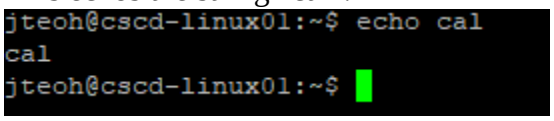
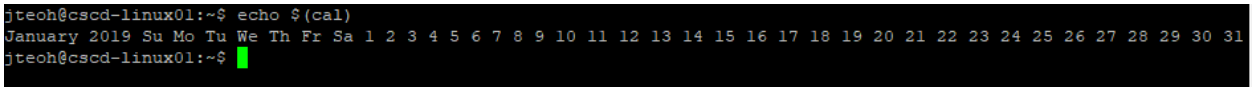
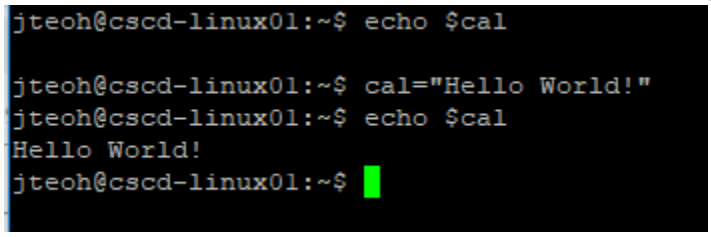
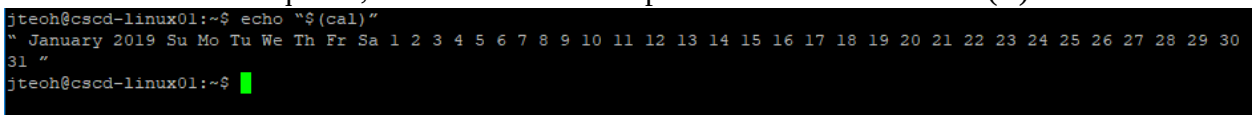
echo’

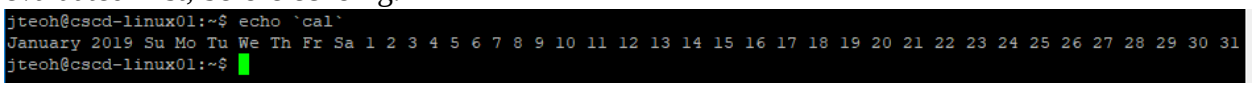
1. What are the differences among the following commands? Explain with screenshots.
(6 pts , 1 point each)

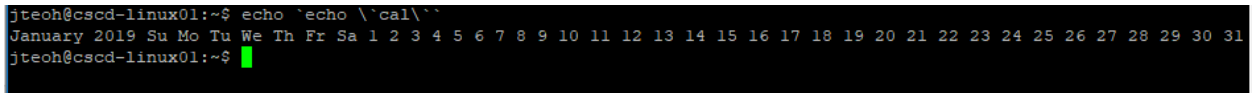
a) echo cal
- This echos the string “cal”.
- 

b) echo $(cal)
- This shows the current calendar, but non formatted.
- 

c) echo $cal
- This echos the variable cal. If the variable has no value, nothing will be displayed.
- 

d) echo “$(cal)”
- The double quotes will display anything within the quotes literally, with the exception of escape characters and variables. Seeing that this command uses variables and double quotes, we know that the output should be similar with (B)
- 

e) echo `cal` ←This is back quote
- when backticks/backquotes are used, the argument within the ticks are evaluated first, before passing it into the other command. In this example, “cal” will be evaluated first, before echoing.
- 

f) echo `echo \`cal\`` ← This is back quote
- This should be similar to E as well.
- 

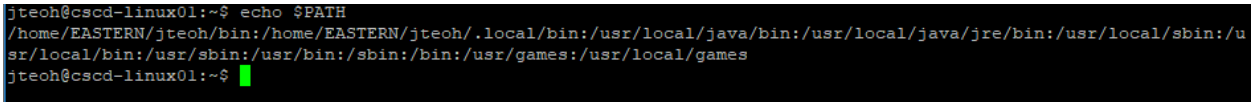
Environment variables

2 What command will show you all the environment variables?

"env" will work.

1. What command will display the environment variable named PATH? Show both with screenshot.

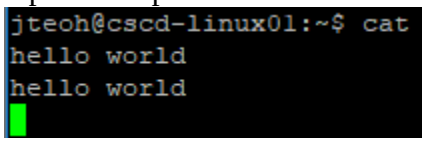
"echo $PATH" will work.



I/O

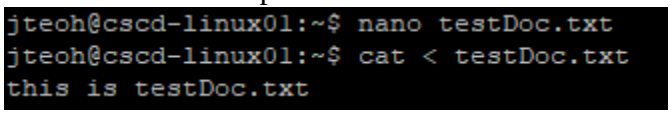
3) What are the differences among the following commands. Explain with examples and screenshot. (4 points, 1 point each)

cat

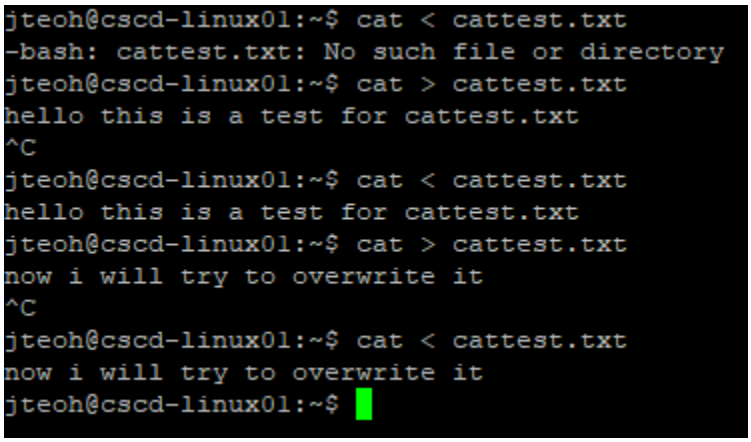
- Cat without any argument will simply show the users input as output. Standard input to output.
- 

- In my example, I typed in Hello World and it showed me what I type.

cat < filename

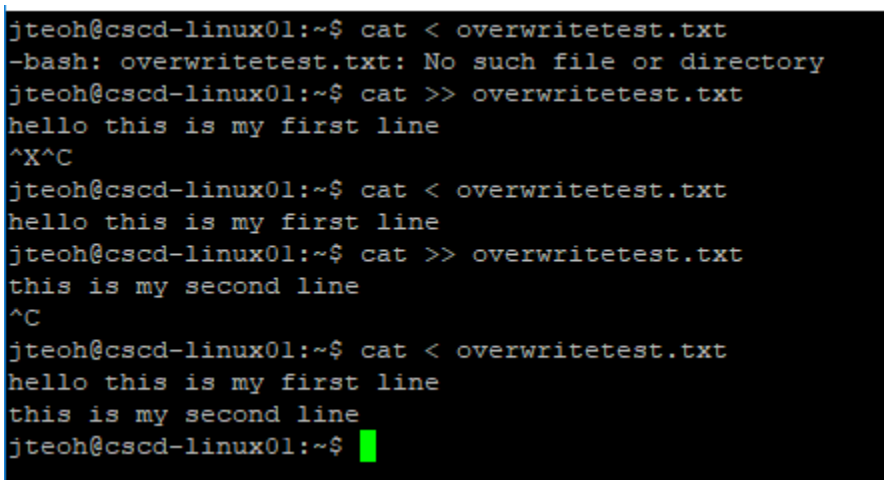
- Cat < filename will print whatever is in filename to the screen.
- 

cat > filename

- This will save whatever is typed into the standard input as filename. If the file does not exist, the file will be created. If it exists, the file will be OVERWRITTEN.
- 

cat >> filename

- This will concatenate/merge the input to cat with the existing file. If the file does not exist, it will simply be created. Does not overwrite.

- 

4) **Write a command that counts the total number of lines the string "bird" exists in a file named "The Rhyme of Ancient Mariner" in your current directory. (1 point)**

   - Grep -c bird "The Rhyme of Ancient Mariner"

5) **Write a command that searches the string "line" in all .c and .txt files starting from your current directory and all sub directories. (1 point)**

   - grep -r –include=*.{c,txt} "line" .
   - The dot at the end is part of the command.

Metacharacters in Regular Expression

6) **What will the following patterns match? Explain. (4 points, 1 point each)**

   a) ^bags$

   - This looks for lines that only have "bags". ^ indicates that it looks for a line that STARTS with bags, but $ also indicates that the line has to END with bags. Thus, the line can ONLY have bags.

   b) ^…$

   - This looks for a line that starts and end with 3 alphanumeric characters. The dots represent any alphanumeric character, but just one, as opposed to the * wildcard where one or more characters can be represented with one wildcard. The dot will only allow one.

   c) l.g

   - This looks for words that start with l and end with g, with any alphanumeric in between. Unlike previous commands, the word can be in a line that is populated by others, as it does not have ^ and $ together.

   d) ^\.

   - This looks for a line that starts with any character. \. Is an escape sequence.

‘grep’, ‘find’, pipe

7) **Consider the following file named “FruitsList.txt”. Try the following commands and explain each output with screenshot. (6 points, 1 point each)**

a) **grep “[A-Z]e” FruitsList.txt**
- This looks for words that start with any uppercase letters from A to Z, followed by a lowercase e, in the file FruitsList.txt. In this case, nothing will be shown.
-
```
jteoh@cscd-linux01:~$ grep "[A-Z]e" FruitsList.txt
jteoh@cscd-linux01:~$
```

b) **grep -i “[A-Z]e” FruitsList.txt**
- This is similar to (a), but this ignores upper and lower case, as indicated by the -i argument.
-
```
jteoh@cscd-linux01:~$ grep -i "[A-Z]e" FruitsList.txt
apple
Orange
Pineapple
lemon
berry
jteoh@cscd-linux01:~$
```

c) **grep “[^A-Z]e” FruitsList.txt**
- This will look for any words that do NOT start with uppercase A to Z, followed by a lowercase e. While ^ is normally used to indicate the beginning of a line, using it in a square bracket will mean to exclude whatever follows, if used in the beginning of the regex pattern. [^abc] will exclude abc, but [a^bc], it will look for ab^c.
-
```
jteoh@cscd-linux01:~$ grep "[^A-Z]e" FruitsList.txt
apple
Orange
Pineapple
lemon
berry
jteoh@cscd-linux01:~$
```
- Notice how all of them (red highlighted) have no uppercase letters before the lowercase e.

d) **grep –i “^[A-Z]e” FruitsList.txt**
- This looks for a line that starts with (indicated by ^) any upper or lowercase alphanumeric character (indicated by -i), followed by a lowercase e.
-
```
jteoh@cscd-linux01:~$ grep -i "^[A-Z]e" FruitsList.txt
lemon
berry
jteoh@cscd-linux01:~$
```

e) **grep “^le” FruitsList.txt**

- This looks for line that starts with “le”.

```
jteoh@cscd-linux01:~$ grep "^le" FruitsList.txt
lemon
jteoh@cscd-linux01:~$
```

f) grep “le$” FruitsList.txt

- This looks for lines that end with “le”

```
jteoh@cscd-linux01:~$ grep "le$" FruitsList.txt
apple
Pineapple
jteoh@cscd-linux01:~$
```

8) Suppose you are in your home directory. What are the differences between the following commands? Explain with screenshot.
1 point

find . –name "*.txt"

- This looks for any .txt file type recursively starting from the current directory.

```
jteoh@cscd-linux01:~$ find . -name "*.txt"
./testDoc.txt
./cattest.txt
./FruitsList.txt
./overwritetest.txt
jteoh@cscd-linux01:~$
```

find ~ -name "*.txt"

- This looks for any .txt file type recursively starting from the home (~) directory.

```
jteoh@cscd-linux01:~$ find ~ -name "*.txt"
/home/EASTERN/jteoh/testDoc.txt
/home/EASTERN/jteoh/cattest.txt
/home/EASTERN/jteoh/FruitsList.txt
/home/EASTERN/jteoh/overwritetest.txt
jteoh@cscd-linux01:~$
```

Notice that while both show the same files, one shows the absolute path and one shows the relative path.

Write a command that finds all text files in your home directory and subdirectory and shows the long listing.

- Find ~ -name “*.txt” | xargs ls-l

```
jteoh@cscd-linux01:~$ find ~ -name "*.txt" |xargs ls -l
-rw-r--r-- 1 jteoh IT-GenericLinuxGroup 31 Jan 23 19:14 /home/EASTERN/jteoh/cattest.txt
-rw-r--r-- 1 jteoh IT-GenericLinuxGroup 47 Jan 23 19:40 /home/EASTERN/jteoh/FruitsList.txt
-rw-r--r-- 1 jteoh IT-GenericLinuxGroup 51 Jan 23 19:16 /home/EASTERN/jteoh/overwritetest.txt
-rw-r--r-- 1 jteoh IT-GenericLinuxGroup 20 Jan 23 19:11 /home/EASTERN/jteoh/testDoc.txt
jteoh@cscd-linux01:~$
```

-

What will the following commands do? Explain with screenshots. (2 points, 1 point each)

a) ls –l | grep '^…….rw'

This first generates the long listing of the current directory, then passes it onto grep to look for lines that start with 7 alphanumeric characters, then rw. This indicates that it's looking for files or directories with specific permissions..

```
jteoh@cscd-linux01:~$ ls -l | grep '^.......rw'
lrwxrwxrwx 1 jteoh IT-GenericLinuxGroup   13 Jan 16 20:15 netstorage -> /mnt/ns-jteoh
jteoh@cscd-linux01:~$
```

b) grep –n variable *.[ch]

This looks for the word "variable" inside of any .c or .h file. The -n indicates that the output will have line numbers for whichever line that matches the pattern given. However, I do not have any .c or .h file with such pattern, so nothing will be displayed.

```
jteoh@cscd-linux01:~$ grep -n variable *.[ch]
jteoh@cscd-linux01:~$
```

11) What is process? How will you differentiate processes from jobs?

Processes are things that are happening in the background that is not session dependent. Jobs are session dependent, and as such will not continue to run if the current session is terminated. Processes on the other hand, will continue running.

12) What are the difference between the following commands: Explain with screenshot.

ps and ps –aux.

ps prints all currently running processes, however it only shows the processors of the current user.

-a argument shows all users

```
jteoh@cscd-linux01:~$ ps -a
  PID TTY          TIME CMD
 5667 pts/5    00:00:00 bash
 6462 pts/1    00:00:00 cat
 7033 pts/1    00:00:00 nano
 7182 pts/2    00:00:00 ps
jteoh@cscd-linux01:~$
```

-x shows processes that have no TTY associated with it

```
jteoh@cscd-linux01:~$ ps -x
  PID TTY      STAT   TIME COMMAND
 7204 pts/2    R+     0:00 ps -x
23060 ?        S      0:00 sshd: jteoh@pts/
23061 pts/2    Ss     0:00 -bash
```

and -u modifies the output format, allowing more columns and information.

```
jteoh@cscd-linux01:~$ ps -u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
jteoh     7363  0.0  0.0  44084  3636 pts/2    R+   20:07   0:00 ps -u
jteoh    23061  0.0  0.1  33936  6432 pts/2    Ss   18:58   0:00 -bash
```

-aux will just combine all of these 3 arguments together, showing all processes by all users, even if it's not associated with any TTY, and modify the output to show more information. (There are more processes shown, but I cut the screenshot, as otherwise it would be too big)

```
jteoh@cscd-linux01:~$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  37964  6000 ?        Ss   06:12   0:07 /sbin/init
root         2  0.0  0.0      0     0 ?        S    06:12   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    06:12   0:00 [ksoftirqd/0]
root         7  0.0  0.0      0     0 ?        S    06:12   0:06 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    06:12   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    06:12   0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        S    06:12   0:00 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S    06:12   0:00 [watchdog/1]
root        12  0.0  0.0      0     0 ?        S    06:12   0:00 [migration/1]
root        13  0.0  0.0      0     0 ?        S    06:12   0:00 [ksoftirqd/1]
root        16  0.0  0.0      0     0 ?        S    06:12   0:00 [watchdog/2]
root        17  0.0  0.0      0     0 ?        S    06:12   0:00 [migration/2]
root        18  0.0  0.0      0     0 ?        S    06:12   0:00 [ksoftirqd/2]
root        20  0.0  0.0      0     0 ?        S<   06:12   0:00 [kworker/2:0H]
root        21  0.0  0.0      0     0 ?        S    06:12   0:00 [watchdog/3]
root        22  0.0  0.0      0     0 ?        S    06:12   0:00 [migration/3]
root        23  0.0  0.0      0     0 ?        S    06:12   0:00 [ksoftirqd/3]
root        25  0.0  0.0      0     0 ?        S<   06:12   0:00 [kworker/3:0H]
root        26  0.0  0.0      0     0 ?        S    06:12   0:00 [kdevtmpfs]
root        27  0.0  0.0      0     0 ?        S<   06:12   0:00 [netns]
root        28  0.0  0.0      0     0 ?        S<   06:12   0:00 [perf]
```