Lab 01 – Welcome to SENG265

Today we will experiment with the Linux file system and the vim text editor. By the end of the lab you will be able to:

- Start up a Terminal on your workstation;
- Navigate the linux filesystem using the **`cd`** command;
- Create directories with the **`mkdir`** command;
- Rename files using the **`mv`** command;
- Print the contents of files using the **`cat`** command;
- Remove files using the **`rm`** command;
- Copy files using the **`cp`** command;
- Edit file contents using the Vim text editor;
- Change file permissions using the **`chmod`** command;
- Run shell scripts at the command line;
- Create shell scripts using for loops;
- Create shell scripts using if commands;
- Create shell scripts that print to stdout and redirect the output it to a file;
- Create shell scripts that read from stdin and redirect the input to read from a file.

Part 1 – Unix commands and file editors

Launching a Terminal

Once you've logged into the workstation, try launching a *terminal*. The easiest way to do so is by right-clicking on the desktop, and choosing **`Open Terminal`** from the drop-down box. You can also do so from the **`Applications -> System Tools -> Terminal`** menu.

You'll see a prompt that looks something like this:

`[your_netlink_id@ugls23 ~]` (the 23 might be a different number, and of course you'll see your own netlink id.) This is your *bash prompt* - it's waiting for you to enter a command. In the prompt, you'll see your user name, the name of the particular workstation you're working at, and the **~** symbol (which is called the 'tilde', and is the character in the upper left corner of your keyboard.)

This **~** character stands for your *home directory*, which is the particular location in the linux filesystem hierarchy that belongs to you.

Navigating the filesystem

You can always use the **`pwd`** command to show your current location in the file system hierarchy. For example, if you type **`pwd`** and hit enter, you'll see that you are located in your *home directory* at **`/home/<your_netlink_id>`** (remember, this is also called **~**).

You can change your current location (which we call the *working directory*) by using the **`cd`** command. Try navigating one step 'up' to the parent directory, by typing the command **`cd ..`** (note the space between the command and the periods). This puts you in the directory 'above' your home directory, called **`/home`** - you can verify this by using **`pwd`** again.

Try this again by typing **`cd ..`**. You'll see that you're in the **`/`** directory, which is as far 'up' the directory tree as you can go - trying to do **`cd ..`** again won't do anything. We call this 'uppermost' directory the *root directory*.

You can return to your home directory at any time by typing **`cd ~`** (just typing **`cd`** by itself also sends you home).

Listing files and subdirectories

We use the **`ls`** command to list the contents of a directory. Try it in your home directory, and then go back up to the root directory (you can use **`cd /`** to get there directly) and try again from there.

You can vary the command to give you more information. **`ls -a`** will print the names of all directory contents, even those that are normally hidden, while **`ls -l`** gives you more information about each entry, such as file permissions and modification dates.

If you want to get more information about the various options each command supports, you can use the **`--help`** option with the command. For example, to see all the different variants of the **`ls`** command, try **`ls --help`** .

Creating a directory

We'll need a spot in our home directory for our work, so return there if you need to (by using the **`cd ~`** command). We'll create a directory inside our home directory with the **`mkdir`** command - type **`mkdir SENG265`**. If you then list your home directory's contents, you'll see your new directory alongside the default ones.

Copying Lab1 into your SENG265 directory

Roberto has put a copy of the files we need for our lab today in *his* home directory, in **`/home/rbittencourt/seng265/lab-01`**. We'll copy those files into our new SENG265 directory.

Go into your newly-created SENG265 directory, and use the following command:

```
cp -R /home/rbittencourt/seng265/lab-01 .
```

(Note the period at the end, you'll get an error message if you don't include it.) This command means *copy*, *recursively* the contents of the **`/home/rbittencourt/seng265/lab-01`** directory, and put that copy *here* (just as the **`..`** double period means 'one level up', the **`.`** single period means *here*).

Now list your directory contents again, and you'll see that multiple new directories have been created, named after different colours. Each of these directories contains a different exercise for us to try today.

Exercise 1: Moving or renaming a file

The **`blue`** subdirectory contains a file named **`rename_this_file.txt`**. Go into that directory and try renaming the file, using the **`mv`** command (**`mv`** is a command we use to move files from one place to another, but if we use it to move a file into the same directory it's currently in, we can give it a different name, so it sort of works to rename a file).

The syntax for the **`mv`** command looks like this:

```
mv <source_filename> <destination_filename>
```

You can specify different filenames if you want to rename the file, and if you wanted to move the file to a different directory, you could specify the destination using directory names, **`/`** and **`..`** symbols as necessary.

Remember, if you need more information about how a command works, try the **`--help`** option after the command name.

Exercise 2: Moving or renaming a directory

You can move or rename directories the same way you did so with files – with the **`mv`** command. The directory **`yellow`** is mis-spelled; try correcting its spelling with the **`mv`** command.

Exercise 3: Printing the contents of a file

You can find out what's in a file by printing its contents to the screen, using the **`cat`** command. Try printing the contents of **`green/print_this_file.txt`** to the terminal.

Intermission: shortcuts

Typing entire commands over and over can become tedious. Take advantage of two labour-saving features of the shell: tab-completion and shell history.

You can specify a filename by typing the first few characters of the name, and hitting the **`<TAB>`** key. If the shell can guess what you're trying to say, it will complete it for you. For example, instead of typing **`cat print_this_file.txt`**, I can type **`cat pri`**, and hit **`<TAB>`**. Since there's no other file in the directory that starts with the letters **`pri`**, the shell can figure out what I'm trying to do and fill it in for me.

If I've used a command in the past, I can re-type it quickly by using the up and down arrows. This cycles through my command history, and I can hit enter when I get to the command I want to repeat, or edit it if necessary first.

Exercise 4: Copying a file

We used the **`cp`** command when we copied the lab over to our home directory. Let's try using this command again, this time with a single file. The directory **`red`** has a file in it called **`copy_this_file.txt`**. Make a copy of it in the same directory, called **`this_file_backup.txt`**. Once you're done, you'll have two files in the **`red`** directory, one with the old name, and one with the new name. These are now two completely separate files - using the **`cp`** command is a good way to back up your files while your work.

Exercise 5: Removing (deleting) a file

If we want to permanently delete a file or directory, we use the **`rm`** command. This is a dangerous command - Linux has no built-in 'trash folder', so anything you remove is gone forever. Always double-check before you use **`rm`** that you have the filename correct and that you're completely sure. Sometimes it's safer to move a file to somewhere out of the way or rename it, rather than delete it entirely.

The **`purple`** directory has a file we want to delete, called **`delete_this_file.txt`**. Use **`rm`** to get rid of it.

If you wanted to delete an entire directory, you can use the **`rm -r`** variant of the command. Be especially careful with this command - it will remove not only the directory you use it on, but all the subdirectories of that directory!

Exercise 6: Editing a file

We'll use the Vim text editor to modify our files. The directory **`orange`** contains a file named **`program.sh`** which needs to be edited. We can edit this file with the command **`vim program.sh`**. (If the file didn't already exist, we could create an empty file with the **`touch`** command first).

`program.sh` is a *shell script* - it's a program designed to print out a congratulations message. Unfortunately, the code is commented out, so we'll need to modify it to make it work.

When Vim starts, it's in COMMAND mode - in this mode, most of what we type won't appear in the document. In order to modify the document, we'll need to switch to INSERT mode, by pressing the **`Insert`** key or lower-case **`i`**. This puts us into INSERT mode, and we can modify the document.

Use the arrow keys to move around and modify the program - the first two lines are commented-out with the **#** symbol. Remove these two symbols.

It's time to save the file and exit Vim. To do so, hit the ESC key to get back to COMMAND mode, and then enter **`:w`**. Hit enter, and you'll see a message that **`program.sh`** was written. Now you can quit, by typing **`:q`** and then enter. You'll be returned to the command line.

If you want to both save the file and quit Vim in one command, you can use **`:wq`** to do both at the same time. If you want to quit without saving, use **`:q!`** instead.

Vim is unlike most text editors and may be frustrating at first. When in doubt, check what mode you're in - does the text INSERT appear at the bottom left? Then you're in INSERT mode and can modify the document. Do you want to save or quit? Hit ESC to get to COMMAND mode and issue your save/quit commands from there.

Exercise 7: Modifying permissions

To try to run **`program.sh`**, issue the command **`./program.sh`** (this means execute the file named **`program.sh`**, which is located *right here* - remember, **`.`** means *here*).

You'll receive a Permission denied error, which means that the shell still thinks **`program.sh`** is just a text file, not an executable program. We need to change its permissions to executable.

Use the **`ls -l`** command to view its current permissions. Right now, they appear as **`-rw-rw-r--`**, which means that the owner, groupmates and others can read the file, and that the owner and groupmates can write to it, but no-one can execute it (the letter **`x`** doesn't appear). You need to change the permissions to give the owner (you) permission to execute the file. To do so, you'll need the **`chmod`** command - you want to *add* the *execute* permission for the *owner*.

To do this, we'll use **`chmod u+x program.sh`**. Try this command, then re-display the permissions. You'll see that they have changed to **`-rwxrw-r--`**, which means this is now an executable program!

Re-run the program with **`./program.sh`** - You should see:

```
Congratulations! You've successfully completed Part 1 of Lab 1!

The instruction you executed was: ./program.sh
```

Part 2 – Shell scripts

Here you will exercise some simple shell scripts. Start by creating a **`silver`** directory. Make sure to demonstrate your work to your lab TA before you end the lab. This is how you will be assessed.

Exercise 8: Creating files with for loops

Inside the **`silver`** directory, create a shell script named **`create_files.sh`**. Remember to start the script with a shebang to call **`bash`** where it resides (**`/bin/bash`**). In the script, create a directory named **`files`** under the current directory, and then enter that directory. Use a for loop to create 10 files with the **`touch`** command. The files will be named from **`silver0.txt`** to **`silver9.txt`**.

Typical syntax of a for command with a counter is:

```
for (( counter=init ; counter<end; counter++ ))
do
        commands
        # to recover the value of counter, use $counter
done
```

(Hint: to gain confidence on your coding, you can initially replace the **`touch`** command by an **`echo`** command, which helps you debug your script.)

Exercise 9: Deleting files conditionally

Inside the **`silver`** directory, create a shell script named **`delete_files.sh`**. In the script, enter the existing **`files`** directory, and combine a for loop with an if command to remove the files **`silver`*`n`*`.txt`** files indexed with even numbers.

Typical syntax of an if command is:

```
if [ condition ]
then
        commandsA
else
        commandsB
fi
```

Working with arithmetic may be awkward in shell scripts. You may use the **`expr`** command to evaluate an arithmetic expression, but you should better use a **`$()`** to enclose your **`expr`** command. Moreover, relational comparisons use a different format (**`-eq`**, **`-ne`**, **`-gt`**, **`-ge`**, **`-lt`**, **`-le`**).
(Hint: to gain confidence on your coding, you can initially replace the **`rm`** command by an **`echo`** command, which helps you debug your script.)

When you're done

Type **`exit`** to close the terminal.

Remember to log out of your workstation before you leave (the power symbol is in the upper-right corner of the screen, and you'll select the padlock icon from the box that pops up). Make sure your attendance for Lab01 has been recorded by your TA, and we'll see you next week.