LAB 7 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 will also cover "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 80 (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 (remote lab or 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
   ```

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
   ```

5. Rename file `xxx` to `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

```
$ ls
xFile1 xFile2 xFile3
```

7. (1) Create a sub-directory named `2021F` under your current working directory. (2) Then still in the current working directory, create a subdirectory `lab7a` under `2021F`.

8. Verify that the two directories are created successfully, by recursively listing directory `2021F` and its contents.

```
$ ls -R 2021F
2021F:
lab7a

2021F/lab7a:
```

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 was successful, by recursively listing directory `2021F` and its contents.

```
$ ls -l -R 2021F
./2021F:
total 0
drwx------ 2 yourname ugrad 48 Nov 25 15:12 lab7a

./2021F/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
```

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 `2021F` and (2) Confirm you are in `2021F` now, using **`pwd`**.

```
$ cd 2021F
$ pwd
/cs/home/your_account/…/2021F
```

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
```

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

```
$ 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 `2021F` (using one utility).
15. Verify that `lab7b` is created, and contains the same file entries as `lab7a`

```
$ ls -l *
Lab7a:
total 12
-rwx------ 1 yourname ugrad  145 Nov 25 23:32 xFile1
-rwx------ 1 yourname ugrad  145 Nov 25 23:50 xFile2
-rwx------ 1 yourname ugrad  87  Nov 25 23:50 xFile3

Lab7b:
total 12
-rwx------ 1 yourname ugrad  145 Nov 25 23:32 xFile1
-rwx------ 1 yourname ugrad  145 Nov 25 23:32 xFile2
-rwx------ 1 yourname ugrad  87  Nov 25 23:32 xFile3
```

16. Remove the whole directory `lab7a` using `rmdir`. What happened?
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.
19. (1) Verify the exit code of above execution, you should get 0 now

    (2) Verify by trying to list `lab7a`

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

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
```

22. Change the name of directory `lab7b` to `lab7working`
23. (1) Navigate to directory `lab7working`

    (2) Verify that you are in `lab7working`

```
$ your-command
/cs/home/your_account/…/2021F/lab7working
```

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 Nov 25 16:58 xFile1
-rwx------ 1 yourname ugrad  145 Nov 25 16:58 xFile2
-rwx------ 1 yourname ugrad  87  Nov 25 16:58 xFile3
```

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)
$
```

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

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

28. Remove file `temp`
29. Display on stdout the contents of file `xFile1`
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.
32. Display (only) the first two lines of `xFile1`
33. Display the last 3 lines of `xFile2`
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
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.
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 earlier in some courses to do automated grading of your lab or labtest:

```
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
$ cat 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
```

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
```

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. 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.
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 of the file. What do you get? You should get `-rwxrwxr-x` Can you understand what we are doing here?
45. Change the permission of `xFile1` 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 xFile1
    ```

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)`

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

47. Remove the `write` permission of the `owner` of `xFile1`, and try 46 again. What do you get?
48. There is a html file named `Hello.html` in directory **`/eecs/dept/course/2019-20/W/2030tmp`**. Copy this file to the **www** directory of your home account in prism lab. Make sure directory **www** has the `read` and `execute` permission for `others` (how to check?). Open `Hello.html` using a text editor and add your name in place of xxx. Then add `read` permission to `others` of the file (how?). Then, in your browser, enter into address bar the URL, www.eecs.yorku.ca/~abcd/Hello.html, where `abcd` is your EECS user name. You should be able to see the content of the html file. Now share the URL with your friends over the world and they should be able to see the content too. Finally, change the permission of the html file by removing the `read` permission from `others`, and refresh your browser, and you should get *Forbidden* error. Your friends should get the same error.

Question 49-50 should be done without using `sort`. Utility `ls` can do some sorting itself.

49. (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.

    `$` **`your_command`**
    `total 20`

```
-r-xrwxrwx 1 yourname ugrad 166 Nov 25 14:20 xFile1
-rwx------ 1 yourname ugrad 145 Nov 25 14:12 xFile123compact
-rwx------ 1 yourname ugrad 377 Nov 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.

50. (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.

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

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

51. 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
```

52. 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
```

53. (1) 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
```

(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
```

54. 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
```

55. 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
```

56. 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
```

57. The above result is not desirable. Use the utility effectively so that only `John Duncan 2    20 Jan 1966` is displayed.

58. 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
```

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

The (modified) class list of our class can be found at **/eecs/dept/course/2019-20/W/2030tmp/ClassListACadded**. 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.

60. Get the number of students currently enrolled in the course. You should get `194`.

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

62. (1) Try to get the records of students whose family name is **Li**, using **egrep Li classListSu**. You will see that the records of those whose family name is **Liu** and **Liang** and others are also displayed (why?).
    (2) Fix (1) by using **egrep** more effectively. You should see only two lines.

63. (1) Get the number of students whose family name is **Wang**. You should get 4
    (2) Confirm (1) by retrieving the record of students whose family name is **Wang**. You should see five lines

64. (1) Get the number of students whose family name is **Patel**. You should get 5
    (2) Confirm (1) by retrieving the record of students whose family name is **Patel.** You should see seven lines.
65. Get the number of students whose family name is **Wong**. You should get 2.

66. (1) Retrieve the record of students whose family name is **Chen** or **Chan**. You should see four records.
    (2) Retrieve the record of students whose family name is **Liu** or **Lau**. You should see two records.
    ~~(3) Retrieve the record of students whose family name is **Liang** or **Xiang** or **Giang**. You should see six~~
    (3) Retrieve the record of students whose family name is **Lim** or **Lam**. You should see three records.
    (4) Retrieve the record of students whose family name is **Wang** or **Wong.** You should see six records.
    (5) Get the number of students whose family name is **Shen** or **Chen**. You should get 4.
    (6) Confirm (5) by retrieving record of students whose family name is **Shen** or **Chen**. You should see 4 records. The records are listed according to the order in the file. Now, sort the output based on family name and then given name. You should see

```
josh****        *********       Chen    Joshua
ychen**         *********       Chen    Yiming
awrfga          *********       Shen    Weitao
jshenxn         *********       Shen    Xu Nan
```

    (6) ~~Get the record of students whose family name is **Gu** or **Guo**. You should get two records.~~
    (6) Get the record of students whose family name is **Zhu** or **Zhou**. You should get three records.
67. `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 `ClassListACadded` columns are separated by tab.
    - Issue `cut -f 1 ClassListACadded` Observe that the only EECS user info (the first column) is displayed.
    - Issue `cut -f 3 classListACadded` Observe that the only surnames (the 3rd column) is displayed.
    - Issue `cut -f 1-3 classListACadded` Observe that columns 1 to 3 are displayed.
    - Issue `cut -f 1,3 classListACadded` Observe that the first and the 3rd column are displayed.

- Issue `cut -f 3,4 classListACadded > 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 classListACadded | 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:

68. 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.
```

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

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

70. 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.
```

71. 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.
```

72. contains **1980**

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

73. end with **1980**

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

74. contains **`sea`**

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

75. begins with **sea**

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

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

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

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

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

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

78. Go back to the parent directory `2021F`

```
cd ..
```

79. Issue utility

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

What did you get?

80. Now issue the utility

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

What did you get?

81. (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`

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 above we have experienced some of them, for example, **Pipes** |, **Filename substitution (wildcards)** * ? [], **Redirections** < > >>. Here you are going to practice some more functionalities, including **Command substitution** ` `, **Variable substitution** $, **Conditional sequence** && ||, and **Quotes** `''` and `" "`.

82. **Filename substitution (Wild-cards * ? []).** **<u>Don't get confused with * ? used in Regular Expression.</u>**

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

83. **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 Sat Nov 27 15:31:13 EDT 2021. Have a good day`
- Issue a single command to output `There are 194 students in EECS2031AC 2021F`
  where `194` comes from the result of a command that reads from file `ClassListACadded`.
- Issue a single command to output `There are 4 students in EECS2031AC with family name Nguyen`
  where `4` comes from the result of a command that reads from file `ClassListACadded`.

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

85. 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=EECS2031AC;` (no space around =) This assign variable `courseN` with value `EECS2031AC`
  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 filenae-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.

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 as 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 `lab7working`. 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
```

No spaces around **=**

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

Spaces after `[` and before `]`

The above evaluates if value of variable **num** is greater than 4. Observe the exit code 1, which means false. Note that in Bourne again shell (Bash), we can do **((3> 4))** **(($num > 4))** and there is no space restriction.

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.Lab7 ];  echo $?
```

The above tests if **xFile123.Lab7** is a regular file, and then displays the exit code 0. Now issue

```
sh-4.2$  [ -x  xFile123.Lab7 ];  echo $?
```

The above tests if **xFile123.Lab7** is an executable file. Now issue

```
sh-4.2$  [ -d  ../lab7working ];  echo $?
```

The above tests if **../lab7working** is a directory.

2. Problem B

If you haven't done so, copy the class list file **ClassListACadded** from **/eecs/dept/course/2019-20/W/2030tmp/** to your current working directory, and do the following exercise.

Copy the provided shell script **mygrep.sh** from the same directory. 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: classListACadded
Please enter search key:  Wang
```

If **mygrep.sh** does not work, issue **./mygrep.sh**

Observe the output. Then run again, enter your family name. Finally, enter search key **Kim**, observe the result.

3. Problem C

Copy the provided shell script **mygrepArg.sh** from the above directory to your working directory. 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 ClassListACadded Wang
There are 2 command line arguments: ClassListACadded Wang or ClassListACadded Wang
……
……
……
sh-4.2$ mygrepArg.sh ClasslistACadded Choi
There are 2 command line arguments: ClassListACadded Choi or ClassListACadded Choi
sh-4.2$
```

4. Problem C2

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 about the research result, outputs `Pattern "pattern" was not found in file "filename"` where `pattern` and `filename` are the 2nd and 1st arguments respectively.
- If the search pattern is found in file, then after the raw search results, outputs `Pattern "pattern" was found in file "filename"` where `pattern` and `filename` are the 2nd and 1st arguments respectively.

Sample Inputs/Outputs:

```
sh-4.2$ mygrepArg.sh ClassListACadded Choi
There are 2 command line arguments: ClassListACadded Choi or ClassListACadded Choi
Pattern "Choi" was not found in file "ClassListACadded"
sh-4.2$ mygrepArg.sh ClasslistACadded Wang
There are 2 command line arguments: ClassListACadded Wang or ClassListACadded Wang
……
……
……
```

```
Pattern "Wang" was found in file "ClassListACadded"
sh-4.2$
```

5. Problem C3

In the programs above, the script just executes `grep`, letting the "raw" result of `grep` be displayed. Sometimes we may want to turn off the raw outputs of the utility call.
Modify the program in C2 so that it produces the following output:

Sample Inputs/Outputs:

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

6. Problem D

Variable reading + Branching + Looping in Bourne (again) shell
Copy the script `numbers.sh` from the above directory. 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$
```

Note that you don't need to submit anything for this lab, but you are responsible for the materials covered in this lab.

In class we mostly follow the bourn shell. Our lab environment is Bourne Again shell (bash) environment. If you know bash specific syntaxes, feel free to use them.

If you have a UNIX-like environment on your local machine (e.g, Mac, Ubuntus), and prefer to work on your local machine (not recommended), here is how you can get started.

In a terminal of your Mac or Ubuntus, issue

```
ssh your-username@red.cse.yorku.ca
cp  /eecs/dept/course/2019-20/W/2030tmp/xxx .
```

This copies the file **xxx** to your EECS home directory.

Then in another terminal of your machine, navigate to your local working directory (using **cd**), then issue

```
scp your-username@red.cse.yorku.ca:xxx  .
```

This copies file **xxx** from your eecs home directory to your local working directory.
This can also be done by using a file transfer software such as FileZilla for Mac.