LAB 7 (Mar 31) — Unix Utilities and common shell functionalities

Due: optional. Apr 13 (Saturday) 11:59 pm

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**, **chmod**, **chgrp**, **grep/egrep**. We also discussed how to use pipe to use one utility's output as the input of another utility. In the next class we will also cover **uniq**, **sort**, **cmp/diff**, **cut**, **find** etc.

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

Go through the following 110 (small) practices. Write down your answers (with question #) besides the questions.

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

3. There are two files named `xFile2` and `xFile3` in directory **`/eecs/dept/course/2018-19/W/2031Z/`**
   Copy these two files to your current working directory using one 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 2 files are copied successfully to the current directory.

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

5. Rename file `xxx` to `xFile1`

6. 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).

   Also verify by listing files

```
$ your_command
xFile1 xFile2 xFile3
```

7. (1) Create a sub-directory named `2019` under your current working directory. (2) Then still in the current working directory, create a subdirectory `lab7a` under `2019`.

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

```
$ ls -R 2019
2019:
lab7a

2019/lab7a:
```

9. Still in current working directory, (1) remove `lab7a` 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) subdirectories `2019` and `2019/lab7a` under the current working directory, using `mkdir 2019/lab7a`. What do you get?

    (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.

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

    (3) confirm by recursively listing directory 2019 and its contents, you should see same results as in 8.

13. Move `xFile1` into subdirectory `lab7a` (with same name)

14. Then move all the other 2 files (together) into `lab7a` (using one entry of utility)

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

```
$ ls -l -R 2019
./2019:
total 4
drwx------ 2 yourname ugrad 4096 Jul  5 15:12 lab7a

./2019/lab7a:
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
```

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 of `lab7a`.

```
$ ls -l lab7a
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 `lab7a` itself

```
$ your-command
drwx------ 2 yourname ugrad 48 Mar  25 15:12 lab7a
```

18. Copy directory `lab7a` to a new directory named `lab7b` (using one utility).

19. Verify that `lab7b` is created and the two directory are identical

```
$ 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
```

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

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

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

23. (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
```

24. Move `xFile1`, which is in subdirectory `lab7b`, 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.

```
$ find . -name "xFile*"  or  find . -name "xFile?"
./lab7b/xFile2
./lab7b/xFile3
./xFile1
```

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

27. Navigate to directory `lab7working`
28. Verify that you are in `lab7working`

```
$ your-command
/cs/home/your_account/…/2019/lab7working
```

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 `lab7working`

```
$ 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
```

31. Display on stdout the contents of file `xFile1`
32. 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.)
33. Check how many lines `xFile1` contains. You should get 5.
34. Display (only) the first two line of `xFile1`
35. Display the last 3 lines of `xFile2`
36. (1) Confirm that `xFile1` and `xFile2` are identical now, using a utility, which should return silently (Hint: `cmp` or `diff`). (2) Then examine the exit code, you should get 0.
37. (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.
38. (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.
39. (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 are used by some professors to do automated grading of your lab or labtest :

```
gcc yourCode.c
a.out > yourOutputFile
cmp yourOutputFile professorsOutputFile
```

A student gets 0 if the last command does not return silently. That means, student gets 0 even if the student's output contains an extra white space. ☹

40. (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`.

```
$ 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
```

41. 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
```

42. Show on the stdout the content of `xFile123`, with duplicate lines removed/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
```

43. Remove the identical lines and save the result into a file `xFile123compact`.

    Hint: Just redirect the output of 42 using redirection `>`

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

45. 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?

46. Now issue `chmod 777 xFile1`, and then examine the resulting permission mode of the file. What do you get? You should get `-rwxrwxrwx` Can you understand what we are doing here?
47. Change the permission of `xFile123compact` using octal numbers so that the permission becomes

    `-rwxr--r--  1 yourname ugrad 140 Mar 25 17:23 xFile123compact`
48. Change the permission of `xFile123compact` by giving `group` an `execute` permission. You should get the following result: `-rwxr-xr--  1 yourname ugrad 145 Mar 25 17:23 xFile123compact`

49. Change the permission of `xFile123compact` by giving `group` a `write` permission, 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`
50. Change the permission of `xFile123compact` by removing `write` permission from `group`, and giving `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`

51. 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 xFile123compact`

52. 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` and then enter `"this is a xxx new line"`, followed by Ctrl-D.

Question 54-57 should be done without using `sort`. Utility `ls` can do sorting itself.

53. List the files in the current directory, sorted by the modification time. `xFile1` should be the first file in the list and other files are also sorted according to the modification time.

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

54. List the files, sorted by the modification time, in reverse order. `xFile1` should become the last file in the list.
55. List the files, sorted by the size of the files. `xFile123` should be the first file in the list and other files are also sorted according to the sizes.

```
$ your_command
total 20
```

```
-rw-r--r-- 1 yourname ugrad 377 Jul  6 13:35 xFile123
-rwxrwxrwx 1 yourname ugrad 168 Jul  6 13:42 xFile1
-r-xr-x--x 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
```

56. List the files, sorted by the size of the files, in reverse order.
57. Try to get the type of the file `xFile123compact` (Hint: use `file` utility)

```
xFile123compact: ASCII text
```

58. Try to get the type of one of your c source files.

```
hello.c: ASCII C program text
```

59. Try to get the type of one of your `a.out` files.

```
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared
libs), for GNU/Linux 2.6.32, …… not stripped
```

60. Recall that utility `who` lists the people who are currently logging on to the EECS server. Get how many people are currently log on to EECS server
61. Sort the list of people who are currently logging on.
62. Sort the list of people who currently logging on, based on the logon date (the 3rd column), in chronological order.
63. Get the information of the first three people who logged on the system.

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

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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 logging on using `yorku.ca` network. (2) Then 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. 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
```

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

77. The EECS department maintains the records of all the students, staff and faculty members in a file **/etc/passwd,** 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.

78. Issue a utility to find out how many people are in the list. You should get about `3656`.
79. Find out the number of people with name **Wang** in the file **/etc/passwd**. You should get about `43`.

The (modified) class list of our class can be found at **/eecs/dept/course/2018-19/W/2031Z/classlist**. Each line of the file contains one student information, where the first column is the EECS login id.

80. Get the number of students currently enrolled in the course. You should get `145`. You can use the file directly (by giving the pathname), or, copy the file to your current directory.

81. Retrieve your record from the class list.

82. Get the number of students whose family name is **Wang**. You should get 0

83. Get the number of students whose family name is **Zhang**. You should get 2.

84. Confirm 83 by retrieving the record of students whose family name is **Zhang**. You should see two lines.

85. Get the number of students whose family name is **Leung**. You should get 2.

86. Confirm 85 by retrieving the record of students whose family name is **Leung**. You should see two lines.

87. Get the number of students whose family name is **Chen** or **Chan**. You should get 2.

    Hint, from the perspective of regular expression, `Ch[ae]n` or "`Chan | Chen`" will do the trick.

88. Confirm 87 by retrieving the record of students whose family name is **Chen** or **Chan**. You should see two lines.

89. Look for the students whose eecs login id (in the first column of the file) starts with **cse**.

```
$ egrep cse classlist
cse*****        dshwet**        *********       Dixit, Shweta
cse*****        pavi***         *********       Kugarajah, Pavithra
cse*****        pbst***         *********       Thabet, Pierre
yuying          cse****         *********       Yu, Ying
```

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

```
$ your command
cse*****        dshwet**        *********       Dixit, Shweta
cse*****        pavi***         *********       Kugarajah, Pavithra
cse*****        pbst***         *********       Thabet, Pierre
```

91. `cut` is a utility that can extract columns of a text file. By default `cut` assumes that the columns are separated by tab. (We can also specify other delimiters such as space or comma). To specify the columns to extract, use `-f`.

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

    Issue `cut -f 4 classlist` Observe that the only names (the 4th column) is displayed.

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

    Issue `cut -f 1,4 classlist` Observe that the first and the 4th column are displayed.

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

92. 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.
```

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

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

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.
```

95. contains **1980**

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

96. end with **1980**

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

97. contains **sea**

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

98. begins with **sea**

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

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.
```

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.
```

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

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

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

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

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.
```

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,
```

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.
```

107. Go back to the parent directory

```
cd ..
```

108. Issue utility

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

What do you get?

109. Now issue the utility

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

What do you get?

110. Now issue

```
find  .  -name "xFile*"  -exec mv {}  {}.Lab7 \;
```

What we intend to do here?

List directory `lab7working` and examine what happens to the files in `lab7working`?

Note: for solutions using grep/egrep

- **you can use egrep or 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 the last lecture we 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 will practice some more functionalities, including **Command substitution** `` ` ` ``, **Variable substitution** `$`, **Conditional sequence** `&& ||`, and **Quotes** `''` and `" "`.

111. **Filename substitution (Wild-cards * ? []).** Navigate to your working directory.

- Issue `ls *` Observe that all files in the 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` (but not `xFile123.Lab7 and xFile123compact.Lab7)` are listed (why?)
- Issue `ls xFile???.Lab7` Observe that only file `xFile123.Lab7` is listed (why?)

112. **Command substitution** ``

- Issue a single command to output current date and time. Something like
  `Hello, now is Sat Mar 30 18:08:44 EDT 2019. Have a good day!`
- Issue a single command to output `There are 145 students in EECS2031Z`. where `145` comes from the result of a command that reads from file `classlist`.
- Issue a single command to output `There are 2 students in EECS2031Z with family name Zhang` where `2` comes from the result of a command that reads from file `classlist`.
- Issue a single command to output `There are 3 students in EECS2031Z whose eecs username begins with cse` where `3` 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 Wang 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 Wang 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 Wang classlist || echo HELLO`, observe that HELLO is printed (why?).
- Issue `egrep -w Zhang 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=2031Z;` (no space around =) This assign variable `courseN` with value `2031Z`
  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.

End of lab7, finally 

Submission

This lab is optional, in that officially it is not a weighed lab. However, if you submit by the deadline, I will be happy to evaluate it and potentially you will get some bonus marks.

If you decide to submit, write the question numbers and your answers for question 1-110, and 112 in a file, or, make a scan/image of your handwriting answers, and submit the file using **`submit 2031Z lab7 your_file_name`**