CS 122 – Lab #1 – Starting with Linux and Java

The labs in this course will make use of the department's Linux server, cs.furman.edu. Each of you has a personal account on the server. The first thing you need to do today is to log in to the server. The best way is to use Putty, the secure remote shell program.

As you log into cs.furman.edu, you need to know your username and password. Some of you have already done work on this server in your earlier classes. If so, then you should continue using your same username and password. However, if this is your first experience using the server, then your account is new. Your username will be your first initial and then your last name, according to the name used for your Furman e-mail address. For example, if your name is Donald Duck, then your username is dduck.

Initially, your account has a default password, which you must immediately change when you log in. To change your password, use the `passwd` command. It will prompt you for your old and new passwords. Select a new password for yourself. *You are responsible for the security of your password*, because sharing accounts can lead to academic integrity violations.

Part 1: How to use Linux

1. Now that you are on the server, it's time to practice basic Linux commands. You should see a command prompt, which means the operating system's shell is ready for your commands. You will notice that that the command prompt gives both the name of the server, your user name, and your current location in the file hierarchy. Every time you log in, you are taken to your home directory, otherwise known as ~.

2. In Linux, to tell the server to do something, you just type in the appropriate command and hit enter. On PCs, there is also a command-line interface called DOS, which is similar. So, entering Linux commands is just like using the DOS prompt, but of course the commands are a little different since Linux is a different operating system.

3. In the terminal window, type in the command `who` and then hit return. This command asks Linux who is logged into the machine. In response, you should see your username listed, along with the time you logged in. You may see other names listed too: it's possible for several people to be using the same machine at the same time. The CS server is a multitasking time-share system, which means it can easily handle many people logged in at once.

4. At this point you should carefully look over sections 1-9 in the appendix on Linux commands, to review the basic commands covering basic file management (creating, renaming, copying, etc.). However, we probably don't have any files to manipulate yet! Let's create some.

   There are basically 2 ways to create a file: either by using an editor to type out the contents we want; or by running a command/program and making sure its output goes into a file, such as by redirecting the output. Let's try both techniques.

Complete section 1 of the Linux commands appendix. It deals with an editor called `nano`, which is fairly simple to use.

If you have not done so already, create a directory called lab01. Use `cd` to enter this directory. Use `nano` to create a file called Hello.java, the classic "Hello, world" program in Java. When you are finished typing, save the file and exit `nano`.

```
public class Hello
{
  public static void main(String [] args)
  {
    System.out.println("Hello, world!");
  }
}
```

Now you are ready to compile. Type `javac Hello.java`. If you have any syntax errors, then re-edit the file Hello.java.

If there are no error messages, then the compilation was successful. Type ls. What new file was just created? To run your program, type `java Hello`. The java command runs the Java virtual machine. Note that although the name of the class file is Hello.class, you do not include the ".class" file name extension when you invoke the java command. You should see your output immediately on the screen.

Now, here is the 2nd way to create a file. Anytime you run a program that has output, you can *redirect* the output to a file. To accomplish this, you need to use the special ">" symbol and give the name of the output file you want. For example, you can type `java hello > hello.out`. This command says that we want to run "java hello", and write its output to the file called "hello.out". **Caution** – if this file already exists you will be overwriting it. Linux will not warn you about this.

In the Linux command appendix, you'll see a short section on the cal command. If you type `cal 2019` you'll see this year's calendar to the screen. And it's not hard to write this calendar to a file. You can type: `cal 2019 > this_year.txt`. Of course, just like any other file, you could use the "more" or "less" commands to look at it.

5. We'll do much more with Linux later (e.g. there is such as thing as "input redirection"), but for the time being, it would be a good idea to experiment with renaming, copying and moving files. From now on, each lab should go in its own directory, so you should become comfortable navigating from one directory to another.

Part 2: Creating Java programs

The remainder of today's lab deals with creating two new Java programs. You should test your programs with various input. When you are finished, please demonstrate the programs to the instructor or lab aide. Have fun!

1. The first program will be called Vertex.java. The purpose of this program is to find the vertex of a parabola. If $y = ax^2 + bx + c$, then the vertex is at the point $\left(-\frac{b}{2a}, C - \frac{b^2}{4a}\right)$. Your program should follow the following design:

   - The user will enter a, b and c at the command line, so that `args[0]` is a, `args[1]` is b, and `args[2]` is c. Assume that these are real numbers, not integers.
   - Create a new variable x that calculates the x-coordinate of the vertex.
   - Create a new variable y for the y-coordinate of the vertex.
   - Print out the point (x, y). Don't forget the parentheses and comma.

   For example, if we run the program as:
   ```
   java Vertex 10 9 -1
   ```

   The output should be:
   ```
   (-0.45, -3.025)
   ```

2. The second program will be called Name.java. The user will supply a person's first and last name. You may assume that `args[0]` will be the first name and `args[1]` will be the last name. The program should output this name with the last name first, followed by a comma and space, followed by the first name. For example if we run the program as:
   ```
   java Name Hazel Flagg
   ```

   Then the output should say:
   ```
   Flagg, Hazel
   ```

That's it… You're done with the first lab!

APPENDIX: Handy Reference of Linux Commands

Some of the more useful Linux commands are described here. You should keep this lab handout handy until you feel more comfortable with the commands. Just one word of caution: Linux was developed primarily for computer professionals. It was designed to get work done quickly and powerfully. Consequently, it is easy to shoot yourself in the foot if you are not careful – Linux will do exactly what you tell it to do. For example, when you tell it to delete a file, it won't prompt you for a confirmation – that file is gone forever! In just a few keystrokes it is possible to destroy all your files. So be especially careful when using special wild-card characters like the '?' and '*'. (The question-mark will match any single character in a file name, and the star will match any number of characters in a file name.)

Commands for files

1. The `nano` command

   `nano` is a text editor, which allows us to create a text file. Most often we will use an editor to write a program's source code, or some other input file. Since we will be using Linux as a programming environment, it is important to become comfortable with a text editor. Another text editor on this system is vi, but it is more difficult to learn.

   To edit a file, simply type in `nano` at the command prompt, followed by the name of the file you want to edit. For example, type in `nano test1`. Since this file does not already exist, you will see it empty inside `nano`. At this point, type in the following sentence:

   ```
   This is my first text file created with nano.
   ```

   Also put in your name and date, and anything else you wish. The arrow keys allow you to navigate through the file, and the backspace and delete keys do what you would expect. Notice that along the bottom of the window, there is a list of control commands. For example, to save your file, hit control-O and verify the file name. You also have the opportunity to save the current file under a new name as well. Then, to quit from `nano`, hit control-X. If you enter control-X without saving, you will be warned and given a chance to save the file before you leave.

   If you happen to have a bigger file, using the arrow keys to go up and down can be quite slow, so to move up and down faster, there are control keys for this too: control-V goes down and control-Y goes up. (We consider the "top" of the file to be the beginning.)

   Note that it is possible to start `nano` without specifying a file name.

2. The `ls` command

   This is like the `dir` command in DOS. The ls command is saying, "Please list the files that are in this directory". Various options can give you additional information.

At the command prompt, type `ls` . (The period is not part of the command!) Not much of a list, is it? Now try `ls -a` . When we add that option `-a`, this stands for "all" files, including those whose names begin with a dot. The dot-files are special files that deal with the particular characteristics of your computer account.

Now type ls `-l` . Here, the `-l` option stands for "long listing". (That option is the lowercase letter L not the number one.) This provides more detailed information about each file, including its size and when it was last modified.

If you type `ls -la` , this combines both options: print all files, and give a long listing.

Now type `ls -lat` . The extra option t means to list the files in chronological order according to the date and time of their last modification.

Now type `ls -latr` . The option r means to list in reverse (chronological) order.

3. The `pwd` and `mkdir` commands

`pwd` stands for "present working directory". If you type `pwd`, the system will tell you in which directory you are currently working. Since you have just recently logged in, you should still be in your home directory.

If you want to create a new directory (for example, a subdirectory within your home directory), we use the `mkdir` "make directory" command. So to create a directory called demo, simply type `mkdir demo`. Note that when a directory is created, it is initially empty: there are no files inside. (You can create as many directories as you wish to hold files, and directories can themselves contain directories: this is called a hierarchical file system.)

Now type `ls` to see that a new directory is in fact listed.

4. The `cd` command

cd stands for "change directory". This is like the `cd` command from DOS, except that the convention in Linux is to separate directory names with a forward slash instead of a backslash. With this command, you can navigate the directory hierarchy, or give a specific destination. Here are some examples:

Command	*Meaning*
`cd`	changes to your home directory no matter where you are
`cd ~`	does the same thing as just typing "cd"
`cd demo`	changes to the current directory's subdirectory demo
`cd ~/demo`	no matter where you are, go to the demo directory that is located right under your account's main level

`cd ..`	goes back to the next higher directory in the hierarchy
`cd /usr/share/dict`	changes to the specific directory /usr/share/dict

5. The `cp`, `mv` and `rm` commands

   `cp` stands for "copy", `mv` stands for "move" and `rm` stands for "remove".

   We use the `cp` command in order to create another copy of the same file into the same or a different directory. First, type `cd` to go to your home directory. Let's try some examples of copying files:

`cp test1 first_file`	creates a copy of the file with a new name
`cp first_file demo/.`	creates a copy of this file and puts it in the demo directory Note: the "." means to keep the same name as the original.

   Now use the `cd` command to go into the demo directory, and then type `ls .` Do you see the first_file there?

   Now that we are in the demo directory, let's copy the other text file `test2` here as well. Type `cp ../test2 second_file.` This literally says "copy the file test2 from the parent directory and put it in here under the name second_file".

   (Please call me over any time you don't understand what's going on.)

   The `mv` (move) command is used either to rename a file or to move it into another directory. The big difference between `cp` and `mv` is that `mv` is not making a copy.

   Make sure you are in the test1 directory. Try these examples:

`mv first_file 1st_file`	change the name of this file
`mv second_file ..`	move this file back up to the home directory

   Now type ls to see the changes. Do you understand what happened? Go to the home directory (either by typing `cd` or `cd ..`) and type `ls` there. You should notice that second_file has just been moved here.

The `rm` (remove) command deletes the file(s) you indicate. Once deleted, a file cannot be restored, so be very careful! Try this sequence of commands inside the demo directory:

`ls`	list the contents of the directory
`cp 1st_file new_file`	create a copy of the file
`ls`	list the contents of the directory
`rm new_file`	remove the file
`ls`	list the contents of the directory

6. A little help from the shell: filename completion and history.

   This is not really a command, but I wanted to share with you a couple of feature of the operating system shell. (Note: the "shell" is the part of the OS that we directly communicate with at the terminal.) Anytime you are typing some command, and you need to enter the name of some file, directory or command, it is normally not necessary to type every character. Often, all you need to do is type the first few letters, and Linux can figure out the rest. This happens when those first few letters already uniquely identify what you are talking about. For example, if you have a directory called homework, and it is the only thing that starts with the letter h, then typing the characters "cd h" and then hitting tab will complete the rest of the file name.

   If you ever need to repeat a command you recently issued, you can use the up and down arrow keys to navigate your history of recent commands. If you actually want to see a list of the commands you have typed, you can simply type "history" and hit enter.

7. The `scp` and `ssh` commands

   These are useful commands for communicating with another machine. `scp` means "secure copy" and `ssh` means "secure shell."

   `scp` is just a generalization of `cp`, and the syntax is similar. The idea is that you have an account on another machine, and somewhere in your directory structure over there, you have a file you wish to retrieve. A typical command may look like this:

   ```
   scp mmouse@server.disneyworld.edu:menu/dinner.txt .
   ```

   The @ sign separates the name of the user from the hostname of the machine. Following the : we have the name of the file, which in this case lies inside some directory. That final '.' means to copy the file to the current working directory on this machine, just like with the `cp` command. As soon as you enter your `scp` command, you may be prompted for a password before the file transfer can take place.

   The `ssh` command allows you to remotely log in to another machine.

8. The `more` and `less` commands

   This is one of the most commonly used commands. With the `more` command, you can view the contents of one or more text files, one screenful at a time. To proceed through a file, press the *space bar* to continue to the next screenful, or the *return* key to advance just a single line. You can exit from more when you reach the end of the file, or by hitting **q** at any time. Typing **h** gives a concise help menu for using more.

   If you are looking at a small file that can fit completely on the screen, then more will immediately exit, although you can still see the file. Experiment with these:

`more /etc/motd`	Show me the message of the day.
`more /usr/share/dict/words`	Show me the spell checker's dictionary.

   Believe it or not, `less` is similar to `more`, but allows a little more flexibility in navigating a file. For example, when you reach the end of a file, less does not automatically quit. Within less, you can use instant single-letter commands such as "g" to go to the beginning of a file, or "G" to go to the end. The space bar goes to the next page, and the "u" command goes up. To search for text within the file, preface your search with the slash key. To exit, hit "q".

   If you simply want to dump the contents of a text file to the screen without stopping page by page, you would use the cat command.

9. Compilers

   `gcc` is the C compiler. It is customary for a C program to have a file-name extension of lowercase c. One note about using C compilers – by default, the executable file is called `a.out`, which is sometimes not the name you want. You can specify your own name for the executable by using the `–o` option, like this: `gcc –o hello hello.c`. In Linux it's customary for executable filenames not to have an extension.

   Warning! Be careful when using the `–o` option. The file name that immediately follows –o will become the executable file. You will not be warned if this file already exists. In particular, the following command would be a fatal mistake: `gcc hello.c –o hello.c`. This will have the effect of compiling your program and overwriting your source code with the executable. Your source code would be destroyed! In short, when typing –o, be sure that what follows is simply "`hello`" and not "`hello.c`".

   On the CS department Linux servers, it turns out that other compilers exist such as `g++` for C++, and `javac` for Java. Similarly, the Python interpreter is `python`.

Commands for general information

1. The `date` command

   This command prints the current date and time.

2. The `cal` command

   If you type `cal`, it will give you the calendar for the current month. If you want the calendar for some other month, then you need also to give the desired month and year, such as `cal 9 2001`. If you want to see the calendar for an entire year, you only need to give the year number, as in `cal 2002`.

   Try the calendar for September 1752. Notice anything funny about this?

3. The `man` command

   `man` stands for "manual". This may become one of your best friends in the Linux/UNIX world. If you ever come across some new command you don't understand or just want to learn more about it, the man command can tell you what it does. This command works like the less command. When you are finished reading the documentation, hit "q" to exit. Try these examples:

`man cal`	tell me what the cal command does
`man mv`	tell me what the mv command does
`man man`	tell me what the man command does ☺

   Also, if you want to look for a command, but you don't know what it's called, use the `-k` option to initiate a "keyword search".

`man -k file`	list commands having to do with files
`man -k compiler`	list commands having to do with a compiler

A few miscellaneous commands

You should probably just skim this section for now. Some of these commands are a bit advanced or not available on the machines we are currently working on. Some commands will only make sense if you are on a multi-user system. And some commands are simply not needed until you have done more work in the Linux environment. For example, you don't need to worry about running out of disk space until you start generating some big output files, which probably won't happen in this class.

1. The `look` command

   When writing some (English) document, sometimes you need to know how to spell a word, and you don't have time to get to a dictionary. With the look command, give just the first few letters and the system will tell you all the words it knows that begin with those letters. The word list is somewhat abridged, so many English words are not included, but it does contain about 250,000 words. It does include many proper names and some abbreviations. Try look aqua to see what words begin with these letters.

2. The `diff` command

   If you have two files that are nearly the same, diff can tell you on which lines they differ. The response you will get from diff is the line numbers corresponding to the differing lines followed by what those lines look like. If `diff` doesn't say anything, then the two files are identical.

   For example, `diff one.c two.c` might give output like this:

```
23c23
<   int floor, dest, duration;
---
>   int floor = 1, dest, duration;
41c41
<       duration = SPEED;
---
>       duration = SPEED * (dest - floor);
```

   The output shows that the two files differ at lines 23 and 41. The "<" at the beginning of the line refers to the first file you specified, while the lines beginning with ">" refer to lines within the second file.

   As you can see, diff is useful when you are working with two versions of the same program, and you want to see what has changed.

   Here's another example output for a command like `diff one.C two.C`:

```
39a40,41
>       printf("going up\n\n");
>       duration = SPEED * (dest - floor);
```

   This output indicates that the second file two.C contains two new lines (40 and 41) that are not present in the first file one.C.

   If you are comparing two files that are very large and you know that there aren't many line-by-line differences, then you can use the `-h` option to make diff go much faster.

   If it turns out that there are large differences between the two files, it may be helpful to slow down the output by sending the output through more, as in:

```
diff one.c two.c | more
```

The vertical bar (|) tells the system to send the output of diff through more. This is a useful trick in general, anytime you have a lot of output coming onto the screen.

3. The `cat`, `head` and `tail` commands

   `cat` dumps a file to the screen, which is not helpful if the file is large. More commonly cat is used for concatenating several files into one. If you still have the files test1 and test2 in your home directory, you can try concatenating them. Try this:

   `cat test1 test2 > combined`. This command creates a new file called `combined`. Take a look at this file.

   Note the '>' sign when concatenating files. This symbol is used to separate the list of files to be concatenated from the name of the new file.

   By the way, here is a general rule in Linux – any time you ask the system to create a new file for you, be sure it does not already exist, or else it will be automatically overwritten!

   `head` displays the first 10 lines of a file. Similarly, tail shows the last 10 lines of a file. You can specify a different number of lines as an option like this:

   `head -25 time.c` show me the first 25 lines of time.c

   `tail -40 prog2.c` show me the last 40 lines of prog2.c

4. The `wc` command

   wc stands for "word count". It can tell you how many lines, words and characters are contained in the files you specify. Suppose we have a file called "prog2.c", and we type wc prog2.c . The output would look something like this:

```
    162     411     2845 prog2.c
```

5. The `grep` command

   This command is often used to look for a word, phrase or some other text pattern in a file. The output from this command consists of lines from the file that contain what you are looking for. Note that if you are searching for a phrase, or a "word" containing special symbols, you should enclose it in quotes before you give the file name. Some examples:

   `grep file wordgame.c` displays lines from wordgame.c containing "file"
   `grep Eight /etc/motd` displays lines of motd containing "Eight"

Actually, it would be even more useful if you could find out which line numbers are being printed, especially if it you are looking in some big file. To do this, pass the option –n, as in grep -n file unix . This will print all the lines from the file unix containing the word "file", but at the beginning of each line will be its line number. For example, the output may look like this:

```
105:   name of a file
220:   file name.
319:      does not get sent, but appears in your home directory as a
file
337:   name of the file you wish to create/edit.
371:   executable file will be called "a.out".  You can specify a
different name
```

6. The `du` and `df` commands

   The `du` command can tell you how you disk space is distributed among all of your directories or any directory you specify. Each line of output tells how much space (measured in blocks of 1024 bytes = 1KB) is taken up by that particular directory including all of its subdirectories.

   The `df` command gives you an overview of the entire file system. It shows how many kilobytes are in use and available on the memory device(s).

7. The `chmod` command

   If you are ever in a multi-user system, this command may be useful. It will give you some control over the security of your files. Depending on how your account is set up, by default, when you create a new file or directory, it may be readable by anyone who knows where your file is. The command `chmod` stands for "change mode". To protect your files from eavesdropping, here is what you do:

   If you only want to turn off the permissions on a single file, enter this command:

   ```
   chmod 600 <filename>
   ```

   The number 600 means that you have read and write privileges, and everyone else has no access. If you are trying to protect a directory or executable, use the number 700 instead in order to preserve execute access for yourself.

   To protect an entire directory, here is how to do it:

   Go up one directory level by typing the command "cd ..". Then enter the command:

   ```
   chmod 700 <name of directory>
   ```

8. The `ps` command

   This command will tell you what commands you have running on the machine. Each process will be printed one per line. At the beginning of the line is the process number, and at the end of the line is the name of the command. To get a list of all the processes running on the machine at this instant, use `ps -ef`. Similarly, to retrieve a list of all processes running by you, then you can use `grep` to filter the entire list. For example, if your username is dduck, you would say:
   ```
   ps –ef | grep dduck
   ```

9. The `kill` command

   In case you need to terminate a (runaway) process, try ctrl-C. If this doesn't help, you need to kill the process. That's right: in Linux you have the power to kill (a running program, that is). Look up the process number with the `ps` command, and then type kill followed by the process number. For example, let's say that `ps` shows you:

   ```
    PID TTY      TIME CMD
   6641 pts/1    0:01 ksh
   6735 pts/1    0:00 sleep
   ```

   In this case, to terminate the sleep process, we type `kill 6735`.