LAB 8 Unix Utilities and common functionalities

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 choice. We have covered the following basic utilities/commands: **man**, **pwd**, **ls**, **cd**, **mkdir**, **rmdir**, **cat**, **more**, **less**, **head**, **tail**, **cp**, **mv**, **rm**, **wc**, **file**, **chmod**, **chgrp**. We also discussed "pipe", which allows the output of one utility to be used as the input of another utility. We also covered "advanced" utilities/commands **grep/egrep**, **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 about 85 (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 -l`).**

0. Login to your prism lab 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 (using **cd**). **Note: you are encouraged to work in the lab environment (by ssh to red.cse.yorku.ca). If you prefer to work on your local computer, see instructions at the end of this lab manual.**

1. There is a file named `xxx` in directory **/eecs/dept/course/2019-20/W/2030tmp/**

   In your terminal, issue one of the following commands to copy this file to your current working directory.

   ```
   $ cp /eecs/dept/course/2019-20/W/2030tmp/xxx .
   $ cp /eecs/dept/course/2019-20/W/2030tmp/xxx ./
   $ cp /eecs/dept/course/2019-20/W/2030tmp/xxx ./xxx
   ```

2. Verify that the file is copied into the current working directory, by listing the content of your working directory.

   ```
   $ ls
   xxx
   ```

   ```
   2  ls
   or ls  ./
   ```

3. There are two files named `xFile2` and `xFile3` in same directory **/eecs/dept/course/2019-20/W/2030tmp/**

   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. (* and ? are <u>filename substitution wildcards</u>. 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
   xFile2  xFile3  xxx
   ```

   ```
   3  cp /eecs/dept/course/2019-20/W/2030tmp/xFile2 eecs…/xFile3   .
   or cp /eecs/dept/course/2019-20/W/2030tmp/xFile?  .
   or cp /eecs/dept/course/2019-20/W/2030tmp/xFile*  .
   or cp /eecs/dept/course/2019-20/W/2030tmp/xFile[23] .
   ```

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

5. Rename file `xxx` to `xFile1`

```
5 mv xxx  xFile1   or   mv ./xxx  ./xFile1
```

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

   One (professional) way to verify if the execution of a utility is successful is to examine the exit code (aka, return value) of the execution process. The exit code is a integer number ≥ 0, and 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

```
6(1) echo $?
```

```
$ ls
xFile1 xFile2 xFile3
```

```
6(2) ls ./ or ls xFile1 xFile2 xFile3  or ls xFile*  or  ls xFile?
```

7. (1) Create a sub-directory named `2020F` under your current working directory. (2) Then still in the current working directory, create a subdirectory `lab7a` under `2020F`.

8. Verify that the two directories are created successfully, by recursively listing directory `2020W` and its contents.

```
$ ls -R 2020F
2020F:
Lab7a

2020F/lab7a:
```

```
8  ls -R 2020F
```

```
7(1) mkdir 2020F   or    mkdir ./2020F
 (2) mkdir 2020F/lab7a or mkdir ./2020F/lab7a
```

```
9   mv xFile1  2020F/lab7a
or  mv xFile1  2019F/lab7a/xFile1
```

9. Move `xFile1` into subdirectory `lab7a` (with same name), using relative path.

10. Then move all the other 2 files (together) into `lab7a` (using relative path), using one entry of utility)

11. Verify that the above moving were successful, by recursively listing directory `2020W` and its contents.

```
$ ls -l -R 2020F
./2020F:
total 4
drwx------ 2 yourname ugrad 4096 Nov 25 15:12 lab7a

./2020F/lab7a:
total 12
-rwx------ 1 yourname ugrad 145 Nov  25 15:11 xFile1
-rwx------ 1 yourname ugrad 145 Nov  25 15:11 xFile2
-rwx------ 1 yourname ugrad  87 Nov  25 15:11 xFile3
```

```
10  mv xFile*    2020F/lab7a
or  mv xFile?    2020F/lab7a
or  mv xFile[23] 2020F/lab7a
```

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

12. (1) Navigate to subdirectory `2020F` and (2) Confirm you are in `2020F` now, using **`pwd`**.

```
$ cd 2020F
$ pwd
/cs/home/your_account/…/2020F
```

```
12 (1) cd 2020F  or  cd ./2020F
   (2) pwd
```

13. (1) List the files in subdirectory `lab7a`.

```
$ ls -l lab7a
total 12
-rwx------ 1 yourname ugrad 145 Nov 25 15:11 xFile1
-rwx------ 1 yourname ugrad 145 Nov 25 15:11 xFile2
-rwx------ 1 yourname ugrad  87 Nov 25 15:11 xFile3
```

```
13 (1) ls  -l lab7a
    or ls  -l ./lab7a
```

(2) Then list the information of subdirectory `lab7a` itself

```
13 (2) ls  -ld   lab7a
or     ls -l -d  lab7a
```

$ **your-command**

```
drwx------ 2 yourname ugrad 48 Nov 25 15:12 lab7a
```

14. Copy directory `lab7a` to a new directory named `lab7b`, under same directory `2020F` (using one utility).

```
14  cp -r  lab7a   lab7b
or  cp -r ./lab7a  ./lab7b
```

15. Verify that `lab7b` is created, and contains the same file entries as `lab7a`

$ ls -l *

```
Lab7a:
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

Lab7b:
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
```

```
15  ls -l *
or  ls lab7a lab7b
or  ls lab7?
or  ls lab7*
or  ls lab7[ab]
```

16. Remove the whole directory `lab7a` using `rmdir`. What happened?

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

```
17   echo $?
```

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

18. Remove the whole directory `lab7a` using a more effective utility.

```
18   rm -r lab7a
  or rm -r ./lab7a
```

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

```
19(1)  echo $?
```

(2) Verify by trying to list `lab7a`

$ **ls lab7a**

```
ls: cannot access lab7a: No such file or directory
```

```
19(2) ls lab7a
or    ls -l lab7a
or    ls -ld lab7a
```

20. Move `xFile1`, which is in subdirectory `lab7b`, to current (parent) directory, using relative pathname.

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

$ **find . -name "xFile*"** or **find . -name "xFile?"**

```
./lab7b/xFile2
./lab7b/xFile3
./xFile1
```

```
20 mv  lab7b/xFile1  .
or mv  lab7b/xFile1  ./xFile1
or mv  lab7b/xFile1  xFile1
```

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

" " can be ' '

22. Change the name of directory `lab7b` to `lab7working`

```
22  mv lab7b lab7working
```

23. (1) Navigate to directory `lab7working`

```
23 (1) cd lab7working  or  cd ./lab7working
```

(2) Verify that you are in `lab7working`

```
23(2) pwd
```

$ **your-command**

```
/cs/home/your_account/…/2020W/lab7working
```

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

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

25. Verify that the moving was successful by listing all the files currently in `lab7working`

```
$ your_command
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
```

```
25  ls -l
```

26. Issue the following command. Observe that `cat` reads a line of input from stdin and prints to stdout, until EOF.

```
$ cat
Hi
Hi
There
There
^D (press Ctrl and D)
$
```

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

```
$ cat > temp
Hi
There
^D (press Ctrl and D)
$
```

```
27 (2) ls
   (3) cat temp  or cat < temp
```

(2) List the current directory to confirm that file `temp` is created.

(3) View the content of file `temp` by using `cat` again.

```
29 cat  xFile1   or  cat < xFile1
or more xFile1   or  more < xFile1
or less  xFile1  or  less < xFile1
```

28. Remove file `temp`

```
28 rm temp
```

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

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

30. 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.) Observe that `xFile1` and `xFile2` have the same content.

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

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

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

```
32  head -2 xFile1
or  cat xFile1 | head -2  or  more xFile1 | head -2
```

33. Display the last 3 lines of `xFile2`

```
33  tail -3 xFile2
or  cat xFile2 | tail -3  or  more xFile2 | tail -3
```

34. (1) Confirm that `xFile1` and `xFile2` have identical content, using a utility, which should return silently (Hint: `cmp` or `diff`). (2) Examine the exit code, you should get 0

```
34/35(1)  cmp xFile1  xFile2
     (2)  echo $?
```

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

```
34/35(1)  diff xFile1  xFile2
     (2)  echo $?
```

36. (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.

37. (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 :

```
36(1) diff xFile2  xFile3
  (2) echo $?
```

```
37(1) cmp xFile2  xFile3
  (2) echo $?
```

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

This program gets 0 mark if the last command prints 1, which indicates that `yourOutputFile` and `professorsOutputFile` are not exactly identical. ☹

38. (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
```

```
38 (1) cat xFile1 xFile2  xFile3 > xFile123

   (2) cat xFile123    or   more  xFile123
```

39. Sort lines in file `xFile123` (in lexicographic order), 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
```

```
39  sort xFile123
or  cat  xFile123  |  sort
or  more xFile123  |  sort
or  less xFile123  |  sort
```

```
40  sort xFile123  |  uniq
or  cat xFile123   |  sort | uniq
```

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

```
$ 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
```

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

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

42. Show on the `stdout` the content of `xFile123compact`. You should get same output as in question 40.

```
42  cat xFile123compact
or  more xFile123compact
```

```
43  ls -l xFile1
    cat: xFile1: Permission denied
```

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

44. Issue `chmod 775 xFile1` and then examine the resulting permission mode c should get `-rwxrwxr-x` Can you understand what we are doing here?

```
44  chmod 775 xFile1
    ls -l xFile1
```

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

```
-rwxr-xrwx  1 yourname ugrad 145 Nov 25 17:23 xFile123compact
```

```
45 chmod g-w,o+rw xFile123compact
```

46. 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 (press Ctrl and D)
```

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

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

Then view the content of `xFile1` by using `cat`.

47. Remove the `write` permission of the `owner` of `xFile1`, and try 46 again. What do you get?

Question 48-49 should be done without using `sort`. Utility `ls` can do some sorting itself.

48. (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, `xFile123compact` is the 2nd, and other files are also sorted according to the modification time.

```
48(1) ls -l -t   or   ls  -lt
```

```
$ 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
……
```

```
48(2) ls -l -t -r    or   ls  -ltr
```

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

49. (1) List the files, sorted by the size of the files. By default, "largest first", so `xFile123` should be the first file in the list and other files are also sorted according to their sizes.

49(1) `ls -l -S` or `ls -lS`

```
$ your_command
total 20
-rw-r--r-- 1 yourname ugrad 377 Mar 25 13:35 xFile123
-r-xrwxrwx 1 yourname ugrad 168 Mar 25 13:42 xFile1
-rwxr-x-wx 1 yourname ugrad 145 Mar 25 13:37 xFile123compact
-rwx------ 1 yourname ugrad 145 Mar 25 13:27 xFile2
-rwx------ 1 yourname ugrad 87  Mar 25 13:27 xFile3
```

49(2) `ls -l -S -r` or `ls -lSr`

(2) List the files, sorted by the size of the files, in reverse order. The above list should be reversed.

50. 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
```

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

`cat` can be replaced with `more` or `less`

51. 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
```

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

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

`cat` can be replaced with `more`

```
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
```

52(1) `sort -n -k6 xFile123compact`
or `cat xFile123compact | sort -n -k6`

(2) 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
```

52(2) `sort -n -k6 -r xFile123compact`
or `cat xfile123compact | sort -nr -k6`

53. 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
```

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

54. 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
```

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

55. 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
```

56. 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.

```
56 egrep  -w 2 xFile123compact
```

57. 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
```

```
57   egrep  195.$ xFile123compact
```

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

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

The (modified) class list of our class can be found at **`/eecs/dept/course/2019-20/W/2030tmp/classlist`**. Each line of the file contains one student information, starting with EECS username, followed by student id (hidden), surname and given name. Copy the file to your working directory, view the content of the file. For such a long file, `cat` is not a good choice.

59. Get the number of students currently enrolled in the course. You should get `153`.

60. Retrieve your record from the class list using your family name. Does anyone else has the same family name as?

```
59 wc -l classlist   or
 cat /eecs/…/classlist | wc -l
```

```
60 egrep yourname  classlist   or  cat classlist | egrep
```

61. (1) Try to get the records of students whose family name is **Li**, using **egrep Li classlist**. You will see that the records of those who

```
61 egrep -w Li classlist  or cat classlist | egrep -w Li
```

(2) Fix (1) by using **egrep** more effectively. You should see three lines.

```
62(1) egrep -w Wang classlist | wc -l or cat classlist | egrep -w Wang | wc -l
```

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

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

```
62(2) egrep -w Wang classlist  or cat classlist | egrep -w Wang
```

63. (1) Get the number of students whose family name is **Kim**. You should get 2.

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

```
63(1) egrep -w Kim classlist |wc -l or cat classlist | egrep -w Kim | wc -l
```

64. Get the number of students whose famil

```
63(2) egrep -w Kim classlist or cat classlist | egrep -w Kim
```

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

65. (1) Get the number of students whose family name is **Wang** or **Wong**. You should get 7.

Hint,

(2) Co

(3) Re

```
65(1) egrep -w W[ao]ng classlist | wc -l   or  cat classlist | egrep -w W[ao]ng  | wc -l
or    egrep -w "Wang|Wong" classlist | wc -l or cat classlist | egrep -w "Wang| Wong" |wc-l

(2) egrep -w W[ao]ng classlist     or  cat classlist | egrep -w W[ao]ng
    egrep -w "Wang|Wong" classlist or  cat classlist | egrep -w "Wang|Wong"  note: " " can be  '  '
(3) egrep -w Ch[ae]n classlist | wc -l   or  cat classlist | egrep Ch[ae]n  | wc -l   or
    egrep  -w "Chan|Chen' classlist | wc -l   or  cat classlist | egrep -w "Chan|Chen"  | wc -l
(4) egrep -w L[ai]u classlist | wc -l   or  cat classlist | egrep -w L[ai]u  | wc -l
    egrep  -w "Liu|Lau' classlist | wc -l   or  cat classlist | egrep -w "Liu|Lau"  | wc -l
```

66. `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`. View the content of `tmp` to confirm the results.
- Issue `cat classlist | sort -k 3 | cut -f 3,4` This pipeline of commands sorts the file based on surnames, and then extracts the surname and given name columns.

There is a file **lyrics** in directory **/eecs/dept/course/2019-20/W/2030tmp.** Find the lines in **lyrics** that:

67. contains **the**

```
#So turn off the light, 1980
Say all your prayers and then,
Beautiful mermaids will swim through the sea,
```

```
67 egrep the lyrics
or  cat lyrics | egrep the
```

```
And you will be swimming there too.
sea 1980 I got there by chance.
```

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

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

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

69. 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.
```

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

70. 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.
```

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

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

71. contains **1980**

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

```
71 egrep 1980 lyrics
or  cat lyrics | egrep 1980
```

72. end with **1980**

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

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

73. contains **sea**

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

```
73 egrep sea lyrics
or  cat lyrics | egrep sea
```

74. begins with **sea**

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

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

75. begins with one (any) character followed by **nd**, as a whole world

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

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

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

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

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

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

77. Go back to the parent directory `2020W`

```
cd ..
```

78. Issue utility

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

What did you get?

```
78
$ find . -name "xFile?"
./lab7working/xFile2
./lab7working/xFile1
./lab7working/xFile3
$
```

79. Now issue the utility

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

What did you get?

```
79
$ find . -name "xFile*"
./lab7working/xFile123
./lab7working/xFile2
./lab7working/xFile123compact
./lab7working/xFile1
./lab7working/xFile3
$
```

80. (1) Now issue `find . -name "xFile*" -exec mv {} {}.Lab7 \;` What do we intend to do here?

(2) Now issue `ls lab7working` or `ls -l -R` to examine what happens to the files in `lab7working`.

(3) Now issue `find . -name "xFile*" -exec chmod 775 {} \;` What do we intend to do here?

(4) Now Issue `ls -l lab7working` or `ls -l -R` to examine what happens to the files in `lab7working`

80 (1) Find all the files whose name begins with xFile, and change name of them.
Each new name now has a .Lab8 suffix.

80 (2) Find all the files whose name begins with xFile, and change permission to rwxrwxr-x

Part II Common shell functionalities and corresponding meta-characters

In class we also discuss some functionalities that are common among the different shells, and their associated meta-characters. In part I abo
* ? [], **Redirections** < > >
`, **Variable substitution**

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

81. **Filename subst**

Navigate to your working directory.

- Issue `ls *` Observe that all files in the working directory are listed
- Issue `ls xFile*.Lab7` Observe that all files whose name begins with `xFile` are listed
- Issue `ls xFile?.Lab7` Observe that files `xFile1.Lab7`, `xFile2.Lab7`, `xFile3.Lab7` are listed (but not `xFile123.Lab7` and `xFile123compact.Lab7`) (why?)
- Issue `ls xFile???.Lab7` Observe that only file `xFile123.Lab7` is listed (why?)
- Issue `ls xFile[1,3].Lab7` Observe that files `xFile1.Lab7` and `xFile3.Lab7` are listed (why?).
- Issue `ls xFile[1-3].Lab7` Observe that files `xFile1.Lab7`, `xFile2.Lab7` and `xFile3.Lab7` are listed.
- Issue `wc -l xFile?.Lab7` Observe the results
- Issue `wc -l xFile???.Lab7` Observe the results
- Issue `wc -l xFile[1,3].Lab7` Observe the results
- Issue `wc -l xFile[1-3].Lab7` Observe the results

82. **Command substitution** `` ` ` `` or **`$()` in bash**

- Issue a single command to output the following message, where time and date info comes from utility `date`.
  `Hello, now is Fri Mar 27 13:05:54 EDT 2020. Have a good day`
- Issue a single command to output `There are 153 students in EECS2031A 2020F`
  where `153` comes from the result of a command that reads from file `classlist`.
- Issue a single command to output `There are 3 students in EECS2031A with family name Nguyen`
  where `3` comes from the result of a command that reads from file `classlist`.

82 you can also add double quote " " around the messages. Cannot use single quote
$ echo Hello, now is `date`. Have a good day! or echo "Hello, now is `date`. Have a good day!"
Or 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 2020F. or There are `cat classlist | wc -l` students
Or echo There are $(wc -l classlist) students in EECS2031A 2020F. or There are $(cat classlist | wc -l) students in EECS2031A 2020F.

```
$ echo There are `grep -w Nguyen classlist | wc -l` students in EECS2031A with family name Nguyen or
  echo There are `cat classlist | grep -w Nguyen | wc -l` students in EECS2031A with family name Nguyen or
  echo There are $(grep -w Nguyen classlist | wc -l) students in EECS2031A with family name Nguyen or
  echo There are $(cat classlist | grep -w Nguyen | wc -l) students in EECS2031A with family name Nguyen
```

83. **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 Zhang 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 Zhang 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 Zhang classlist || echo HELLO`, observe that HELLO is not printed (why?).

84. 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:

- ❖ Single quotes (' ') inhibits both wildcard substitution, variable substitution, and command substitution.
- ❖ Double quotes(" ") inhibits wildcard substitution only.
- Issue `courseN=EECS2031M;` (no space around =) This assign variable `courseN` with value `EECS2031M`
  Then issue `echo 3 * 4 = 12, course name is $courseN - today is `date`, bye`
  Observe that both filename-substitution (wildcard) `*`, variable-substitution `$courseN`, and command-substitution `` `date` `` are interpreted. The wildcard-substitution `*` is interpreted as 'any file name'.
- Then issue `echo ' 3 * 4 = 12, course name is $courseN - today is `date`, bye '`
  Observe that interpretation of filename-substitution (wildcard) `*` is inhibited. Interpretation of variable-substitution `$courseN` and command substitution `` `date` `` are also inhibited, due to the fact that single quote `' '` inhibits the interpretation of both the three substitutions.
- Finally, issue `echo " 3 * 4 = 12, course name is $courseN - today is `date`, bye "`
  Observe that interpretation of `*` is inhibited. Interpretation of variable-substitution `$courseN` and `` `date` `` are not inhibited, due to the fact that double quote " " inhibits the interpretation of filename-substitution (wild-card) * only.

End of lab7, finally