CSCD 240
Lab 3

All answers are to be produced by using ssh to login to **cslinux.eastern.ewu.edu** to complete this assignment. In **all cases** you should capture the prompt, the command, and the output from the command.

1) Clearly explain the difference between **which, whereis, grep,** and **find**.

which - shows the full path of (shell) commands.
whereis - locate the binary, source, and manual page files for a command.
grep - searches the named input FILEs for lines containing a match to the given PATTERN.
find - search for files in a directory hierarchy.

2) Issue the find command looking for the file named ld starting at the root directory.
   a. Assuming you are not logged in as root, you should get a list of errors as well as where the file was found. Capture the output and include it in your submission – you do not need to include all the permission errors just a few to get the idea but do include where the file was found.

```
jaimew@cslinux:/$ find -name ld
find: `./home/svn': Permission denied
find: `./home/EASTERN/rparkisoniii': Permission denied
find: `./home/EASTERN/cheale': Permission denied
find: `./home/EASTERN/tpham10': Permission denied
find: `./home/EASTERN/aanderson15': Permission denied
```

   b. Repeat the command (again not as root) – illustrating a method of eliminating the error messages and printing only what was found.

```
jaimew@cslinux:/$ find -name ld 2> /dev/null
./usr/lib/compat-ld/ld
./usr/lib/gold-ld/ld
./usr/bin/ld
./usr/share/doc/binutils/ld
```

3) In class we talked about the '–name' option for the find command.
   a. Explain how to use the size option.

-size n[cwbkMG]
File uses n units of space. The c, w, and b switches allow searching for bytes (b), two-byte words (w) and bytes (c). Switches k, M, G refer to standard metric units.

b. Issue and capture the results of the find command in your home directory that display all files that are greater than 1K. Do not search for more than 3 subfolders. Do not display error messages.

```
jaimew@cslinux:~$ find ~ -maxdepth 3 -size 1k 2>/dev/null
/home/EASTERN/jaimew/netstorage
/home/EASTERN/jaimew/cscd240/lab2/date.txt
/home/EASTERN/jaimew/cscd240/lab2/err.txt
/home/EASTERN/jaimew/cscd240/lab1/test1
/home/EASTERN/jaimew/cscd240/lab1/files.tgz
/home/EASTERN/jaimew/.local/share/gsettings-data-convert
…
/home/EASTERN/jaimew/.mozilla/firefox/profiles.ini
```

4) Use a text editor on the remote machine to create a file named frost.poem that contains the following text:

```
The Road Not Taken by Robert Frost
Two roads diverged in a yellow wood,
And sorry I could not travel both and
be one traveler, long I stood And
looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim
Because it was grassy and wanted wear,
Though as for that the passing there
Had worn them really about the same,
```

a. Use the grep command, capture both the command and the output, to finds all lines, including the line number, that end with a comma.

```
jaimew@cslinux:~$ grep -nr [,$] frost.poem
2:Two roads diverged in a yellow wood,
4:be one traveler, long I stood And
8:Then took the other, as just as fair,
10:Because it was grassy and wanted wear,
12:Had worn them really about the same,
```

b. Use the grep command, capture both the command and the output, to finds all lines, including the line number, containing the word as.

```
jaimew@cslinux:~$ grep -nrw as  frost.poem
5:looked down one as far as I could
8:Then took the other, as just as fair,
11:Though as for that the passing there
```

c. Use the grep command, capture both the command and the output, to finds all lines, including the line number that starts with the word and (case DOES NOT matter).

```
 jaimew@cslinux:~$ grep -nrwi and  frost.poem
3:And sorry I could not travel both and
4:be one traveler, long I stood And
9:And having perhaps the better claim
10:Because it was grassy and wanted wear,
```

d. Use the grep command, capture both the command and the output, to finds all lines, including the line number that starts with the word and (case DOES matter).

```
jaimew@cslinux:~$ grep -nrw and  frost.poem
3:And sorry I could not travel both and
10:Because it was grassy and wanted wear,
```

5) Capture, creating a directory named lab3.

```
jaimew@cslinux:~$ mkdir lab3
```

a. Capture placing a copy of frost.poem in the directory lab3. There should be one copy of frost.poem in your home directory and one in lab3.

```
jaimew@cslinux:~$ cp frost.poem lab3
jaimew@cslinux:~$ ls lab3
frost.poem
```

b. Within your home directory, capture the grep command and its output that will recursively find all instances of the word I (case DOES matter) in all files that end with .poem.

```
jaimew@cslinux:~$ grep -wr I *.poem
And sorry I could not travel both and
be one traveler, long I stood And
looked down one as far as I could
```

6) In lab 1 we examined .bashrc. There are many places to place the PATH variable. Clearly explain .profile, .bash_login, .bashrc and .bash_profile.

To ensure that variables are accessible to a shell at all times, you must place variables in a file that is executed each time a user logs in and starts a BASH shell. These files are called environmental files. The /etc/profile is always executed immediately after login for all users on system and set most environmental variables. After /etc/profile the home directory of the user is searched for the environment hidden files .bash_profile, .bash_login and .profile. These hidden files allow a user to set customiuzed variables independent of BASH shells used by other users on the system.

7) Using a text editor create a file named myScript that contains the following:

```
#!/bin/bash
string=”Hello World”
echo $string
```

a. Try to execute the script with ./myScript and capture the output.

```
jaimew@cslinux:~$ ./myScript
-bash: ./myScript: Permission denied
```

b. Execute and capture the command that will change the permissions on myScript to be user executable without changing any other permissions.

```
jaimew@cslinux:~$ ll myScript
-rw-r--r-- 1 jaimew IT-GenericLinuxGroup 49 Oct  8 23:00 myScript
jaimew@cslinux:~$ chmod 744 myScript
jaimew@cslinux:~$ ll myScript
-rwxr--r-- 1 jaimew IT-GenericLinuxGroup 49 Oct  8 23:00 myScript*
```

c. Execute the script with ./myScript and capture the output.

```
jaimew@cslinux:~$ ./myScript
Hello World
```

8) Using a text editor create a file named secondScript that contains the following:

```
#!/bin/ksh
string=”Hello World”
print $string
```

a. Try to execute the script with ./secondScript and capture the output.
b. Execute and capture the command that will change the permissions on secondScript to be user executable without changing any other permissions.

c. Execute the script with ./secondScript and capture the output.

```
jaimew@cslinux:~$ ll secondscript
-rwxr--r-- 1 jaimew IT-GenericLinuxGroup 48 Oct  8 23:09 secondscript*
jaimew@cslinux:~$ ./secondscript
Hello World
```

d. What does the #! mean?
Indicates to system that the file is a script.

e. In problem 7 what shell did the code execute in?
bash

f. In problem 8 what shell did the code execute in?
ksh

9) Copy secondScript into a file named thirdScript, ensure you preserve the timestamp of secondScript. Capture the output of stat on both files to prove you preserved the timestamp.

```
jaimew@cslinux:~$ stat secondscript
  File: `secondscript'
  Size: 48              Blocks: 8          IO Block: 32768  regular file
Device: 15h/21d Inode: 49062249    Links: 1
Access: (0744/-rwxr--r--)  Uid: (900648664/  jaimew)   Gid: (800000001/IT-GenericLinuxGroup)
Access: 2013-10-08 23:07:26.000000000 -0700
Modify: 2013-10-08 23:09:20.000000000 -0700
Change: 2013-10-08 23:09:20.000000000 -0700
 Birth: -
jaimew@cslinux:~$ stat thirdScript
  File: `thirdScript'
  Size: 48              Blocks: 8          IO Block: 32768  regular file
Device: 15h/21d Inode: 49062256    Links: 1
Access: (0744/-rwxr--r--)  Uid: (900648664/  jaimew)   Gid: (800000001/IT-GenericLinuxGroup)
Access: 2013-10-08 23:07:26.000000000 -0700
Modify: 2013-10-08 23:09:20.000000000 -0700
Change: 2013-10-09 00:14:10.000000000 -0700
 Birth: -
```

10) Using the man page for env

a. Describe (in your own words not with captures from the man page) the output of env command with no arguments.

Output appears to show current definitions of system environmental variables.

b. Describe the similarities and differences of printenv and env

They describe the system environment.

c. Capture a command other than pwd that will show your current working directory.

ls -d

11) Capture the output of printenv in a file named penvout.txt.

jaimew@cslinux:~$ printenv 1>penvout.txt

12) Capture the output of env in a file named envout.txt

jaimew@cslinux:~$ env 1>envout.txt

13) Capture the diff command, ignoring case and white space, on envout.txt and penvout.txt.

```
jaimew@cslinux:~$ diff -ib penvout.txt envout.txt
18c18
< _=/usr/bin/printenv
---
> _=/usr/bin/env
```

14) What is the difference between a shell variable and an environment variable in the bash shell?

A shell variable will affect only the specific shell (bash, csh, ksh) while an environmental variable can affect any shell.

15) Define what a process is and what a job is, clearly explain how jobs differ from processes.

Process: A program loaded into memory and running on the processor performing a specific task.
Job: A fork off the shell running a process in either the foreground or background.

16) In the lab4 directory create the C file named lab4.c with the following code.

```
#include <stdio.h> int
main()
{
    printf("Hello World\n");
    return 0;
}// end main
```

```
jaimew@cslinux:~/lab4$ ll lab4.c
-rw-r--r-- 1 jaimew IT-GenericLinuxGroup 79 Oct  8 23:38 lab4.c
jaimew@cslinux:~/lab4$ stat lab4.c
  File: `lab4.c'
  Size: 79              Blocks: 8          IO Block: 32768  regular file
```

Device: 15h/21d Inode: 49062262 Links: 1
Access: (0644/-rw-r--r--) Uid: (900648664/ jaimew) Gid: (800000001/IT-GenericLinuxGroup)
Access: 2013-10-08 23:38:03.000000000 -0700
Modify: 2013-10-08 23:38:03.000000000 -0700
Change: 2013-10-08 23:38:03.000000000 -0700
Birth: -

17) Give the grep command that will start in your home directory and show the file names and line numbers containing the term "stdio" in all .c files in the home directory and all directories below the home.

```
jaimew@cslinux:~$ grep -rn "stdio" */
grep: cscd240/lab2/stu2: Permission denied
grep: cscd240/lab2/stu1: Permission denied
lab4/lab4.c:1:#include<stdio.h>int
```

18) Consider the following command ls – al | more.
   a. How many processes are created with that command?
      Two
   b. What exactly does "|" do in this command?
      The pipe directs the output from the ls –al into the more command.

19) Using the man page for ps
   a. Issue and capture the ps command with the appropriate options to allow listing of all processes in the system.

```
jaimew@cslinux:~$ ps -A
PID    TTY        TIME          CMD
1      ?      00:00:02          init
2      ?      00:00:00          kthreadd
3      ?      00:00:41          ksoftirqd/0
5      ?      00:00:00          kworker/u:0
6      ?      00:00:01          migration/0
7      ?      00:00:11          watchdog/0
…
29242 pts/28  00:00:00          ps
31207 ?       00:00:01          a.out
31870 ?       00:00:00          a.out
32251 ?       00:00:01          a.out
32749 ?       00:00:01          a.out
```

b. Using the output from part A, what was the first process started and by whom was it started?

```
jaimew@cslinux:~$ ps -U root
PID TTY          TIME         CMD
1        ?        00:00:02        nit
```

c. What was the first non-root process that was started?

```
jaimew@cslinux:~$ ps -U root
PID              TTY         TIME       CMD
1                ?        00:00:02       init
```

d. What was the last process started and by whom?

```
jaimew@cslinux:~$ ps 29771
PID              TTY             STAT            TIME            COMMAND
29771            ?               S               0:00            [kworker/0:0]
```

20) Using a text editor create the following C programmed named almostEndless.c #include <stdio.h>

```
int main()
{
      int x = 0;
      while(x < 20000000)
      {
printf("..");
fflush(stdout);
sleep(3);

      }// end while

      return 0;
}// end main
```

a. Compile your program with gcc almostEndless.c
b. Start your program with ./a.out
c. With your program running, describe the commands you would use (without using ctrl-c) to terminate that program, from the same terminal window in which it was started.

```
jaimew@cslinux:~$ ./a.out
....^Z                                  this suspends job
[1]+  Stopped                 ./a.out
jaimew@cslinux:~$ kill -9 4496          this kills via pid
jaimew@cslinux:~$ jobs                  checking to ensure kill
[1]+  Killed                  ./a.out
```

d. Execute and capture the commands, using process notation, to terminate a.out

See above

e. Restart your program with ./a.out

f. Execute and capture the commands, using job notation, to terminate a.out

jaimew@cslinux:~$ ./a.out
.....^Z
[1]+ Stopped ./a.out
jaimew@cslinux:~$ kill %1 && fg
./a.out
Terminated

21) In a single terminal window capture the command to start a.out 3 times each running as background jobs:

a. What are the job numbers of the above?

jaimew@cslinux:~$ ./a.out & ./a.out & ./a.out &
[1] 5736
[2] 5737
....[3] 5738

b. What are the process ID numbers of the above?

jaimew@cslinux:~$ ./a.out & ./a.out & ./a.out &
[1] 5736
[2] 5737
....[3] 5738

c. Capture the command and output to bring the second a.out to the foreground.

jaimew@cslinux:~$ .....fg 2
./a.out

d. Capture the command(s) to send a.out back to the to the background.

.....^Z
[2]+ Stopped ./a.out
jaimew@cslinux:~$ .....bg 2....
[2]+ ./a.out &
jaimew@cslinux:~$ .....jobs
[1] Running ./a.out &
[2]- Running ./a.out &
[3]+ Running ./a.out &

e. Capture the command(s) to kill the third a.out using its job number.

jaimew@cslinux:~$ .....kill -.....9 ....%.....3
jaimew@cslinux:~$ ....j..obs..
[1] Running ./a.out &
[2]- Running ./a.out &
[3]+ Killed

f. Capture the command(s) to kill the first a.out using its process number.

.....k..il..l -9 .....75..17
jaimew@cslinux:~$ ..jobs..
[1]- Killed ./a.out
[2]+ Running ./a.out &

g. Can CTRL C be used to kill any job? Why or why not? Clearly explain why or why not.

Nope, because it does not work.

TO TURN IN:

- A PDF file - Name this text file your last name, first letter of your first name lab3.pdf. This file will contain all your answers. I want the question copied and then the answer to the question below it.
- A zip file that contains your PDF, all files created during this lab (C files, scripts, etc).

You zip will be named your last name first letter of your first name lab3.zip (example steinerslab3.zip)