LAB 8 (Jul 19) — Unix Utilities and common functionalities

Due: Aug 4 (Sun) 11:59 pm (optional)

Part I Unix Utilities/commands

The purpose of this lab exercise is for you to get some hands-on experience on using some fundamental Unix utilities (commands). After this lab, you are expected to be able to accomplish lots tasks using command line utilities, without resorting to your GUI based utilities such as File Manager. Command line execution is faster than GUI based utilities in general. Also in some systems GUI tools are not available at all and thus using command line utilities is your only option.

We covered in class the following basic utilities/commands: **man**, **pwd**, **ls**, **cd**, **mkdir**, **rmdir**, **cat**, **more**, **head**, **tail**, **cp**, **mv**, **rm**, **wc**, **file**, **chmod**, **chgrp**. We also discussed how to use pipe to use one utility's output as the input of another utilities. In the next class we will also cover ed **grep/egrep**, **uniq**, **sort**, **cmp/diff**, **cut**, **find** etc.

You can get the details of each utility by using utility **man**. E.g., **man chmod** or even better, **man 1 chmod**

This part contains 110 (small) practices.

Note: Each question should be solved with only <u>one</u> entry of utility (e.g., `cp file1 file2`) or a pipeline of utilities (e.g., `cat file1 | sort | wc`).

0. Login to your home directory, and change to Bourne (again) shell by issuing `sh` or `bash`. The prompt should change from `%` to `$`. Now create a working directory for this lab, and navigate to the working directory in terminal.

1. There is a file named `xxx` in directory **`/eecs/dept/course/2018-19/S/2031/`**

   Copy this file to your current working directory, using one entry of utility (command).

2. Check that the file is copied here.

```
$ ls xxx
xxx
$
```

```
2  ls xxx
or ls
```

```
1  cp /eecs/dept/course/2018-19/S/2031/xxx .
or cp /eecs/dept/course/2018-19/S/2031/xxx .\
or cp /eecs/dept/course/2018-19/S/2031/xxx .\xxx
```

3. There are two files named `xFile2` and `xFile3` in directory **`/eecs/dept/course/2018-19/S/2031/`** Copy these two files to your current working directory using <u>one</u> entry of utility. Assume these two files are the only files whose names begin with `'xFile'`. Hint: so you can use `xFile*` or `File?` to match these two files. (If you don't understand * and `?`, look for page 10 of the *Guided Lab Tour for CSE1020*). Note, don't confuse that with * and ? that are used in (extended) regular expression.

4. Verify that the two files are copied successfully to the current directory.

```
$ ls xFile*
xFile2  xFile3
$ ls
xxx xFile2  xFile3
```

```
3  cp /eecs/dept/course/2018-19/S/2031/xFile2  eecs/dept/…/2031/xFile3   .
or cp /eecs/dept/course/2018-19/S/2031/xFile?  .
or cp /eecs/dept/course/2018-19/S/2031/xFile* .
or cp /eecs/dept/course/2018-19/S/2031/xFile[23] .
```

```
4  ls xFile2 xFile3  or ls xFile*  or ls xFile?
```

5. Rename file `xxx` to `xFile1`

```
5 mv xxx  xFile1
```

6. (1) Verify that the renaming is successful.

   One (professional) way to verify if an execution of a utility is successful is to examine the exit code (return value) of the executed process, which is stored in a system variable **`$?`**. Issue **`echo $?`** You should see `0`, which means successful (this is opposite to C where 0 means false).

   (2) Also verify by listing files in current working directory

```
$ ls
xFile1 xFile2 xFile3
```

```
6 ls xFile1 xFile2 xFile3  or ls xFile*  or  ls xFile?
```

7. (1) Create a sub-directory named `2019` under your current working directory. (2) Then still in the current working directory, create a subdirectory `lab8a` under `2019`.

```
7(1)  mkdir 2019    (2) mkdir 2019/lab8a
```

8. Verify that the two directories are created successfully, by recursively listing directory 2019 and its contents.

```
$ ls -R 2019
2019:
lab8a

2019/lab8a:
```

```
8  ls -R 2019
```

```
9(1)  rmdir 2019/lab8a
9(2)  rmdir 2019
```

```
10  ls 2019
```

9. Still in current working directory, (1) remove `lab8a` using `rmdir` and (2) then remove `2019` using `rmdir`

10. Verify 9. by trying to list directory `2019`. Should get `ls: cannot access 2019: No such file or directory`

11. (1) Create (again) a subdirectory `2019` and `2019/lab8a` under the current working directory, using `mkdir 2019/lab8a`. What do you get?

```
11(1) mkdir: cannot create directory '2019/lab8a': No such file or directory
```

```
11(2) echo $?
```

   (2) Check the exit code of execution. You should get 1, which means unsuccessful (note that 1 means true in C).

12. (1) Fix the problem in 11. Hint: try `-p` flag of `mkdir`

```
12 (1)  mkdir -p 2019/lab8a
```

```
12(2) echo $?
```

   (2) Check the exit value of execution. You should get 0, which means successful

```
12(3) ls -R 2019
```

   (3) confirm by recursively listing directory 2019 and its contents. You should see same result as in 8.

13. Move `xFile1` into subdirectory `lab8a` (with same name)

```
13  mv xFile1 2019/lab8a
```

14. Then move all the other 2 files (together) into `lab8a` (using one entry of utility)

15. Verify that the creation and moving (12-14) were successful, by recursively listing directory 2019 and contents.

```
$ ls -l -R 2019
./2019:
total 4
drwx------ 2 yourname ugrad 4096 Mar 25 15:12 lab8a

./2019/lab8a:
total 12
-rwx------ 1 yourname ugrad 145 Mar  25 15:11 xFile1
-rwx------ 1 yourname ugrad 145 Mar  25 15:11 xFile2
-rwx------ 1 yourname ugrad  87 Mar  25 15:11 xFile3
```

```
14  mv xFile* 2019/lab8a
or  mv xFile? 2019/lab8a
or  mv xFile[23] 2019/lab8a
```

Note that on each line, the first character
- `-` means this entry is a regular file,
- `d` means this entry is a directory.

16. (1) Navigate to `2019` and (2) list the files in subdirectory `lab8a`.

```
16 (1) cd 2019
   (2) ls  -l lab8a
```

```
$ ls -l lab8a
total 12
-rwx------ 1 yourname ugrad 145 Mar  25 15:11 xFile1
-rwx------ 1 yourname ugrad 145 Mar  25 15:11 xFile2
-rwx------ 1 yourname ugrad  87 Mar  25 15:11 xFile3
```

17. Then list the information of subdirectory `lab8a` itself

```
17  ls  -ld   lab8a
or  ls -l -d  lab8a
```

```
$ your-command
drwx------ 2 yourname ugrad 48 Mar  25 15:12 lab8a
```

18. Copy directory `lab8a` to a new directory named `lab8b`, under same direcoty (using one utility).

19. Verify that `lab8b` is created and the two directory are identical

```
18  cp -r  lab8a   lab8b
```

```
19  ls -l *
or  ls lab8a lab8b
or  ls lab8?
or  ls lab8*
or  ls lab8[ab]
```

```
$ ls -l *
lab8a:
total 12
-rwx------ 1 yourname ugrad  145 Mar  25 23:32 xFile1
-rwx------ 1 yourname ugrad  145 Mar  25 23:50 xFile2
-rwx------ 1 yourname ugrad  87  Mar  25 23:50 xFile3

Lab8b:
total 12
-rwx------ 1 yourname ugrad  145 Mar  25 23:32 xFile1
-rwx------ 1 yourname ugrad  145 Mar  25 23:32 xFile2
-rwx------ 1 yourname ugrad  87  Mar  25 23:32 xFile3
```

20. Remove the whole directory `lab8a` using `rmdir`. What happened?

```
20 rmdir lab7a
rmdir: failed to remove
`lab8a/': Directory not empty
```

```
21   echo $?
```

21. Examine the exit code of the above execution, you should get 1, which means something wrong happened.

22. Remove the whole directory `lab8a` using a more effective utility.

```
22  rm -r lab8a
```

23. (1) Verify the exit code of above execution, you should get 0 now

```
23(1)  echo $?
```

(2) Verify by trying to list `lab8a`

```
23(2) ls lab8a
or    ls -l lab8a
or    ls -ld lab8a
```

```
$ ls lab8a
ls: cannot access lab8a: No such file or directory
```

24. Move `xFile1`, which is in subdirectory `lab8b`, to current (parent) directory, using relative pathname.

25. Verify that the above move was successful. Instead of listing the files, let's verify by searching for the files.

```
24 mv  lab8b/xFile1  .
or mv  lab8b/xFile1  ./xFile1
or mv  lab8b/xFile1  xFile1
```

```
$ find . -name "xFile*"  or  find . -name "xFile?"
./lab8b/xFile2
./lab8b/xFile3
./xFile1
```

```
25   find . -name "xFile*"
or   find . -name "xFile?"
```

```
25   " " can be ' '
```

26. Change the name of directory `lab8b` to `lab8working`

```
26  mv lab8b lab8working
```

27. Navigate to directory `lab8working`

```
27 cd lab8working  or  cd ./lab8working
```

28. Verify that you are in `lab8working`

```
$ your-command
/cs/home/your_account/…/2019/lab8working
```

```
28 pwd
```

```
29 mv ../xFile1  .
```

29. Move `xFile1` (which is in the parent directory) into the current directory using relative pathname.

30. Verify that the move was successful by listing all the files currently in `lab8working`

```
$ ls -l
total 12
-rwx------ 1 yourname ugrad  145 Mar 25 16:58 xFile1
-rwx------ 1 yourname ugrad  145 Mar 25 16:58 xFile2
-rwx------ 1 yourname ugrad  87  Mar 25 16:58 xFile3
```

```
30  ls -l
```

31. Issue the following commands, observe that `cat` reads inputs from stdin and then prints to stdout.

```
$ cat
Hi
Hi
There
There
^ D
$
```

32. (1) Issue the following commands, observe that inputs from stdin are written into file **`temp`**.

```
$ cat > temp
Hi
There
^ D
$ cat temp
```

```
32 (2) rm temp
```

(2) Remove file **`temp`**.

```
33 more xFile1     more < xFile1
or cat  xFile1     cat < xFile1
```

33. Display on stdout the contents of file `xFile1`

```
34 more xFile1 xFile2 xFile3
or more xFile?   or  more xFile*
or more xFile[1-3] or more xFile[123]
```

34. Display on stdout the contents of the three files with one entry (Try `more xFile1 xFile2 xFile3` or `more xFile`? Use space bar to proceed.)

```
35  wc -l xFile1
or  cat xFile1 | wc -l
or  more  xFile1 | wc -l
```

35. Display the number of lines in `xFile1`. You should get 5.

```
36  head -2 xFile1
```

36. Display (only) the first two line of `xFile1`

```
37  tail -3 xFile2
```

37. Display the last 3 lines of `xFile2`

38. (1) Confirm that `xFile1` and `xFile2` are identical now, using a utility, which should return silently (Hint: `cmp` or `diff`). (2) Examine the e

```
38/39  cmp xFile1  xFile2
```

```
38/39 diff xFile1  xFile2
```

39. (1) Confirm that `xFile1` and `xFile2` are identical, using another utility, which should return silently (`diff` or `cmp`) . (2) Examine the exit code, you should get 0.

```
40 diff xFile2  xFile3
```

40. (1) Show that `xFile2` and `xFile3` are not identical, using `diff` utility, which will not be silent this time. Try to understand the message but don't spend too much time on it. (2) Examine the exit code, you should get 1.

41. (1) Show that `xFile2` and `xFile3` are not identical, using `cmp` utility, which will not be silent this time. Try to understand the message but don't spend too much time on it. (2) Examine the exit code, you should get 1.

FYI: these two utilities were used by some professors to do automated grading of your lab or labtest :

```
gcc yourCode.c
a.out > yourOutputFile
cmp yourOutputFile professorsOutputFile
echo $?
```

```
41 cmp xFile2  xFile3
```

A student gets 0 if the last command prints 1. That means, student gets 0 even if the student's output contains an extra white space. ☹

42. (1) Concatenate the contents of the three files into a new file `xFile123`, in the order of `xFile1`, `xFile2` and `xFile3`. (2) After that, show on stdout the content of `xFile123`.

```
$ your_command
$ more xFile123
John Smith  1222 26 Apr 1956
Tony Jones  2152 20 Mar 1950
John Duncan 2    20 Jan 1966
Larry Jones 3223 20 Dec 1946
Lisa Sue    1222 4  Jul 1980
John Smith  1222 26 Apr 1956
Tony Jones  2152 20 Mar 1950
John Duncan 2    20 Jan 1966
Larry Jones 3223 20 Dec 1946
Lisa Sue    1222 4  Jul 1980
John Smith  1222 26 Apr 1956
John Duncan 2    20 Jan 1966
Larry Jones 3223 20 Dec 1946
```

```
42 (1) cat xFile1 xFile2  xFile3 > xFile3

   (2) cat xFile3    or   more  xFile3
```

43. Sort lines in file `xFile123` so identical lines are adjacent now

```
$ your_command
John Duncan 2    20 Jan 1966
John Duncan 2    20 Jan 1966
John Duncan 2    20 Jan 1966
John Smith  1222 26 Apr 1956
John Smith  1222 26 Apr 1956
John Smith  1222 26 Apr 1956
Larry Jones 3223 20 Dec 1946
Larry Jones 3223 20 Dec 1946
Larry Jones 3223 20 Dec 1946
Lisa Sue    1222 4  Jul 1980
Lisa Sue    1222 4  Jul 1980
Tony Jones  2152 20 Mar 1950
Tony Jones  2152 20 Mar 1950
```

```
43  sort xFile123
or  cat  xFile123  |  sort
or  more xFile123  |  sort
```

44. Show on the stdout the content of `xFile123`, but with identical lines merged. Hint: utility `uniq` will do the job

```
44  sort xFile123  |  uniq
or  cat xFile123   |  sort | uniq
```

```
$ your_command
John Duncan 2    20 Jan 1966
John Smith  1222 26 Apr 1956
Larry Jones 3223 20 Dec 1946
Lisa Sue    1222 4  Jul 1980
Tony Jones  2152 20 Mar 1950
```

```
45  sort xFile123  |  uniq > xFile123compact
or  cat xFile123 | sort | uniq > xFile123compact
```

```
46  cat xFile123compact
or  more xFile123compact
```

45. Merge the identical lines in `xFile123` and save the result into a new file `xFile123compact`.

    Hint: Just redirect the output of 44 using redirection `>`

46. Show on the `stdout` the content of `xFile123compact`. You should get same output as in question 44.

```
47  cat: xFile1: Permission
denied
```

47. Issue `chmod u-r xFile1`. This removes the read permission of user (owner). Now examine the resulting permission mode of the file. You should get `--wx------` Now issue `cat xFile1` What do you get?

48. Issue `chmod 775 xFile1`, and then examine the resulting permission mode of the file. What do you get? You should get `-rwxrwxr-x` Can you understand what we are doing here?

```
48  chmod 775 xFile1
    ls -l xFile1
```

49. Now issue `chmod 777 xFile1`, and then examine the resulting permission mode

    You should get `-rwxrwxrwx` Can you understand what we are doing here?

```
49  chmod 777 xFile1
    ls -l xFile1
    -rwxrwxrwx   …
```

50. Change the permission of `xFile123compact` using octal numbers so that the permission mode becomes

    `-rwxr--r--  1 yourname ugrad 140 Mar 25 17:23 xFile123compact`

```
50 chmod 744 xFile123compact
```

51. Change the permission of `xFile123compact` by adding an `execute` permission to `groups`. You should get the following result: `-rwxr-xr--  1 yourname ugrad 145 Mar 25 17:23 xFile123compact`

```
51  chmod g+x xFile123compact
```

52. Change the permission of `xFile123compact` by adding a `write` permission to `groups`, and remove `read` permission of the `others` of the file. **You should issue chmod only once**. You should get the following result:

    `-rwxrwx---  1 yourname ugrad 145 Mar 25 17:23 xFile123compact`

```
52  chmod g+w,o-r xFile123compact
```

53. Change the permission of `xFile123compact` by removing `write` permission from `group`, and adding `write` and `execute` permission to `others`. **You should issue chmod only once.** You should get the following result:

    `-rwxr-x-wx  1 yourname ugrad 145 Mar 25 17:23 xFile123compact`

```
53 chmod g-w,o+wx xFile123compact
```

54. Change the permission of `xFile123` by adding the `read` permission to `user`, `group` and `others` (although they may already have one). **You should issue chmod only once.** You should get the following result:

    `-rw-r--r--  1 yourname ugrad 145 Mar 25 31 17:23 xFile123`

55. Modify `xFile1` by adding a new line at the end of the file. This can be done by

```
$ echo "this is a xxx new line" >> xFile1    or
$ cat >> xFile1
this is a xxx new line
^ D
```

```
54 chmod a+r xFile123    or  chmod ugo+r xFile123  or
   chmod u+r,g+r,o+r xFile123
```

```
55 echo "this is a xxx new line" >> xFile1
```

```
56 chmod u-w xFile1
   echo "this is a xxx new line" >> xFile1
   xFile1: Permission denied.
```

56. Remove the `write` permission of the `owner` of **`xFile1`**, and try 55 again. What do you get?

Question 57-58 should be done without using `sort`. Utility `ls` can do some sorting itself.

57. (1) List the files in the current directory, sorted by the modification time. By default "newest first", so `xFile1` should be the first file in the list and other files are also sorted accord

```
57(1) ls -l -t    or    ls  -lt
```

```
57(2) ls -l -t -r    or    ls  -ltr
```

```
$ your_command
total 20
-r-xrwxrwx 1 yourname ugrad 166 Mar  25 14:20 xFile1
-rwxr-x-wx 1 yourname ugrad 145 Mar  25 14:12 xFile123compact
-rw-r--r-- 1 yourname ugrad 377 Mar  25 14:11 xFile123
……
```

(2) List the files, sorted by the modification time, in reverse order. `xFile1` should become the last file in the list.

58. (1) List the files, sorted by the size of the files. By default, "biggest first", so `xFile123` should be the first file in the list and other files are also sorted accordi

```
58(1) ls -l -S    or    ls  -lS
```

```
$ your_command
total 20
-rw-r--r-- 1 yourname ugrad 377 Jul  6 13:35 xFile123
-r-xrwxrwx 1 yourname ugrad 168 Jul  6 13:42 xFile1
-rwxr-x-wx 1 yourname ugrad 145 Jul  6 13:37 xFile123compact
-rwx------ 1 yourname ugrad 145 Jul  6 13:27 xFile2
-rwx------ 1 yourname ugrad 87  Jul  6 13:27 xFile3
```

(2) List the files, sorted by the size of the files, in reverse order.

```
58(2)  ls -l -S -r    or    ls  -lSr
```

59. Try to get the type of the file `xFile123compact` (Hint: use `file` utility)

```
xFile123compact: ASCII text
```

```
59 file xFile123compact
```

60. Recall that utility `who` lists the people who are currently logged on to the EECS server such as red.cse.yorku.ca. Get how many people are currently logged on to red server. (If you are in the prism lab, issue **`ssh red.`**)

61. Sort the list of people who are currently logged on.

```
60  who | wc -l
```

```
61  who | sort
```

62. Sort the list of people who are currently logged on, based on login date (the 3rd column), in chronological order.

63. Display only the three earliest people who are currently logged on the system.

Hint: pipe the result of 62 to utility **`head`**

```
62 who | sort -k 3
```

```
63 who | sort -k 3 | head -3
```

64. Sort `xFile123compact` according to the numerical value of the 3rd field

```
$ sort -k3 xFile123compact
John Smith   1222 26 Apr 1956
Lisa Sue     1222 4  Jul 1980
Tony Jones   2152 20 Mar 1950
John Duncan  2    20 Jan 1966
Larry Jones  3223 20 Dec 1946
```

```
64 sort -k 3 xFile123compact
or cat xFile123compact | sort -k3
```

`cat` can be replaced with `more`

65. The above result is incorrect (why?). Fix the problem by using the utility more effectively.

```
$ your-command
John Duncan 2    20 Jan 1966
John Smith  1222 26 Apr 1956
Lisa Sue    1222 4  Jul 1980
Tony Jones  2152 20 Mar 1950
Larry Jones 3223 20 Dec 1946
```

```
65  sort -n -k3 xFile123compact
or  cat xFile123compact | sort -n -k3
```

66. Sort `xFile123compact` according to the numerical value of the 3rd field, in reverse order

```
Larry Jones 3223 20 Dec 1946
Tony Jones  2152 20 Mar 1950
Lisa Sue    1222 4  Jul 1980
John Smith  1222 26 Apr 1956
John Duncan 2    20 Jan 1966
```

```
66  sort -n -k3 -r xFile123compact
or  cat xFile123compact | sort -n -k3 -r
```

67. [For your information] In the previous two exercises, `John Smith` and `Lisa Sue`, who have the same 3rd field value, did not get further sorted according to the 4th field. The following command sort `xFile123compact` according to the numerical value of the 3rd field, and then based on this, further sort according to the 4th field.

```
$ sort -n -k3 -k4 xFile123compact
John Duncan 2    20 Jan 1966
Lisa Sue    1222 4  Jul 1980
John Smith  1222 26 Apr 1956
Tony Jones  2152 20 Mar 1950
Larry Jones 3223 20 Dec 1946
```

```
67  sort -n -k3 -k4 xFile123compact
or  cat xFile123compact | sort -n -k3 -k4
```

68. Sort `xFile123compact` according to the year (the last field)

```
Larry Jones 3223 20 Dec 1946
Tony Jones  2152 20 Mar 1950
John Smith  1222 26 Apr 1956
John Duncan 2    20 Jan 1966
Lisa Sue    1222 4  Jul 1980
```

cat can be replaced with more

```
68  sort -n -k6 xFile123compact
or  cat xFile123compact | sort -n -k6
```

69. Sort `xFile123compact` according to the year (the last field), in reverse order.

```
Lisa Sue    1222 4  Jul 1980
John Duncan 2    20 Jan 1966
John Smith  1222 26 Apr 1956
Tony Jones  2152 20 Mar 1950
Larry Jones 3223 20 Dec 1946
```

```
69  sort -n -k6 -r xFile123compact
or  cat xfile123compact | sort -nr -k6
```

70. Sort `xFile123compact` according to the 5th field (month)

```
$ sort -k 5 xFile123compact
John Smith  1222 26 Apr 1956
Larry Jones 3223 20 Dec 1946
John Duncan 2    20 Jan 1966
Lisa Sue    1222 4  Jul 1980
Tony Jones  2152 20 Mar 1950
```

```
70  sort -n -k5 xFile123compact
or  cat xfile123compact | sort -n -k5
```

71. In the previous question, month field is not sorted correctly (why?). Fix by using the utility more effectively.

```
$ your_command
John Duncan 2    20 Jan 1966
Tony Jones  2152 20 Mar 1950
John Smith  1222 26 Apr 1956
Lisa Sue    1222 4  Jul 1980
Larry Jones 3223 20 Dec 1946
```

```
71  sort -M -k5 xFile123compact
or  cat xfile123compact | sort -M -k5
```

```
72(1) who | grep yorku.ca
72(2) who | grep yorku.ca | wc -l
72(3) who | grep -v yorku.ca
```

The following exercises involve searching matches in files. Use **egrep** or **grep -E** (which guarantee to accept the extended regular expression) and make sure you are in sh or bash.

72. (1) Use `grep` or `egrep` to get the people who are currently logged on using `yorku.ca` network. (2) find out how many people are currently logged on using `yorku.ca` network (3) Reverse the result, showing who logon using non `yorku.ca` network (so you can see clearly who is logon from the department and who is logon from outside, e.g., from home. Occasionally I use this trick to check if the professors I want to drop by is currently in the office – you'd better hold off going if he is currently logged on from non-York network such as bell or rogers :)

73. Display records of people in file `xFile123compact` who has a field value 2 in the record.

```
$ egrep 2 xFile123compact
 John Duncan 2    20 Jan 1966
 John Smith  1222 26 Apr 1956
 Larry Jones 3223 20 Dec 1946
 Lisa Sue    1222 4  Jul 1980
 Tony Jones  2152 20 Mar 1950
```

```
74 egrep  -w 2 xFile123compact
```

74. The above result is not desirable. Use the utility effectively so that only `John Duncan 2    20 Jan 1966` is displayed. Hint: do a 'whole word' match.

75. Display the records of people in file `xFile123compact` who were born in 1950s. Hint: from the perspective of regular expression, a person's year field is `195.` where . represent any single character.

```
$ egrep
John Smith 1222 26 Apr 1956
Tony Jones 2152 20 Mar 1950
```

```
75   egrep  195.$ xFile123compact
```

76. Get the number of peoples in `xFile123compact` who were born in 1950s. You should get 2.

```
76 egrep  195.$ xFile123compact | wc -l
```

77. The EECS department maintains the records of all the students, staff and faculty members in a file named **passwd**, located under directory **/etc,** one person per line. Issue a command to see the content of the file. For such a long file, `cat` is not a good choice.

```
77  more /etc/passwd   or   cat /etc/passwd  | more
```

```
78  wc -l /etc/passwd  or   cat /etc/passwd | wc -l
```

78. Issue a utility to find out how many people are in the file. You should get about 3782.

79. Find out the number of people with name **Wang** in the file. You should get about 49.

```
79  egrep -w Wang /etc/passwd  | wc -l
or  cat /etc/passwd | egrep -w Wang | wc -l
```

The (modified) cla **8-19/S/2031/classlist**. Each line of the file contains one student information, where the first column is the EECS login id.

80. Get the number o

```
80 wc -l /eecs/dept/course/2018-19/S/2031/classlist   or  cat /eecs/…/classlist | wc -l
```

(by giving the pathname), or, copy the file to your current directory.

81. Retrieve your record from the class list using yo

```
81 egrep yourname  classlist   or  cat classlist | egrep
```

82. (1) Try to get the records of students whose family name is **Li**, using **grep Li classlist**. You will see that the records of those whose family name is **Liu** and **Liang** are also displayed (why?).

(2) Fix (1) by using **grep** more effectively. You should see six lines.

```
82(2) egrep -w Li classlist or cat classlist | egrep -w Li
```

(3) Get the number of students whose family name is **Li**. You should get 6.

```
82(3) egrep -w Li classlist | wc -l or cat classlist | egrep -w Li | wc -l
```

83. (1) Get the number of students whose family name is **Wang**. You should get 4

```
83(1) egrep -w Wang classlist |wc -l or cat classlist | egrep -w Wang |wc
```

(2) Confirm (1) by retrieving the record of students whose family name is **Wang**. You should see four lines

```
83(2) egrep -w Wang classlist  or cat classlist | egrep -w Wang
```

84. (1) Get the number of students whose family name is **Wu**. You should get 3.

```
84(1) egrep -w Wu classlist |wc -l or cat classlist | egrep -w Wu |wc -l
```

(2) Confirm (1) by retrieving the record of students whose family name is **Wu**. You should see three lines.

```
84(2) egrep -w Wu classlist  or cat classlist | egrep -w Wu
```

85. (1) Get the number of students whose family name is **Patel**. You should get 6.

```
85 egrep -w Patel classlist or cat classlist | egrep -w Patel
```

(2) Confirm (1) by retrieving the record of students whose family name is **Patel**. You should see six lines.

86. Get the number of students whose family name is **Wong**. You should get 1.

```
86 egrep -w Wong classlist | wc -l or cat classlist | egrep -w Wong | wc -l
```

87. (1) Get the number of students whose family name is **Wang** or **Wong**. You should get 5.

Hint, from the perspective of regular expression, W[ao]ng or "Wang| Wong" will do the trick.

(2) Confirm (1) by retrieving the record of students whose family name is **Wang** or **Wong**. You should see five lines.

88

```
87(1) egrep W[ao]ng classlist | wc -l   or  cat classlist | egrep W[ao]ng  | wc -l
or  egrep "Wang|Wong" classlist | wc -l or cat classlist | egrep "Wang| Wong" classlist |wc-l

(2) egrep W[ao]ng classlist     or  cat classlist | egrep W[ao]ng
    egrep "Wang|Wong" classlist or  cat classlist | egrep "Wang|Wong"     " " can be  '  '
```

```
cse****          *********       Yu   Ying
cse******        *********       Lee  JunXu
eqao             cse******       Tong Treacy
```

89. The above result is not desirable. Use this utility effectively, so the last line is filtered out.

```
cse****          *********       Yu   Ying
cse******        *********       Lee  JunXu
```

```
89 egrep ^cse classlist
```

90. `cut` is a utility that can extract columns of a text file. By default `cut` treats `tab` as the column delimiter. (We can also specify other delimiters such as space or comma). To specify the columns to extract, use `-f`.

    The `classlist` columns are separated by tab.

    Issue `cut -f 1 classlist` Observe that the only eecs user info (the first column) is displayed.

    Issue `cut -f 3 classlist` Observe that the only surnames (the 3rd column) is displayed.

    Issue `cut -f 1-3 classlist` Observe that columns 1 to 3 are displayed.

    Issue `cut -f 1,3 classlist` Observe that the first and the 3rd column are displayed.

    Issue `cut -f 3,4 classlist > tmp` Observe the 3rd column (surname) and 4th column (given name) are written file `tmp`. Remove file `tmp`.

There is a file **`lyrics`** in directory **`/eecs/dept/course/2018-19/S/2031.`** Find the lines in **`lyrics`** that:

91. contains **`the`**

```
#So turn off the light, 1980
Say all your prayers and then,
Beautiful mermaids will swim through the sea,
And you will be swimming there too.
sea 1980 I got there by chance.
```

```
91 egrep the lyrics
or  cat lyrics | egrep the
```

92. contains **the** as a whole word

```
#So turn off the light, 1980
Beautiful mermaids will swim through the sea,
```

```
92 egrep -w the lyrics
or  cat lyrics | egrep -w the
```

93. does not contain **the** as a whole world

```
Well you know it's your bedtime,
Say all your prayers and then,
Oh you sleepy young 1970 heads dream of wonderful things,
And you will be swimming there too.
sea 1980 I got there by chance.
```

```
93 egrep -w -v the lyrics
or  cat lyrics | egrep -wv the
```

94. contains digits

```
#So turn off the light, 1980
Oh you sleepy young 1970 heads dream of wonderful things,
sea 1980 I got there by chance.
```

```
94 egrep [0-9] lyrics
or  cat lyrics | egrep [0-9]

or egrep [[:digit:]] lyrics
```

95. contains **`1980`**

```
95 egrep 1980 lyrics
or  cat lyrics | egrep 1980
```

```
#So turn off the light, 1980
sea 1980 I got there by chance.
```

96. end with **1980**

```
#So turn off the light, 1980
```

```
96  egrep 1980$ lyrics
or  cat lyrics | egrep 1980$
```

97. contains **sea**

```
Beautiful mermaids will swim through the sea,
sea 1980 I got there by chance.
```

```
97 egrep sea lyrics
or  cat lyrics | egrep sea
```

98. begins with **sea**

```
sea 1980 I got there by chance.
```

```
98 egrep ^sea lyrics
or  cat lyrics | egrep ^sea
```

99. contains one (any) character followed by **nd**

```
Say all your prayers and then,
Oh you sleepy young 1970 heads dream of wonderful things,
And you will be swimming there too.
```

```
99 egrep .nd lyrics
or  cat lyrics | egrep .nd
```

100. contains one (any) character followed by **nd**, but as a whole world only (so `wonderful` does not match)

```
Say all your prayers and then,
And you will be swimming there too.
```

```
100 egrep -w .nd lyrics
or  cat lyrics | egrep -w .nd
```

101. begins with one (any) character followed by **nd**

```
And you will be swimming there too.
```

```
101 egrep  ^.nd lyrics
or  cat lyrics | egrep  ^.nd
```

102. contains letter **A** or **B** or **C** or **D**

```
Beautiful mermaids will swim through the sea,
And you will be swimming there too.
```

```
102 egrep  [ABCD] lyrics
or  cat lyrics | egrep  [ABCD]

or egrep  [A-D] lyrics
or  cat lyrics | egrep  [A-D]
```

103. begins with a capital letter

```
Well you know it's your bedtime,
Say all your prayers and then,
Oh you sleepy young 1970 heads dream of wonderful things,
Beautiful mermaids will swim through the sea,
And you will be swimming there too.
```

```
103 egrep  ^[A-Z] lyrics
or  cat lyrics | egrep  ^[A-Z]
```

```
104 egrep  a.$ lyrics
or  cat lyrics | egrep  a.$
```

104. ends with `a` and one other character.

```
Beautiful mermaids will swim through the sea,
```

105. contains a character that is either **a** or **b** or **c**, followed by **nd**

```
Say all your prayers and then,
```

```
105 egrep  [abc]nd lyrics
or  cat lyrics | egrep  [abc]nd

or  egrep  [a-c]nd lyrics
or  cat lyrics | egrep  [a-c]nd
```

106. contains a character that is not **a** nor **b** nor **c**, followed by **nd**

```
Oh you sleepy young 1970 heads dream of wonderful things,
And you will be swimming there too.
```

```
106 egrep  [^abc]nd lyrics
or  cat lyrics | egrep  [^abc]nd

or  egrep  [^a-c]nd lyrics
or  cat lyrics | egrep  [^a-c]nd
```

107. Go back to the parent directory

```
cd ..
```

108. Issue utility

```
find  .  -name "xFile?"
```

```
108
$ find . -name "xFile?"
./lab8working/xFile2
./lab8working/xFile1
./lab8working/xFile3
$
```

What did you get?

```
109
$ find . -name "xFile*"
./lab8working/xFile123
./lab8working/xFile2
./lab8working/xFile123compact
./lab8working/xFile1
./lab8working/xFile3
$
```

109. Now issue the utility

```
find  .  -name "xFile*"
```

What did you get?

110. Now issue

```
find  . -name "xFile*"  -exec mv {}  {}.Lab8 \;
```

```
110 Find all the files whose name
begins with xFile, and change name of
them.
Each new name now has a .Lab8 suffix.
```

Issue `ls lab8working` or `ls -l -R` to examine what happens to the files in `lab8working`. Now issue

```
find  . -name "xFile*"  -exec chmod 775 {} \;
```

```
110 Find all the files whose name
begins with xFile, and change
permission to   rwxrwxr-x
```

What do we intend to do here?

Issue `ls -l lab8working` or `ls -l -R` to examine what happens to the files in `lab8working`

```
Note: for solutions using egrep
•  all egrep can be replaced with grep -E
•  all the unquoted search patterns can be quoted
   e.g.,  egrep the lyrics is same as egrep "the" lyrics  or egrep 'the' lyrics
          egrep ^.nd lyrics is same as egrep "^.nd" lyrics  or egrep '^.nd' lyrics

   Actually it is a good habit to always quote search patterns
```

Part II Common shell functionalities and corresponding meta-characters

In class we also discussed some functionalities that are common among the different shells, and their associated meta-characters. In part I above we have experienced some of them, for example, **Pipes** |, **Filename substitution (wildcards)** * ? [], **Redirections** < > >>. Here you practice some more functionalities, including **Command substitution** ``, **Variable substitution** $, **Conditional sequence** && ||, and **Quotes** '' and " ".

111. **Filename substitution (Wild-cards * ? []).**

Navigate to working directory.

- Issue `ls *` Observe that all files in the directory are listed
- Issue `ls xFile*.Lab8` Observe that all files whose name begins with `xFile` are listed
- Issue `ls xFile?.Lab8` Observe that files `xFile1.Lab8`, `xFile2.Lab8`, `xFile3.Lab8` (but not `xFile123.Lab8 and xFile123compact.Lab8)` are listed (why?)
- Issue `ls xFile???.Lab8` Observe that only file `xFile123.Lab8` is listed (why?)
- Issue `ls xFile[1,3].Lab8` Observe that files `xFile1.Lab8` and `xFile3.Lab8` are listed (why?).
- Issue `ls xFile[1-3].Lab8` Observe that files `xFile1.Lab8, xFile2.Lab8` and `xFile3.Lab8` are listed.

112. **Command substitution** ``

- Issue a single command to output current date and time, where time and date info comes from utility `date`.
  `Hello, now is Fri Jul 19 09:13:05 EDT 2019. Have a good day`
- Issue a single command to output `There are 135 students in EECS2031A` where `135` comes from the result of a command that reads from file `classlist`.

```
112  you can also add double quote " " around the messages.
$ echo Hello, now is `date`. Have a good day!   Or  echo "Hello, now is `date`. Have a good day!"

$ echo There are `wc -l classlist` students in EECS2031A. or There are `cat classlist | wc -l` students

$ echo There are `egrep -w Wang classlist | wc -l` students in EECS2031A with family name Whang

$ echo There are `egrep ^cse  classlist | wc -l` students in EECS2031A whose eecs username begins with cse.
```

- Issue a single command to output `There are 4 students in EECS2031A with family name Wang` where `4` comes from the result of a command that reads from file `classlist`.
- Issue a single command to output `There are 2 students in EECS2031A whose eecs username begins with cse` where `2` comes from the result of a command that reads from file `classlist`.

113. **Conditional sequence && ||.** 1) For a series of commands separated by "&&" tokens, the next command is executed only if the previous command returns an exit code of 0, which means 'successful'. 2) For a series of commands separated by "||" tokens, the next command is executed only if the previous command returns a non-zero exit code, which means 'unsuccessful'.

- Issue `egrep -w Leung classlist` and then `echo $?` to examine the exit code 1 which means unsuccessful (no matching found).
- Issue `egrep -w Wang classlist` , and then `echo $?` to examine the exit code 0 which means matching found.
- Issue `egrep -w Leung classlist && echo HELLO`, observe that HELLO is not printed (why?).
- Issue `egrep -w Wang classlist && echo HELLO`, observe that HELLO is printed (why?).
- Issue `egrep -w Leung classlist || echo HELLO`, observe that HELLO is printed (why?).
- Issue `egrep -w Wang classlist || echo HELLO`, observe that HELLO is not printed (why?).

114. There are often times when you want to inhibit the shell's **filename-substitution** (wild-card) * ? [], **variable-substitution $**, and/or **command-substitution** ``` mechanisms. The shell's quoting system allows you to do just that. The way that it works is:

1) Single quotes (' ') inhibits both wildcard substitution, variable substitution, and command substitution.

2) Double quotes(" ") inhibits wildcard substitution only.

- Issue `courseN=EECS2031A;` (no space around =) This assign variable `courseN` with value `EECS2031A`
  Then issue `echo  3  * 4  = 12,  course name is $courseN – today's date is `date``
  Observe that both filename-substitution (wildcard) `*`, variable-substitution $n, and command-substitution `date` are interpreted. Among them the wildcard-substitution is interpreted as 'any file name'.
- Then issue `echo  ' 3  * 4  = 12,  course name is $courseN – today's date is `date` '`
  Observe that interpretation of filename-substitution (wildcard) `*` is inhibited. Interpretation of variable-substitution `$n` and command substitution `` `date` `` are also inhibited, due to the fact that single quote ' ' inhibited the interpretation of both the three substitutions.
- Finally, issue `echo  " 3  * 4  = 12,  course name is $courseN – today's date is `date` "`

Observe that interpretation of `*` is inhibited. Interpretation of variable-substitution `$n` and `` `date` `` are not inhibited, due to the fact that double quote " " inhibits the interpretation of filename-substitution (wild-card) * only.

Part III Bourne (again) shell scripts

In this exercise you will be playing with Bourne shell scripts that can solve some problems. Bourne again shell (bash) contains enhancements to Bourne shell and is compatible with Bourne shell. Thus Bourne shell scripts can also be considered Bourne again shell scripts.
Note: our lab provides Bourne Again shell (bash) environment. If you know bash specific syntaxes, feel free to use them. Also, run the scripts under bash (issue `sh` or `bash`).

The purpose of this exercise is to help you gain better understanding about shell script concepts including variable assignment, reading and access, variable input and output, branches and loops etc.

1. Read user input, and do logical comparisons.

In bourn shell, you read user input using utility **`read`**. Navigate to directory `lab8working`. Issue

```
sh-4.2$ read x

We are the world and children

sh-4.2$ echo $x
```

Observe that **`x`**'s value is the whole input line. Now issue

```
sh-4.2$ read x y z

We are the world and children

sh-4.2$ echo $x, $y, $z
```

Observe that **`x`** and **`y`** get the first and 2nd token in the input, and **`z`** gets the rest of input.

In Bourn shell, evaluate conditional expression using **`test expression`** or **`[ expression ]`**. Issue

```
sh-4.2$ test 3 -lt 4

sh-4.2$ echo $?
```

The above evaluates if 3 is less than 4. Observe the exit code 0, which means true in Unix. Issue

```
sh-4.2$ num=3
sh-4.2$ [  $num  -gt  4  ]
sh-4.2$ echo $?
```

No spaces around **=**

Spaces after **[** and before **]**

The above evaluates if value of variable **`num`** is greater than 4. Observe the exit code 1, which means false. Now issue

```
sh-4.2$ input="quit"
sh-4.2$ [ $input = "quitx" ];   echo $?
```

The above tests if variable **input** equals to string "quitx", and then displays the exit code. Recall that `;` is a shell meta-character that is used to connect a sequence of commands. Now issue

```
sh-4.2$ [ $input = "quit" ];    echo $?
```

Observe the exit code 0 of the above. Now issue

```
sh-4.2$  [ -f  xFile123.Lab8 ];  echo $?
```

The above tests if **xFile123.Lab8** is a regular file, and then displays the exit code 0. Now issue

```
sh-4.2$  [ -x  xFile123.Lab8 ];  echo $?
```

The above tests if **xFile123.Lab8** is an executable file. Now issue

```
sh-4.2$  [ -d  ../lab8working ];  echo $?
```

The above tests if **../lab8working** is a directory.

2. Problem B

Copy the classlist file from **/eecs/dept/course/2018-19/S/2031/classlist** to your current working directory, and do the following exercise.

Download and run the provided shell script **mygrep.sh**. Note that the file should have execute permission for the owner. Set the permission if it does not have.

Try to understand the code of the script. And then issue

```
sh-4.2$ mygrep.sh
Please enter file to search: classlist
Please enter search key:  Wang
```

Observe the output. Then run again, enter search key **Leung**, observe the result.

3. Problem C

Download and run the provided shell script **mygrepArg.sh**. Make sure that this file has execute permission for the owner. Set it if it doesn't.

Try to understand the code of the script. The program, which takes (at least) two command line arguments,

- first checks if less than two command line arguments are given (how to check?). If yes, then outputs `Error! usage:  ./mygrepArg.sh  filename pattern`

  Note that `./mygrepArg.sh` is not hard coded
- if at least two command line arguments are entered, then checks if the first argument represents an existing file in the current directory. If the file does not exist, outputs `Error! "filename" is not an existing file in the current directory` where `filename` is the first command line argument
- if the filename represents an existing file, then displays the two command-line arguments using both `$@` and `$*`.

Then conducts the search by invoking `grep $2 $1` (what is `$2` and `$1`)

Run the script with the following inputs, and observe the output

```
sh-4.2$ mygrepArg.sh
   Error! usage: ./mygrepArg.sh filename pattern
sh-4.2$ mygrepArg.sh classlistX
   Error! usage: ./mygrepArg.sh filename pattern
sh-4.2$ mygrepArg.sh classlistX Wang
  Error! "classlistX" is not an existing file in current directory
sh-4.2$ mygrepArg.sh classlist Wang
There are 2 command line arguments: classlist Wang or classlist Wang
e*andrew        *********       Wang  Andrew Jr-Yu
haoqwang        *********       Wang  Haoqiu
matt***         *********       Wang  Matthew
weihang         *********       Wang  Wei
sh-4.2$ mygrepArg.sh classlist Leung
There are 2 command line arguments: classlist Leung or classlist Leung
```

4. Problem D

In problem C, the script just executes `grep`, letting the "raw" result of `grep` be displayed, which contains either lines of search result, or just nothing.

Enhance the script for problem C in such a way that

- If the search pattern is not found in file, then instead of displaying nothing, outputs `Pattern "pattern" was not found in file "filename"` where `pattern` and `filename` are the 2nd and 1st arguments respectively .

```
#! /bin/sh
if test $# -lt 2
then
  echo " Error! Usage: $0 filename pattern"
elif [ ! -f $1 ]
then
  echo " Error! \"$1\" is not an exsiting file"
else
  echo  there are $# command line arguments: $@  or $*
  grep -w $2 $1
  if [ $? -ne 0 ]
  then
     echo Pattern \"$2\" was not found in file \"$1\"
  fi
fi
```

Sample Inputs/Outputs:

```
sh-4.2$ mygrepArg.sh classlist Leung
There are 2 command line arguments: classlist Leung or classlist Leung
Pattern "Leung" was not found in file "classlist"
```

For interested students: sometimes we may want to turn off the raw outputs from the utility call. In order to do this, change the line of the utility call to `grep $2 $1 > /dev/null`
This redirects the outputs of `grep` **to a 'black hole' at** `/dev/null`, **where the outputs are absorbed.**

```
#! /bin/sh
if test $# -lt 2
then
  echo " Error! Usage: $0 filename pattern"
elif [ ! -f $1 ]
then
  echo " Error! \"$1\" is not an exsiting file"
else
  echo  there are $# command line arguments: $@  or $*
  grep -w $2 $1 > /dev/null
  if [ $? -ne 0 ]
  then
     echo Pattern \"$2\" was not found in file \"$1\"
  else
     echo Pattern \"$2\" was found in file \"$1\"
  fi
fi
```

5. Problem E

Variable reading + Branching + Looping in Bourne (again) shell
Download the script `numbers.sh`. Try to understand the code. Then run the script, observe the output.

Sample Inputs/Outputs

```
sh-4.2$ numbers.sh
Enter a number or 'quit': 22
  22 is a positive number
Enter a number or 'quit': 33
  33 is a positive number
Enter a number or 'quit': -1
  -1 is a negative number
Enter a number or 'quit': 0
  0 is zero
Enter a number or 'quit': -13
  -13 is a negative number
Enter a number or 'quit': quit
  Bye bye
sh-4.2$
```