# Lab 0 – Linux Tutorial

The goal of this lab is to introduce the Linux environment and some tools you will use throughout the semester. Read through this tutorial and practice the steps multiple times. Complete the lab by performing the steps in Part VI.

## Part I – Logging in to the CSE using SSH and SFTP

Locate and start the *PuTTY* application. For the *hostname*, enter one of the following hosts, making sure the *Connection Type* is SSH and the *Port* number is 22.

cse01.cse.unt.edu    cse02.cse.unt.edu    cse03.cse.unt.edu   etc.

You can use up to cse09.cse.unt.edu

Click *Open* to connect to the host. You will be prompted for your username and password. This is your EUID username and password. Note that your password will not appear as you type. After logging in, you will be at the Linux command line prompt.

Now, locate and start the *WinSCP* application. Click *New* to start a new session. Like with *PuTTY*, enter one of the hostnames above into the *Host name* field, making sure the *File protocol* is set to SFTP with *Port number* 22. The username and password is your EUID username password. Once entered, click *Login* to connect to the server.

You now have two applications that will allow you to access the CSE servers. *PuTTY* is used to access the command prompt so you can perform operations, write scripts, compile and run programs, etc. *WinSCP* will allow you to transfer files between your computer and the server using drag and drop.

## Part II – Basic Linux Operations

Listing Directories

Listing directories is done using the **ls** command. Type **ls** at the command prompt and hit enter to see the files and subdirectories in your directory, which is known as your home directory. You can get more detailed information by using additional options. For example, try the following commands and note the differences.

- `ls –a`
- `ls –l`
- `ls –al`

Now try listing files in another directory. Enter the following commands:

- `ls ~dmk0080/public/1040/labs/zero -l`
- `ls ~dmk0080/public/1040/labs/zero/programs -l`

You can use the tab key to auto complete when typing a directory or file name. For example, type the last listing command over, but this time, only type the first two letters of a directory name and press the tab key to auto complete. Sometimes you may need to type more than two letters because the auto complete needs enough characters to distinguish from similar names. You can also use the arrow keys, up and down, to scroll through previous commands.

Creating Directories

Creating directories is done using the **mkdir** command. To create a directory within the directory you are currently in called *tutorial*, enter **mkdir** *tutorial* at the command prompt and use **ls** to see the created directory. Try this now:

- `**mkdir** tutorial`
- `**ls**`

Now create a subdirectory within *tutorial* called *temp*. You can do this by changing directories first (changing directories will be discussed next) or by listing the parent directories for the new directory. So for example, try the following. It works by having the system descended into the *tutorial* directory, which is in the current directory, and then creating the directory *temp*.

- **`mkdir`** `tutorial/temp`
- **`ls`**
- **`ls`** `tutorial`
- **`ls`** `tutorial/temp`

Changing Directories

Changing directories is done using the **cd** command. This command followed by a subdirectory name will move you to that subdirectory. For example, try:

- **`cd`** `tutorial`
- **`cd`** `temp`

You are now in the *tutorial/temp* directory. The Linux file directory is viewed as a hierarchy in which subdirectories are children of one parent directory. This "one parent has many children relationship" means we can move up to a parent directory without knowing its name. This is done by using **..** (two periods) to designate you want to move up one directory in the hierarchy. So if you are in the *tutorial/temp* directory, you can do **cd ..** to move up to the *tutorial* directory. Or to get back to the parent of tutorial, try **cd ../../** to move from *temp*, to *tutorial*, to its parent. Try this from the *temp* directory:

- **`cd`** `../../`
- **`ls`**

You can mix them together. For example, **cd** ../../tutorial from the *temp* directory will move you into the *tutorial* directory because of the first **../**, the next **../** moves you into the parent of *tutorial*, and *tutorial/* moves you into the *tutorial* directory. Needless to say, this example is trivial because you could have done it by typing **cd ../** from the *temp* directory. The idea is that you can quickly navigate the hierarchy in a single command if you know how it is organized. Some other useful ways to change directories include:

- cd ~ returns you to your home directory. You can build paths using this, e.g. cd ~/tutorial/temp
- cd - displays the full path from the root to the current directory
- cd / changes to the root directory of the drive. You can build paths from the root, e.g. cd /home/tutorial/temp

Removing Directories and Files

Removing directories is done using the command **rm**. This command actually has two functions: removing directories and removing files, but in order to remove a directory, you have add the **–r** option. This option does a recursive deletion which will remove the files and subdirectories of the directory you want to remove. So try to remove your tutorial directory using:

- **`rm –r`** `tutorial`

Note that the removal of files and directories is permanent when using the command line and there may not be a delete confirmation. There is no recycle bin like in Windows, so make sure you are not deleting something you want to keep.

Removing files is the same except the **–r** option in not necessary. Create a file by issuing this command:

- **`touch`** `temp_file`

**touch** is a utility that changes the file's timestamp; however, if the file does not exist, it will create the file with no contents. If you do the directly listing with **-l**, you should see the file *temp_file*. You can use the command **cat** *temp_file* to view the contents of the file, which should be none. Perform a directory listing, try **touch** *temp_file* again, do a directory listing, and compare the timestamps of *temp_file*.

Now remove the file:

- **`rm`** `temp_file`

Copying and Moving Files

Copying files uses the command **cp**. The two parameters needed with the command are the location and filename to copy and destination for the copy. Let's walk through an example that will help you later in this lab. First, make a directory called *Tutorial1* next, issue this command:

- **`cp -r`** `~dmk0080/public/1040/labs/zero/* Tutorial1`

The **–r** option is a recursive operation that will copy all the subdirectory along with the files selected. In this case, the wildcard character * (asterisk) is used to select all the files. Replacing * with a single file name will only copy that one file. So this command will copy everything from the public 1040 lab zero directory to your *Tutorial1* directory. Now, let's practice copying one file. Change to the *Tutorial1* directory and do a listing of the files. Copy the file *integers* to the subdirectory *programs*. Perform a listing to make sure the file was copied. Also, the wild card character can be used to specify a group of files that have similar names. For example, **ls** *.txt will list all the files in the direction that end with .txt.

Moving a file is almost the same except the command is **mv** and it will not create a copy. As an example, create a file in the *Tutorial1* directory called *move_this* by typing in the command:

- **`touch`** `move_this`

Now, move this file to the programs directory:

- **`mv`** `move_this programs/`
- **`ls`**
- **`ls`** `programs`
- **`mv`** `programs/move_this ./`
- **`ls`**
- **`mv`** `move_this programs/move_this2`
- **`ls`**
- **`ls`** `programs/`

Some notes about copying and moving:

1. Just as before, you can use the directory hierarchy with directory names and the **..** to designate paths for the copy.

2. Move and copy can be used to rename files quickly. Copy is particularly useful when programming because you can quickly create a backup version of a program before making changes. For example, this would create a copy of the file under a different name allowing you to have the version 1 backup while you edit *helloworld.c*.

   ```
   cp helloworld.c helloworld_v1.c
   ```

   And this would simply rename the file:

   ```
   mv helloworld.c helloworld_v1.c
   ```

3. Files and folders have access permissions. When you do a listing with using **ls –l**, you can see the ten places that are used to designate permissions for a file. The first place indicates whether it is a directory (d) or a file (-). The next three are used to determine the *owner* permissions which can be read (r), write (w), or execute (x). Similarly, the next three indicate permission for a *group* that has some access to it, and the last three are access rights for *others* outside of the group or owner. The example below shows a file in which the owner can read, write, or execute the file, but the group can only read or execute it while others can only execute it. The same basic principle applies to  This means that the copying and moving of files depend on permissions; for example, you may not be able to copy all the files from another directory to your own.

   $$\underset{\text{Type}}{\texttt{-}}\underbrace{\texttt{rwx}}_{\text{User}}\underbrace{\texttt{r-x}}_{\text{Group}}\underbrace{\texttt{--x}}_{\text{Other}}$$

   Do a file listing of `~dmk0080/public/1040/labs/zero/` and its subdirectories and compare it to a listing of your *Tutorial1* directory and subdirectories. Are there any files that were not copied because of the permissions?

**Part III – Redirection**

Input and output of many programs in the Linux environment use something called *standard in* and *standard out*. This allows you to enter information into a program, typically, by keyboard and to see output on the screen. Actually, you have already experience this with the commands you have been using. For example, when you issued the command **cp** or **mv**, the filenames are placed into the copy program as standard in. Redirection allows you to enter data into a program from a file but treated as standard in, or to capture output from standard out into a file. For example, let's use the Linux program **sort** to sort some integers.

Find the file *integers* in your *Tutorial1* directory. You can quickly view the contents of this file by using the command **cat** as shown below.

- `cat integers`

This is a list of unsorted integers. We need to give the **sort** program this file using standard in.

Redirection involves two symbols < and >. < is used to redirect file contents into a program as standard in while > direct contents out of standard out and into a file. Trying this with **sort**, we would issue this command to see the integers sorted (note they are in alphabetical order, not numerical):

- `sort < integers`

Now to capture the output into a file, try this:

- `sort < integers > sorted_integers`
- `cat sorted_integers`

A final note about these commands and programs: You can use the command **man** to find out more about these commands and programs as well as many other Linux utilities. For example, try **man cp** or **man mv** to find out more information about how these work. You can often receive help on the command by adding the option **--help**, e.g. **cd --help**.

**Part IV – Compiling a C program**

The GNU C compiler (**gcc**) takes C code written in text and produces an executable file. Any file created for this compiler requires the *.c* extension, e.g. *helloworld.c*, but the file can be created and edited in just about any text editor. The gcc compiler requires, at the very least, the *.c* file to compile. For example, suppose you have a file called *helloworld.c* . To compile this on the CSE machines, the command is **gcc** *helloworld.c* (assuming *helloworld.c* is in the current directory). The result is an executable file called *a.out*. In order execute this file in the current directory, you must add *./* before the file name, i.e. *./a.out*, so Linux knows you want to execute it in the current directory. Let's try an example:

1. Change to the *programs* directory in your *Tutorial1* directory.
2. Compile the program *dice.c* using the gcc compiler.
3. Run the program and roll the dice 1,000 times: *./a.out* 1000

You can use *control-c* to break the execution of a program, which is useful in the event your program executes out of control. Run the dice program again except with 1,000,000,000 rolls of the dice this time. Break the program using *control-c* while it is still running.

You can also name your executable file using the *-o* option with gcc instead of staying with the default a.out file name. For the dice.c program, run these commands:

1. **gcc** dice.c *-o* dice
2. **./dice** 1000

**Part V – Other Useful Information**

Now that you have a feel for Linux, here are some additional helpful tools and commands. Use **man** and the internet to read about them and practice.

1. Find out how to copy and paste using *PuTTY* (this is not a Linux thing, but will be useful when programming.)
2. Find out what **pwd** does.
3. Find out more about file permissions and the command **chmod** to change them. Practice this by creating some files and changing the permissions. Also, look into the command **chown**, which allows you to change the owner of a file, and the Linux **file mask**. Practice changing owners of a file with a classmate.
4. Read about and practice the commands **find** and **grep**, and how to pipe commands together using the pipe character: |.

5. There are many text editors installed on the CSE machines that will allow you to edit files while connected using *PuTTY* including **vi** (or **vim**), **pico**, and **joe**. These editors are often helpful because you can edit your program files right on the CSE machines and often color code your program syntax to make it easier to read. Because they are command line based, they often use unusual shortcut keys to perform operations, but once learned, it can be more efficient than using a mouse. For each of these, at the very least, learn how to open files, save files, delete a line, copy and paste, and how to exit.

**Part VI – Assignment**

In this section, you will need to perform the following operations. You will have to submit the files so be sure to save them.

1. Create a directory called *Lab0*
2. Copy the files from `~dmk0080/public/1040/labs/zero/` into your *Lab0* directory 3
3. Create a copy of *integers* in the same directory and name this *integers_v*1 4
4. Create a listing of your *Lab0* directory and redirect the output to the file *Lab0_listing*1
5. Create a listing of your subdirectory *programs* using redirection and name the file *Lab0_listing*2 6
6. Sort the *integers* using redirection and redirect the output to a file named *sorted*

You are not required to submit any files for this assignment as it does not count for a grade. Hopefully this has served to get you acquainted (or re-aquainted) with the CSE Linux Servers.