type the command:

    ```
    ssh-add
    ```

    It will prompt you for the password for the key file. It will store the key in a keyring for you, and you do not need to enter it again in current session. (Note that "ssh-agent" terminates when you logout, clearing any passwords stored in the keyring. Also, it will clear your passwords stored in the keyring in three hours.)

Part 2: Advanced Subversion

1. View the contents of "svntest" by typing:

   ```
   cat svntest
   ```

   Now type the commands:

   ```
   svn update –rPREV svntest
   cat svntest
   ```

   What changes do you notice?

   ____

   The "`-r`" option may be used to recover previous versions of a file (so you don't have to make backups by hand!). You can also specify a revision number if you remember it (The command, "`svn log`" might help when you need to find out a revision number), or a date in the format like "`{2009-06-27 14:19:33}`".

   Example:

   ```
   svn update -r"{2009-06-27 14:19:33}" svntest
   svn update -r1022 svntest
   ```

   Without the "`-r`" option, update will make sure that the file is the most recent version checked in, possibly undoing any uncommitted modifications you have made. However, if all you want to do is to undo uncommitted modifications, the command "`svn revert <filename>`" will work more reliably.

2. Type the following command and note the output:

   ```
   svn log svntest
   ```

3. Create a file called "test2". Add it to the repository and commit it. Rename the file to "test3" using the command:

   ```
   svn mv test2 test3
   ```

   The "`svn mv`" command, like its UNIX counterpart, is used to move or rename files.

   Then commit changes to the repository using:

   ```
   svn commit
   ```

   Now delete "test3" from the repository using:

   ```
   svn rm test3
   ```

   Note that the file is removed from both the repository and the working directory. Also keep in mind that the file "test3" is not actually gone from the repository. You can easily recover "test3" and even "test2" using the "`svn update`" command discussed above.

4. Run the commands:

   ```
   svn mkdir test_dir
   ls
   ```

   A directory has been created in the repository and in the working copy. This is a shorthand for the commands:

   ```
   mkdir test_dir
   svn add test_dir
   svn commit test_dir
   ```

5. Create a file called "test4". Make it executable and note the permissions by running the commands:

   ```
   chmod +x test4
   ls -l test4
   ```

   Note: "`ls -l`" displays a string with information on the permissions of the file such as:

   ```
   -rwxr-----
   ```

   The first character is the file type. It is `—` for regular files and `d` for directories. The next three characters represent the permissions for the owner of this file. The next three represent the permissions for users in the same group as this file, and the last three are the permissions for everyone else. In the example above, the file's owner may read, write, and execute the file, while users in the same group may only read the file and execute it. Any other users have no access at all to the file. When we ask you what a file's permissions are, it is usually sufficient to write down this string.

   Now add "test4" to the repository and commit it. Delete "test4" from the working copy by running the command:

   ```
   rm test4
   ```

   Recover the file from the repository and note its permissions by running:

   ```
   svn update test4
   ls -l test4
   ```

Have the permissions changed? If so, how?

____

Now create another file "test5". DO NOT make it executable. Add it to the repository and commit it. NOW make it executable using “`chmod +x test5`”. Modify "test5" and commit the changes. Remove "test5" with "`rm test5`" and restore it with "`svn update`".

Have the permissions changed? If so how?

____

You will have to make your scripts executable in order to run them, and the moral of the story is to do so BEFORE committing them to the repository. If you forget, Subversion will store the wrong permissions, and the graders will see a non-executable copy. In this case, however, you can fix it by running:

```
svn propset svn:executable '*' test5
svn commit test5
```

6. Run the command “`ls -l test5`” and verify that it is now executable.
   Raise your hand and have your TA sign below:

____

Part 3: Useful Shell Shortcuts

1. Execute the following commands:

   ```
   echo "Hello World"
   ```

   Now press the Up arrow, and press Enter. What happened when the Up arrow was pressed?

   ____

2. Now execute the following commands:

   ```
   ls -l
   ```

   Now press the Up arrow, and press Enter. What happened NOW when the Up arrow was pressed?

   ____

3. Execute the following commands IN ORDER and describe their outputs:

   1. `!e` ____
   2. `echo "Welcome to ECE 364"` ________________________________
   3. `whoami` ____
   4. `!e` ____
   5. Is there any difference between the outputs of commands in 1 and 4? If so, explain it.

   ____

Part 4: Copying files, globs, and brace expansion

The `'cp'` command can be used to copy files. For example, to copy a file named source to a file named “dest”, you would use the command `'cp source dest'`

This is useful for copying one file, but many times you will want to copy a group of files. You can use 'globs' (refer to lecture notes) to do this. The asterisk (*) can be used to match many files to copy. Run the following command:

```
cp ~ee364/labfiles/Lab00/* ~/Lab00/
```

This will copy all files from the directory ~ee364/labfiles/Lab00 to your Lab00 directory. Now, execute the following commands and write down a **description** of which files are listed (do not simply list the output of the command):

1. `ls *.c` ____
2. `ls a.*` ____
3. `ls a*` ____

Brace expansion is a useful feature that can simplify many tasks. (Refer to lecture notes.) Write down the output of the following commands:

1. `echo {1..5}` ____
2. `echo {a..f}` ____
3. `echo {a..c}{3..2}` __________________________________
4. Write down a one-line command that will display all ece364 logins. The logins are of the form **ee364x#**, where x is a letter between a and j inclusive, and # is a number from 1 to 15 inclusive.

   ____

Part 5: Brief Introduction to Bash

1. Write a Bash script that checks the parameters passed into the program. Go to your Lab00 directory and create a file called ‘Script1’. Open it in an editor and add the following as the very first line:

   ```
   #! /bin/bash
   ```

   This is called a shebang statement, and this causes the script to be executed under the Bourne-again Shell. **You MUST include this line at the top of all your Bash scripts that you will write this semester.**

   Now put the following lines in your script:

   ```
   Num_Of_Param=$#
   Param_Values=$@
   echo "The program received $Num_Of_Param parameters"
   echo "The parameters are : "
   echo "$Param_Values"
   ```

```
echo "The 2nd parameter is \"$2\""
echo "The 4th parameter is \"$4\""
exit 0
```

Remember that "`$#`" is a special variable that contains the number of parameters passed into the program and "`$@`" contains the values of all parameters passed into the program.

On the first line we have defined a variable "`Num_Of_Param`" and assigned it the value (contents) of "`$#`". **Note that when we assign variables in Bash, there cannot be any space around the "=".**

On the third line, the echo statement prints the sentence in quotes on the screen. The "`$`" in front of the variable means we want the value of the variable instead of its name.

On the fifth line, the echo statement prints the value of variable "`Param_Values`" that has been assigned the values of all parameters passed to the program.

On the sixth and seventh line, the echo statement prints the value of the 2nd and 4th command line parameter passed to your new script. Experiment what happens when you provide **no arguments** on the command line. What is the value of the command line argument variables? ______________________________________

Note that on the last line we call the exit command with a return value of 0 to indicate a successful termination of this script. You should always end your scripts with an exit command and an appropriate return code.

Save your file and try it out. If everything works as expected, add and commit it to SVN.

2. Write a simple Bash script that takes a positive integer as an argument and prints all integers from this integer down to 0 (inclusive).

Script name:	Script2
Number of Arguments:	1
Output:	STDOUT
Input:	None
Exit code:	0
Description:	Script2 takes a positive integer as an argument and prints all integers from this positive integer down to, and including, 0.

Sample output:

```
$ Script2 4
4
3
2
1
0
```

Note: "`$`" represents the shell prompt in this example.

HINTS: Use a loop counter to print the integers. Initialize the loop counter to the integer passed as the argument. Refer to lecture notes for some ideas.Make sure your program works, and that you've added and committed the file to SVN. Raise your hand and demo your program to the TA. Have him/her sign below:

____

3. Copy Script2 to a new file named Script3. Modify Script3 so that, on each iteration of the loop, the script checks whether the number is a multiple of 3. If it is, print it. Otherwise, simply continue to the next iteration.

Script name:	Script3
Number of Arguments:	1
Output:	STDOUT
Input:	None
Exit code:	0
Description:	Script3 takes a positive integer as an argument and prints all integers **that are divisible by 3** from this positive integer down to, and including, 0.

Sample output:

```
$ Script3 11
9
6
3
0
```

Note: "`$`" represents the shell prompt in this example.

HINTS: Use a loop counter to print the integers. Initialize the loop counter to the integer passed as the argument. The '%' operator will be useful in determining if an integer is divisible by another integer.

Make sure your program works, and that you've added and committed the file to SVN. Raise your hand and demo your program to the TA. Have him/her sign below:

____

Part 6: Verifying the Final Commit

You should commit all the scripts/programs you have written during the lab if you have not already done so. The TAs will grade only committed versions of your programs/scripts. Failure to commit your scripts to SVN on time will likely result in a grade of 0.

To verify which files have been committed to SVN, you may run the following sequence of commands:

```
svn update
```

```
svn list
```

It is a good idea to perform this check for every lab and prelab.

What is the penalty if you forget to commit your code to SVN?

____

The End!

This concludes your first lab for ECE 364. This lab was *very easy*. If you struggled with it you should speak to the instructor immediately. Future labs will be much more involved and take longer to complete.

Finally, please remember to always logout of your account and not keep your session active. This can drain the server resources, and may result in work loss.