CIS-350
INFRASTRUCTURE TECHNOLOGIES
Unix/Linux – Advanced Command Line Interface (CLI)
LAB #4

This Lab is worth 50 points. The questions in the Lab 4 Report that you need to answer are worth 25 points. You **must** log in to Ubuntu Linux and actually work the lab commands on the Ubuntu Linux system. The hands-on work is worth remaining 25 points.

Due date: **See Blackboard**

Objectives: Learn more about **Linux/Unix commands**

You will need:

1. The Ubuntu Linux account and VMWare Horizon client.
2. PuTTY and PSFTP
3. Link to the *vi* commands such as https://www.tutorialspoint.com/unix/unix-vi-editor.htm

In the previous lab you were experimenting with some Linux/Unix commands like *passwd*, *who*, *date*; file manipulation and directory commands such as: *ls*, *pwd*, *mkdir*, *cd /*, cd, *cat*, *more*, cp, *ln*; redirections (>, <); piping (|); and finally with the *vi*, *nano*, and *pico* editor commands.

In this lab, you will enhance your knowledge of Linux/Unix. You will learn more about (1) the *vi*, *nano*, and *pico* editors; (2) compiling, link-editing, and running C programs; (3) opening and closing the script files; (4) *more, stty, chmod*, and *chgrp* commands, (4) checking the status of processes (*ps*), (5) redirection, piping, and sort utility (*sort*), (6) running jobs in background (*bg*) and foreground (*fg*), and (7) *sleep* utility.

The commands to type in are in bold and italic.

1. Log in to Ubuntu Linux as described in the previous lab on Linux.

After you login you should see *youruserid@mercury:~$* prompt indicating that you are in the default Bourne Again (bash) shell. For example, *jmzura01@mercury:~$*.

To delete the entire command you are entering from the prompt, hit *Ctrl-U*. Note that file names and commands are in lower-case. Unix/Linux is picky about case.

To obtain **help** on any commands, you would type *man command_name* at any time. For example, type

man sort and press Enter

If you do not want to browse through the entire sort command, type *Ctrl-Z* to quit or hit *q*.

2. Storing the complete interaction with the Linux/Unix system

Type:

script Lab4 and press *Enter*

This will start the script and all the input and output from the terminal will be written to file *Lab4*. (If you had pressed Ctrl-D keys now it would have terminated the script session. Do not hit Ctrl-D yet. You will do it at the end of this lab.)

3. The terminal control-key settings and **profile files**.

Type:

stty -a and press *Enter*

to see your terminal control-key settings and other information. You should see something like this:

intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; start = ^Q; stop = ^S; susp= ^Z;

Control key	ssty Name	Function Description
^C	intr (interrupt)	Stop current command
^*D*	eof	End of input
^\	quit	Exit Unix/Linux
^S	stop	Halt output to screen
^Q	start	Restart output to screen
^*U*	kill	Erase entire command line
^*Z*	susp	Suspends execution of command

(The ^*C* notation stands for *Ctrl-C*, hold down the *Control* key and press the *C* key). The control keys you will probably use most often is ^*C* (sometimes called the *interrupt* key) and ^Z (*suspend*). The ^*C* stops (interrupts) the command/process that is currently running without possibility of resuming it, whereas ^*Z* suspends the command/process with the possibility of resuming it. The suspend command (^*Z*) is used pretty often in multitasking. It allows you to suspend temporarily any process (preferably the one that consumes a great deal of the CPU time) that executes in foreground and move it to background using the *bg* command. This way you can use your Linux machine to perform other urgent activities in foreground. You can move a command/process running in background to foreground with the *fg* command.

To erase the entire command you type, use ^*U*, which stands for *kill*. (The ^ stands for the CTRL key.) Type some erroneous command (garbage) such as:

kkkkkkkkkkk and hit ^*U* (hold down CTRL key and press U)

It will cancel the garbage that you have just typed. You can obviously type a legitimate Linux command such as *ls* and hit ^*U* to cancel it.

It is beyond the scope of this lab to discuss in detail all the control key settings. However, you might want to know that to change the control settings, you might use the *stty* command in the following manner.

For example, if you want ^*G*, instead of ^*U*, to erase the entire command you type, you can redefine *kill* by typing:

stty kill ^G

Now type

stty -a

to verify that *kill* definition has been changed. You should not see ^*U* any longer for *kill*. You will see ^*G* instead. Now, let's type an erroneous command and let's try to erase it with ^*U*. It would not work. However, ^*G* will do the job. Type on purpose this erroneous command:

kkkkkkkkkk hit ^*U* and *Enter*. You should get an error message.

Type again:

kkkkkkkkkkk and hit ^*G*.

It will cancel what you have just typed and on the same line you can just type the correct command, for example, type

ls

From now on, you will be using ^*G* to erase/kill the entire command line. However, this definition/assignment will be in effect for the current session only.

For the ^*G* command to take permanent effect, you would have to change the profile file which is executed each time you log in to the system or each time you switch to a different shell. Simply, the command *stty kill ^G* would have to be added to the profile. The profile file contains commands which help you customize your Linux/Unix account. To be on the safe side, however, in this lab we will not be experimenting with this and we will not be altering the profile file. If you screw up your profile, you may not be able to log in to the system and will need to seek help from the system administrator. The profile file is an invisible file as its name starts with the period. However, in Linux/Unix you can display them by using the *ls –al* or *ls –a* commands and/or modify their contents using *nano, pico*, or *vi* editors. The profile file is very similar to the Windows *autoexec.bat* file and Windows registry system which are also automatically executed when Windows is booted.

If you type

ls -al .profile and hit *Enter*.

You should see the following write permissions for the owner, users in the owner's group and all other users.

- rw- r-- r--

The owner of the account has the write authority to file .profile. The first hyphen – stands for the ordinary file.

Sometimes it may be wise to take your write permission away to the profile file to protect it against accidental changing. You can do it by typing *chmod u-w .profile* for the bash shell.

In the profile file you may include many other commands to help you further customize your Linux environment. For example, you might include the *umask* command that allows you to set the default file permissions that will be assigned to all files that you create. The argument of a *umask* command is a bit mask, specified in octal, which identifies the protection bits that are to be turned off when a new file is created. The access permission value for a file is computed by the expression:

file access permission = default permissions - mask

where 'mask' is the argument of the *umask* command and 'default permissions' are *777* for an executable file and *666* for a text file.

Without going into the complexities of this command, let it suffice to say the a value of *022* (octal) in the *umask 022* command applied to text files will produce the permissions (*666-022=644*), i.e., *rw- r-- r--* (read-write by owner, but read-only be everyone else). A value of *002* will produce (*666-002=664*), i.e., *rw- rw- r--* (read-write by owner and group, but read-only by everyone else). The file *.profile* is in your home directory.

To check it, type

ls -al .profile

You may want to see its content by typing:

cat .profile or ***more .profile***

You will learn more about profile files when you use Linux/Unix more often in the future.

4. Compilers and linkage editors

Compilers and linkage editors are part of the OS as well. In this short exercise you will learn how to use the *nano, pico*, or *vi* editor to key in and edit a C program. Then you will use the Unix/Linux commands to compile, link-edit, and run the program. (It is not my intention to make you program in C language, though after taking or while taking the C# class it might be quite easy.) The program below generates 2,000,000 random integer numbers and writes them to a file named *random.dat*. Using the *nano, pico,* or *vi* editor, carefully type the program in and name it *prog1.c*.

This program has to compile and run successfully because it produces file *random.dat* that you will need in the commands later in this lab.

To open one of the editors, type:

nano prog1.c or ***vi prog1.c*** or ***pico prog1.c***

The editing and saving commands for the *nano* or *pico* editor should be self-explanatory from the menu displayed on the screen. A link to the *vi* editor commands is posted in the Lab 3 folder.

```
/* An example C program */
#include  <stdio.h>
#include  <stdlib.h>
#include  <math.h>
int main()  {
      int  MAX_SIZE = 2000000;
      int  x[MAX_SIZE],  i;
      for  (i=0;  i<MAX_SIZE;  i++)
            x[i] = rand();
      {
            FILE*  ptr;
            ptr = fopen ("random.dat", "w");
            for (i=0;  i<MAX_SIZE;  i++)
                  fprintf (ptr,  "\n%d", x[i]);
            fclose (ptr);
      }
return 0;
}
```

To display the program and verify if your program displayed is the same as the one above, type:

cat prog1.c

If you notice any syntax errors, invoke *nano*, *pico*, or *vi* editor again, correct the errors, and save the corrected version.

To compile the program, type:

cc prog1.c or ***gcc prog1.c***

If, after several attempts, your program still does not compile, copy *prog1.c* from the Linux *tmp* directory to your home/login directory by performing the steps below. I placed programs *prog1.c* and *designmenu* (will need in Lab 5) in the Linux *tmp* directory. If you do not see two programs in the *tmp* directory, please send me e-mail.

cd / and press	<Enter>	- to bring you to the root directory
cd tmp	<Enter>	- to move you to *tmp* directory
ls -al	<Enter>	- to check that file *prog1.c* is there
cp prog1.c /home/youruserid	<Enter>	- to copy *prog1.c* to your login directory
cd	<Enter>	- to bring you to your login directory
ls –al prog1.c	<Enter>	- to check that *prog1.c* is there

To compile and link edit the program, type:

cc prog1.c or ***gcc prog1.c***

This program has to compile and run correctly because it produces the *random.dat* file that you will need in the commands like *sort* later in this lab.

If compilation of program *prog1.c* is successful, the default executable file named *a.out* is created. The *a.out* is the default name given by Linux to the executable file.

You can display the attributes of this file by typing:

ls –al a.out

To run the C program, type:

./a.out

If you would like to have a user-defined name, such as *prog1*, of the executable file, you would have to compile the C program with the command:

cc –o prog1 prog1.c or ***gcc –o prog1 prog1.c***

In the above command, an option *o* followed by a user-defined filename *prog1* allows one to name the object file. The *prog1* object file would be created when compilation is successful.

Then to run the C program you would type:

./prog1

If the program runs correctly and a sufficient amount of disk space was allocated to your account when it was created, the output will be routed to a *random.dat* file. The file will contain 2,000,000 random numbers. Note that the program does not route any output to the computer screen. The reason for generating that many random numbers is to demonstrate the difference between running the *sort* utility in foreground and background. This is shown later on pages 6-7.

Display the directory using *ls -al* command and verify that all the above files are there. Using the command *more random.dat* we will soon be displaying the contents of the file. The command *cat* would be good to display small files. If you had just used the *cat* command, without any options, you would have noticed that the content of the file rolls up faster that you can read. To stop and quit, you would have to use ^C or ^Z. (The contents of the file would scroll off anyway for some time). For large files, the command *more random.dat* , *cat random.dat | more* or *cat random.dat | more* will work better.

To display the file content one screen at a time, type

more random.dat

The *more* command may feel pretty useless if, after the first screen is displayed, you keep pressing the Enter key to display next page. With the Enter key the screen just goes down line by line. If you want the screen to go down page by page, press the space bar.

Note that the numbers generated by the C function *rand()* in *prog1.c* are truly random. The prompt at the bottom says that you are 0% of your way through the file. (It's a very large file). You can enter *h* (for help) at the *more* prompt to display the *more* command navigation commands available on the system. You should see the table with the key definitions for *more*. Part of this table is replicated below. Commands flagged with an asterisk ("*") may be preceded by a number.

<space>	Display next k lines of text [current screen size]
Z	Display next k lines of text [current screen size]*
<return>	Display next k lines of text [1]*
d or ctrl-D	Scroll k lines [current scroll size, initially 11]*
q or Q or <interrupt>	Exit from more

To quit *more*, type

q

5. Protecting and sharing files.

Type the command

ls -al

The command displays all (*a)* files (including invisible files) in your directory in a long (*l*) form. To be able to correctly interpret the directory listing, refer to the previous lab. The following items appear in the following horizontal order:

(1) *type* of the file displayed as (d or -)
(2) *access modes*
(3) *links* (the # of files and directories linked to the file)
(4) *owner*
(5) *group* that owns the file
(6) *size* in bytes
(7) *date* and *time* when a file was last modified, and
(8) *name* of the file and *extension.*

Traverse through different directories using *cd ..*, *cd /*, *cd*, *pwd*, and *cd dir_name/dir_name* command, where *dir_name* should be replaced by the name of your actual directory. Which directory are you in after each command?

Type:

ls -al random.dat

Examine the permission for the *random.dat* file. It should be *rw- rw- r--*.

You as a file owner have a read (*r*) and write (*w*) permission only. Access permissions for users in the owner's (your) group are *rw-* and for all other users are *r--*. Assume that the file *random.dat* is of a great importance to you, and you and users in the owner's (your) group want to be able to read this file only. You have to deny yourself and to the group the write (*w*) permission. To do this type:

chmod ug-w random.dat

The same could be achieved by using an "octal" version of the *chmod* command. You may type *man chmod* for the details.

To see if this change has been implemented, type

ls -al random.dat

You should see: *- r-- r-- r--* …… and other attributes of the file

Then type

nano random.dat

and try to modify the file and save it. The system should not allow you to alter the file contents. You have to exit the file without saving it.

You may want to add read access permission to one of the students in our class by using a command *chgrp* or *chown.* Use *man chgrp* for help.

6. The sort utility, multitasking, input/output redirection, foreground, and background.

Let's say we want to numerically (option *n*) sort the file *random.dat* in the descending order (option *r*) and save the output in the file *randsor.dat*. To do this, type the following command:

sort -nr random.dat > randsor.dat

Note the use of the input and output redirection. Also, note that since we have not used the ampersand (&) following the command, the command (process) runs in foreground. The file *random.dat* is large (2,000,000 random numbers), however, it will take only several seconds to sort it. To be able to execute other commands, you have to wait for about 10 seconds until the prompt reappears. You will see the prompt again when the *sort* command is done.

If you had assigned 5,000,000 to *MAX_SIZE* in the instruction *int MAX_SIZE = 5000000* in the C program on page 4 and your Unix/Linux account disk space would have been large enough, the program would have generated 5,000,000 random numbers. It would obviously have taken longer (say, 60 seconds or more of the CPU time) to sort the numbers in the file *random.dat* and send them to file *randsor.dat*. You could not execute any other commands during that time. As you would have wanted your terminal to be available for other activities, you would have to permanently kill the process *sort -nr random.dat > randsor.dat* by hitting *Ctrl-C*. The *~$* prompt would reappear immediately.

You will do two things now: (1) start *sort* in background (&) right away or (2) start *sort* in foreground and then move to background. Let's try out the 1st approach.

Type

sort -nr random.dat > randsor.dat &

Linux will display: the job id in the brackets [ ] and the 3-6 digit random process number for this task, for example, [1] and 28580, respectively, and will submit the task for execution in background.

Type immediately

ps

to list all the processes running on your terminal. Note that a *bash* shell itself is the process as well, so do not be surprised when you see the *bash* shell process running. There should be at least three processes running in background including the *sort* process and perhaps other Korn or C shell processes, if you invoked them earlier by *ksh* or *csh* commands, respectively. The system will inform you when the *sort* process is over by sending the following message to the screen.

[1]+ Done sort -nr random.dat > randsor.dat

Finished processes are erased (killed) automatically. You will only see the *sort* process running in background if you issue the *ps* command fast enough. Again, after about 10 seconds the *sort* process will be done and killed.

When the sort is done, examine the contents of the file *randsor.dat* using the command

more randsor.dat

You will see that the numbers are sorted in the descending order.

To quit, type

q or hit ***Ctrl-Z*** (hold down CTRL key and press Z)

Now let's try the 2nd approach. Let's start the process in foreground and then move it to background, so you can run do some other task in foreground.

Type:

sort -nr random.dat > randsor.dat

The file *randsor.dat* already exists and it will be overwritten. To suspend this process, hit immediately

Ctrl-Z

The *$* prompt should appear immediately. Now, type

bg

Now you can type *ps* again to check the status of processes. You can see the sort process running in background.

You can recall a job executing in background to foreground by using an *fg jobid* command. Both *bg* and *fg* commands, used without a job id number, refer to the most recent job. To kill a selected process use the *kill* command followed by the process id (PID) number. For example, *kill 18987*.

Note that it will take up to 10 seconds only to sort the file *random.dat*. Thus, it does not make much difference if you sort this file in foreground or background because the prompt will reappear after about 10 seconds anyway. To see the difference between running the *sort* commands (*sort -nr random.dat > randsor.dat* and *sort -nr random.dat > randsor.dat &*) in foreground and background, the data file to sort would have to be larger and contain at least 5,000,000 numbers or even more. We will revisit foreground and background at the end this lab using the *sleep* utility.

To release space on your disk, we will soon be erasing the files: *random.dat* and *randsor.dat*.

7. The alias and banner commands

The last thing we will try is to use commands *alias* and banner.

To check the current list of aliases, type:

alias

Say, that you want to remove the two files created above: *random.dat* and *randsor.dat*. If we consider the command *rm* is awkward, we can substitute for *rm* an easier term such as *erase* to remember. Type

alias erase=rm

Do not put blank spaces on either side of an equal sign.

Now *rm* means *erase* but this change is valid only for the current session or until you undo it by typing *alias rm=erase*. You could also permanently customize this command by including it in your *.profile* file in your home directory. Type:

erase random.dat randsor.dat

Since *random.dat* has the read status only, the system will ask, before erasing the file, whether you want, or not, to erase it. If you reply "y" (without quotes), the file will be erased. You may also add the write permission to the file *random.dat* using the command *chmod u+w random.dat* before you issue the above *erase* command.

Now, try a *banner* command, and type:

banner your_first_name

8. Foreground and background

Let's revisit this important topic again. Foreground allows one to work on one job/task at a time. Background allows one to work on several jobs/tasks at a time. This means that you can submit several tasks for execution in background. Let's outline the difference between executing tasks in foreground and background again and moving a task from foreground to background and from background to foreground using the *sleep* utility. The *sleep* command is one of the ways to affect scheduling. The command does nothing for a specified time. For example, you can have a shell script suspend operation for a period by using *sleep*. The command *sleep time_ in_seconds* will put the bash shell to sleep for *time_ in_ seconds*. You can use *sleep* to control when a command is run, and to repeat a command at regular intervals. This will be done in Lab 5.

The following steps will start the *sleep* command in background and move it to foreground.

To execute the *sleep 40* command in background, type:

sleep 40 &

Linux will assign and display on the screen the job id number and random 3-6 digit process number. The job/task id is likely to be [1]. The prompt reappears.

To see *sleep 40 &* as task [1] (likely) running in background, type:

jobs

To move sleep 40 & from background to foreground, type:

fg 1

and wait until the task is done. You will see the *~$* prompt when it is done.

The following steps will start the sleep command in foreground and move it to background.

Type:

sleep 40

This puts the bash shell to sleep for 40 seconds and when it wakes up the prompt will reappear. You cannot execute any commands in the meantime. However, we do not to wait 40 seconds.

To suspend, hit:

Ctrl-Z

The prompt will reappear.

To move the command to background, type:

bg

To see this task running in background, type:

ps

9. Saving the log of your lab session

To save the script in a file named *Lab4*, type *^D* at the beginning of the command line. It will terminate the script session. Look up the contents of the *Lab4* file using one of the editors and verify whether it contains the log of your entire Unix/Linux session with *Lab4* activities.

10. Type *exit* to log out from Linux.

Submit Lab 4 Report. (See the Assignments/Labs folder on Blackboard)