Student Name: _____ **Lab Section:** _____ **/3** marks

Due Date - Upload to Blackboard by 8:30am Monday February 27, 2012

Submit the completed lab to Blackboard following the Rules for submitting Online Labs and Assignments.

Linux Shell Features

Commands, topics, and features covered

Use the on-line help (**man** command) for the commands listed below for more information.

- **`bash`** – Linux full-featured **Bourne-Again SHell** for interactive use
- **`dash`** – smaller Bourne-like shell for use in scripts (may be linked to from **`/bin/sh`**).
- **`alias`** – (**`man bash`**) built-in **`bash`** command to create synonyms for command names.
- **`history`** – (**`man bash`**) built-in **`bash`** command to show history of commands typed
- Shell **wildcard** (GLOB char) patterns: **`*  ?  [...] [!...]`**
- Shell I/O **redirection**: **`>  <  >>`**
- Shell command **pipelining**: **`|`**
- Shell **aliases**
- Shell curly **brace** expansion: **`{...,...}`**
- Shell command **history** **`!!`** **`!120`**
- **`sum`** – generate a checksum
- **`date`** – show the current time and date
- **`grep`** – search for patterns and print the lines containing the patterns
- **`wc`** – count lines, words, and characters
- **`head`** – select lines at the start (head) of the input (default is to select first 10 lines)
- **`tail`** – select lines at the end (tail) of the input (default is to select last 10 lines)
- **`nl`** – read lines and prefix them with line numbers (see also: **`cat -n`**)

Do this lab as a regular user, not as the root or super-user. **Do not use root for this lab.** Work in your own home directory.

1 Using Wildcards (GLOB chars) in Pathname Specifications

- Use the **`echo`** command to see how patterns expand **first**, **before** you use them in a real command line, e.g. **`echo /etc/pas* ; echo /dev/sd* ; echo /bin/*sh`**

`*`	Matches 0 or more characters
`?`	Matches any 1 character (including blanks!)
`[aed]`	Matches 1 character in the list: **`a`**, **`e`**, or **`d`**
`[a-e]`	Matches 1 character in the list from **`a`** to **`e`** inclusive (i.e. **`a b c d e`**)
`[!a-e]`	Matches 1 character that is **not** in the list from **`a`** to **`e`** inclusive (also written as **`[^a-e]`**)

1.1 Using `?` to match any single character

As we do with most sections of each lab, we recursively remove and recreate a directory in which to do this section. This lets you return to this place and redo a section without having files left over from previous work.

```
[user@localhost ]$ cd ; rm -rf lab4.1 ; mkdir lab4.1 ; cd lab4.1
[user@localhost lab4.1]$ touch  f1   f2   f3
[user@localhost lab4.1]$ touch  f10  f20  f30
[user@localhost lab4.1]$ touch  f11  f12  f13
[user@localhost lab4.1]$ touch  f33  fffff
[user@localhost lab4.1]$ mkdir  dir1  dir2  dir3
[user@localhost lab4.1]$ ls
dir1  dir2  dir3  f1  f10  f11  f12  f13  f2  f20  f3  f30  f33  fffff
[user@localhost lab4.1]$ ls | sum
33442    1              ( make sure you get this number, otherwise start over at the beginning)
[user@localhost lab4.1]$ cp  f?  dir1
```

a) Which files have been copied to the directory **dir1**? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.1]$ cp  f?0  dir2
```

b) Which files have been copied to the directory **dir2**? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.1]$ cp  f?3  dir3
```

c) Which files have been copied to the directory **dir3**? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.1]$ echo  ?  ??  ?????
```

d) One of the above three patterns fails to expand to any pathname. Which pattern, and why?

_____

_____

_____

```
[user@localhost lab4.1]$ cp  ?  ??  ?????  dir3
```

e) **Why** does the above command line give an **error** message from the copy command?

_____

_____

```
[user@localhost lab4.1]$ ls  /dev/tty*
```

f) The above command shows all terminal devices configured in your Linux system. Try it.
What command line, similar to the above, shows **only** the **five-character** terminal device names,
i.e. shows **tty10** and **tty63** and **ttyS0**, etc. but does *not* show **tty** or **tty1** or **tty9** etc.?

_____

1.2 Using * to match zero or more characters

This section **depends** on the files created at the start of section 1.1. **Create those files first**.

```
[user@localhost lab4.1]$ cd ; rm -rf lab4.2 ; mkdir lab4.2 ; cd lab4.2
[user@localhost lab4.2]$ cp -a ../lab4.1/.  .    (note the single dot destination directory)
[user@localhost lab4.2]$ ls
dir1  dir2  dir3  f1  f10  f11  f12  f13  f2  f20  f3  f30  f33  fffff
```

```
[user@localhost lab4.2]$ ls | sum
33442    1            ( make sure you get this number, otherwise start over at the beginning of 1.1)
[user@localhost lab4.2]$ rm  dir?/*          (clean out all files in these directories)
[user@localhost lab4.2]$ ls  f*
```

a) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.2]$ ls  -d  d*
```

b) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.2]$ ls  f*1             (that pattern contains a digit, not a letter)
```

c) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.2]$ ls  -d  *1          (that pattern contains a digit, not a letter)
```

d) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.2]$ ls  -d  *1*         (that pattern contains a digit, not a letter)
```

e) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

1.3 Using [ ] to match a single character from a list of characters

```
[user@localhost ]$ cd ; rm -rf lab4.3 ; mkdir lab4.3 ; cd lab4.3
[user@localhost lab4.3]$ touch   hat  help  hit  hot  hut
[user@localhost lab4.3]$ ls | sum
07916    1            ( make sure you get this number, otherwise start over at the beginning of 1.3)
[user@localhost lab4.3]$ ls  h[ao]t
```

f) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.3]$ ls  h[aeiou]t
```

g) What is the output of the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.3]$ ls  h[aeiou]*
```

h) Comparing with the previous output, which **additional** file is displayed? _____

i) Why? _____

_____

_____

```
[user@localhost lab4.3]$ cd ; rm -rf lab4.3b ; mkdir lab4.3b ; cd lab4.3b
[user@localhost lab4.3b]$ touch  sda  sdb  sdc  sdd
[user@localhost lab4.3b]$ touch  sda1  sdb2  sdc3  sdd4
[user@localhost lab4.3b]$ ls | sum
61395    1            ( make sure you get this number, otherwise start over at the beginning of 1.3)
```

```
[user@localhost lab4.3b]$ ls  -i  sd[abc]
```

j) Which names are output by the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.3b]$ ls  -i  sd[a-c]
```

k) Which names are output by the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.3b]$ ls  sd[b-d][1-3]
```

l) Which names are output by the last command, above? (First try to **answer without typing** the command.)

_____

```
[user@localhost lab4.3b]$ echo  sd[a-d][1-4]
```

m) Which names are output by the last command, above? (First try to **answer without typing** the command.)

_____

2 Using Shell I/O Redirection

- Shell I/O redirection can appear *anywhere* in the command line, even at the very beginning.
- The redirection syntax is **removed** from the command line **before** calling the command to run, so it is not counted as an *argument* on the command line. The command knows nothing about it.
- Redirection happens **first**, before the command runs. Redirection files are **erased** to **empty** even if the command is **not found** or has **no output**. This truncation happens **before the command runs**.

2.1 Redirection happens first, before the command runs

```
[user@localhost ]$ cd ; rm -rf lab4.4 ; mkdir lab4.4 ; cd lab4.4
[user@localhost lab4.4]$ date > foo              (erase foo and then put the date in it)
[user@localhost lab4.4]$ cat foo foo foo         (show the file contents three times)
[user@localhost lab4.4]$ cat foo foo foo >foo    (send foo into foo three times)
[user@localhost lab4.4]$ cat foo                 (what is in foo now?)
```

a) Explain in **detail** what you see inside the file **foo** now and **exactly** how the contents got that way:

_____

_____

_____

2.2 Creating a text file from keyboard input

- Most Unix/Linux commands that read data from **file names** as arguments will read your **keyboard** ("*standard input*") if you don't supply any file names to read. Examples: **cat**, **wc**, **sum**, **sort**, **grep**

```
[user@localhost lab4.4]$ cat > myfile          (no file names are given to the cat command)
```

The cursor waits at the beginning of the next line for input from the keyboard. No prompt appears!

Type: **Hello world!**
Press **[Enter]**
Press **[Ctrl]+d** *(hold **Ctrl** down and then press the **d** key to **close** the keyboard and send EOF)*

```
[user@localhost lab4.4]$ cat myfile                    (This is Output One.)
```

Note the output of the above command. This is **Output One**.

```
[user@localhost lab4.4]$ cat > myfile          (no file names are given to the cat command)
```
Type your name. *(No prompt appears! You are typing directly into the **cat** program.)*
Press **[Enter]**
Press **[Ctrl]+d** *(hold **Ctrl** down and then press the **d** key to **close** the keyboard and send EOF)*

```
[user@localhost lab4.4]$ cat myfile                    (This is Output Two.)
```

a) Explain **exactly** why does *Output Two* show **none** of the text from *Output One*?

2.3 Appending to or Overwriting an existing file

```
[user@localhost lab4.4]$ date >  bar          (note only one > on this line!)
[user@localhost lab4.4]$ date >> bar
[user@localhost lab4.4]$ date >  bar          (note only one > on this line!)
[user@localhost lab4.4]$ date >> bar
[user@localhost lab4.4]$ date >> bar
```

a) How many lines are in output file **bar** now (**guess** without executing commands first)? _____

2.4 Concatenating files using `cat`

```
[user@localhost lab4.4]$ cat > f1
```
Enter the following keyboard text (and **close** the file as before using EOF): **Hello everybody**
```
[user@localhost lab4.4]$ cat > f2
```
Enter the following keyboard text (and **close** the file as before using EOF): **My name is *My Name***
```
[user@localhost lab4.4]$ cat > f3
```
Enter the following keyboard text (and **close** the file as before using EOF): **Good-bye**
```
[user@localhost lab4.4]$ cat f1 f2 f3 > f4
```

a) **Describe** the **contents** of file **f4** after the last command, above. What happened?

3 Using shell pipelines to connect commands

```
[user@localhost ]$ cd ; rm -rf lab4.5 ; mkdir lab4.5 ; cd lab4.5
[user@localhost lab4.5]$ ls -l /bin          (too much output to see on one screen)
[user@localhost lab4.5]$ ls -l /bin | less
```
Use the **[spacebar]** to jump to the next screen of information in **less**. You can use **q** to exit the command.

```
[user @localhost lab4.5]$ grep 'tmpfs' /etc/fstab
[user @localhost lab4.5]$ grep 'sysfs' /etc/fstab
[user @localhost lab4.5]$ grep 'sysfs' /etc/fstab | grep 'tmpfs'
```

a) The first two commands show output, so why is there no output from the **last** command pipeline, above?

```
[user@localhost lab4.5]$ nl /etc/passwd
[user@localhost lab4.5]$ nl /etc/passwd | head -n 5
[user@localhost lab4.5]$ nl /etc/passwd | head -n 5 | tail -n 1
```

b) Describe **concisely** what the last pipeline above **selects** from any input file (read some **man** pages first):

_____

_____

_____

4 Using shell aliases

- Aliases let you define your **own** command names. You can also **redefine** existing command names and/or add your **favourite** options. Many Linux systems come with some aliases defined already.

4.1 Listing all current shell aliases

```
[user@localhost lab4.5]$ alias
[user@localhost lab4.5]$ alias | grep 'vim'
```

a) Record the output of the last command pipeline, above: _____

4.2 Creating a shell alias

```
[user@localhost lab4.5]$ alias myls='ls -alF'
[user@localhost lab4.5]$ alias | grep 'myls'
```

a) Record the output of the last command pipeline, above: _____

```
[user@localhost lab4.5]$ ls  ..                    (two dots - parent directory)
[user@localhost lab4.5]$ myls  ..                  (two dots - parent directory)
```

b) **Describe** how typing **`myls`** gives *different* output compared to typing **`ls`**:

_____

_____

_____

4.3 Removing a single shell alias

```
[user@localhost lab4.5]$ alias myls='ls -alF'
[user@localhost lab4.5]$ myls
[user@localhost lab4.5]$ unalias myls
[user@localhost lab4.5]$ alias | grep 'myls'
[user@localhost lab4.5]$ myls
```

a) Record **exactly** the **error message** from using the **undefined** alias (undefined command name):

_____

4.4 Temporarily bypassing a current alias for an existing command

```
[user@localhost lab4.5]$ alias ls='ls -al'
[user@localhost lab4.5]$ ls  ..                    (two dots - parent directory)
[user@localhost lab4.5]$ \ls  ..                   (two dots - parent directory)
```

a) **Describe** how typing **`\ls`** gives *different* output compared to typing the **aliased** version of **`ls`**:

_____

_____

b) Given what you see above, what is the function of the backslash character **`\`** in front of an alias?

_____

c) Give a command to remove the **`ls`** alias: _____

d) What command will remove **all** current aliases (RTFM): _____

4.5 Curly Brace Expansion - `{...,...}`

```
[user@localhost ]$ cd ; rm -rf lab4.6 ; mkdir lab4.6 ; cd lab4.6
[user@localhost lab4.6]$ touch  file{a,b,c}  in{2,3}days
```

a) Record the **five** files that were created by the above command (**guess** without executing commands first):

_____

```
[user@localhost lab4.6]$ rm  file{b,c}  i{n2,n3}days
```

b) Which files were removed by the above command? (First try to **answer without typing** the command.)

_____

IMPORTANT: Make sure all the **braces match** and that there are **NO BLANKS** inside the following pattern:

```
[user@localhost]$ mkdir -p lab4/{old,new}/{lab{1,2},theory{1,2},test{1,2}}
```

c) How many directories **and** sub-directories in **total** have been created by the above command? _____

d) What two-command **pipeline** could be used to **find** all the pathnames in the **`lab4`** directory and then **count** them, outputting just a **single number** for the count of pathnames (same answer as previous question)? (Hint: What command **finds** pathnames? What command counts lines? See your previous labs!)

_____

5 Using shell command line history (similar to DOS doskey*)*

Purpose	Command Line Example
Display the current history buffer	**`history | less`**
Re-execute the last command	**`!!`**
Re-execute any previous command	Use the up and down arrow keys, then push [Enter]
Re-execute any previous command	**`!`*`n`*** where ***`n`*** is the event number as listed in the **`history`** output

5.1 Using the built-in shell history command

- **`history | less`**

a) **Describe** what **kind** of output the above history command pipeline generates:

_____

_____

b) **Describe** what the double-exclamation command **`!!`** would do, but **don't** actually do it:

_____

c) **Describe** what typing exclamation-two **`!2`** would do, but don't actually do it:

_____

d) If the **`history`** command shows you a command, number **120**, that you would like to **re-execute**, what would you type into the shell to do that?

_____

6 Review: Basic Commands

a) As an ordinary user, type the following three commands and record the output of the last one:

- **`cd ; cp -a /etc/passwd foo ; ls -lis foo`**

_____

b) What is the difference between the command line shell **prompt** for an **ordinary** user and the **root** user? (In other words, how does the shell **prompt** indicate that you are the **super-user**?)

_____

c) Execute, then **describe** the purpose of the **`whoami`** command (read the man page):

_____

d) Execute, then **describe** the purpose of the command: **`hostname`**

_____

e) What is the purpose of the **`-i`** switch (option) to the **`ls`** command?

_____

f) What is the purpose of the **`-d`** switch (option) to the **`ls`** command?

_____

g) Execute, then **describe** the purpose of the command: **`pwd`**

_____

h) Give the **output** of the command: **`wc /etc/passwd`**

_____

i) **Describe** the purpose of the **`wc`** command:

_____

j) What command name and options copy an **entire** directory, including **all** the other files and sub-directories stored under it?

_____

k) What command name do you use to delete/remove an **empty** directory?

_____

l) What command name and options do you use to delete/remove a **complete** directory structure including **all** the other files and sub-directories stored under it?

_____

m) What command name and options would **find** any **pathname** ending in the ***basename*** of **`passwd`**, found in any directory, starting from the **root** directory?

_____

n) What command name and options would **find** and print all the **pathnames** under the directory **`/etc`**?

_____

o) What command would **find** and print all the pathnames owned by the **root** user under **`/var/spool`**:

_____

p) Rewrite the pathname **`/usr/./bin/../local/./bin/../sbin`** as a simple **absolute** pathname, **without** using any "**`.`**" or "**`..`**":

_____

q) What is the full command name and options used to safely shut down Linux **immediately** from the command line, assuming you have a **`root`** privilege shell prompt? (see the lecture notes!)

_____

Did you read **all** the words?