CIS-350
INFRASTRUCTURE TECHNOLOGIES

LAB #4

Due: **See Blackboard**

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

You will need:

1. The CRC and VPN accounts.
2. Link to the *vi* commands posted in the Lab #3 folder.

In the previous lab you were experimenting with some Unix/Linux 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* and *nano* editors commands.

In this lab, you will enhance your knowledge of Unix/Linux. You will learn more about (1) the *vi* and *nano* 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, sort utility (*sort*), and (6) running jobs in background (*bg*) and foreground (*fg*).

1. Log in to CRC RedHat Linux as described in Lab 3.

After you login you should see the *$* prompt indicating that you are in the default bash shell.

To delete the entire command you are entering from a ***$***, hit <u>*CTRL-U*</u>. Note that file names and commands are in lower-case. Unix/Linux is picky about case.

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

man sort

If you do not want to browse through the entire sort command, type *CTRL-C* to quit.

2. To store the complete interaction with the Unix/Linux 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 this file. (If you had pressed CTRL-D keys now it would have terminated the script session. Do <u>not</u> hit CTRL-D yet. You will do it at the end of this lab.)

3. Now let's **learn about 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 *CTRL* key and press the *C* key). The control keys you will probably use most often is *^C* (sometimes called the *interrupt* key) and *susp* which is defined as ^Z. The *^C* stops (interrupts) the command/process that is currently running without possibility of resuming it, whereas *^Z* suspends the commands 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 the foreground and move it using the *bg* command to the background. This way you can use your machine to perform other urgent activities in the foreground.

It is beyond the scope of this lab to discuss in detail all these 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:

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

Type again:

kkkkkkkkkkkk and hit *^G*.

It will cancel what you have just typed and on the next 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 Unix/Linux account. To be on the safe side, however, in this lab we will not be experimenting with and 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 Unix/Linux you can display them by using the *ls –al* or *ls –a* commands and/or modify their contents using *nano* or *vi* editors . (These files are very similar to the DOS/Windows *autoexec.bat* file (or registry system) which is also automatically executed when DOS/Windows is booted.) 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 .bash_profile* for the bash shell.

In the profile file you may include many other commands to help you further customize your Unix/Linux environment. For example, you might include there 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 *.bash_profile* is in your home directory. To check it, type *ls -al .bash_profile*. You may want to see their contents by typing *cat .bash_profile* or *more .bash_profile*.

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

4. Compilers and linkage editors are part of the OS as well. In this short exercise you will learn how to use the *nano* editor 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 here to program in C language.) The program below generates 1000 random integer numbers and writes them to a file named *random.dat*. Using the *nano* editor or *vi* editor, carefully type the program in and name it *prog1.c*. The command to do it is

nano prog1.c or *vi prog1.c*

The editing and saving commands for the *nano* 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>
#define  MAX_SIZE  1000
main()  {
        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);
        }
}
```

Use *cat prog1.c* command to display the program and verify if your program displayed is exactly the same as the one above. If you notice any syntax errors, invoke *nano* or *vi* editor, correct the errors, and save the corrected version. To compile and link edit the program, type:

```
cc  prog1.c
```

If, after several attempts, your program still does not compile, copy the exact commands from the program above, paste it to the *nano* editor, save, and recompile it. The program should run now. This program has to run correctly because it produces the "random.dat" file that will be used in the commands used later in this lab.

If compilation is successful, the default executable file named *a.out* is created.

To run the program, type:

```
./a.out
```

If the program runs correctly, the output will be routed to the file *random.dat*. The file will contain 1000 random numbers.

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 contents 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* or *cat random.dat | more* will work better.

To display 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.

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

Commands of the form "^X" are control characters, i.e. control-X.	
h	Display this help.
f, ^F, SPACE *	Forward N lines, default one screen.
b, ^B *	Backward N lines, default one screen.

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 your handout (Lab #3 - pp. 202-203). 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*, 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?

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

Assume that the file *random.dat* is of a great importance to you and you yourself want to be able to read this file only. (Access permissions for group and others are already removed.) To do this type:

```
chmod  u-w  random.dat
```

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

Type *ls -al* to see if your request has been implemented. 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.

7. Sort utility, multitasking, redirection.

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*. (Note the use of the redirection). To do this, type the following command.

```
sort -nr random.dat > randsor.dat
```

Note that since we have not used the ampersand (*&*), the command (the process) runs in the foreground. Because the file *random.dat* is relatively small, it will be sorted almost immediately and the $ prompt would reappear shortly to take the next command. Now imagine that if you type 500000 in the instruction *#define MAX_SIZE* in the C program on page 4 and your Unix/Linux account disk space is large enough, the program will generate 500000 random numbers. It may take a couple of minutes of the CPU time to sort the numbers in the file *random.dat* (actually it will just take several seconds) and send them to the file *randsor*.da*t*, and you cannot execute any other commands during that time. As you want to have your terminal available for other activities, you will have to permanently kill the process *sort -nr random.dat > randsor.dat* by hitting *^C*. The *$* prompt will appear and you can do two things: 1) sort in the background (*&*) right away, or 2) start in the foreground and than move to the background. Let's try out the 1st approach. Type

```
sort -nr random.dat > randsor.dat &
```

You will see that Unix/Linux will assign a 5- or 6-digit process number to this process and will submit it for execution in background. Type immediately *ps* to list all the processes running on your terminal. Note that the shell itself is the process as well, so do not be surprised when you see a shell process running. There should be at least three or four processes running plus perhaps other Korn or C shells, if you invoked them earlier by *ksh* or *csh* commands, respectively. The system will inform you when the *sort* process is over by sending a message *done*. Finished processes are erased (killed) automatically. (The *sort* process will most likely finish before you type the *ps* command.)

Examine the contents of the file *randsor.dat* using one of the command *more randsor.dat*.

Now let's try the 2nd approach. Let's start the process in the foreground and then move it to the background, so you can do some other useful job in the foreground. The file *randsor.dat* already exists and it will be overwritten. Type:

sort -nr random.dat > randsor.dat

To temporarily suspend this process, hit immediately ^Z, and then type *bg*. The *$* prompt should appear immediately. (Again, the *sort* process will most likely finish before you hit ^Z and type the *bg* command.) Now you can type *ps* again to check the status of processes. You can recall a job executing in the background to the foreground by using an *fg* command. Both *bg* and *fg* commands, used without a process number, refer to the most recent process. To kill a selected process or move it to the foreground from the background or move it to the background from the foreground, you would have to specify a process number. For example *kill process#* (*kill 18987*). Note that since the file *random.dat* is really small, a fraction of a second of the CPU time will be used to sort the file. Thus, it does not make much difference if you sort this file in foreground or background because the *$* prompt will reappear almost immediately. To notice the desired effect of the commands *sort -nr random.dat* > *randsor.dat* and *sort -nr random.dat* > *randsor.dat &*, the data file to sort needs to be very large and contain, say, 500000 or even more, to sort.

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

The last thing we will try is to use an *alias* command. Say, that you want to remove the two files created above: *random.dat* and *randsor.dat*. The command *rm* that does it is a little awkward and we would like to have it in a more readable (meaningful) form. Type

alias erase=rm

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 *.bash_profile* file in your home directory. Type:

erase random.dat randsor.dat

Since *random.dat* has the read status only, the system will display 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.

To save the script in a file named *lab4*, type *^D* at the beginning of the 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. Using SSH Secure File Transfer Client, download the *lab4* file to your desktop. The file may be pretty large. Reduce the size of the file to 3-4 pages by deleting some of its contents and print it.

<u>Submit</u>

(1) Hardcopy of *lab4* file (optional, if you can reduce its size to ≤4 pages only)
(2) A separate typed sheet containing comments what worked well in this lab and what did not. Your comments will allow me to improve the lab in the future. Your comments are important as this is the first iteration of this lab on the CRC RedHat Linux.