This is Lab Worksheet 5 - not an Assignment

This Lab Worksheet contains some practical examples that will prepare you to complete your Assignments. You do **not** have to hand in this Lab Worksheet. Make sure you complete the separate Assignments on time. Quizzes and tests may refer to work done in this Lab Worksheet; save your answers.

Before you get started - REMEMBER TO READ *ALL* THE WORDS

You must have an account on the Course Linux Server to do this lab. Log in to the server and use the shell. Review the Class Notes related to this worksheet as you work through it. Leave your work on the Linux server. Do not delete any work from the Linux server until the term is over and your marks are complete!

Commands introduced and used in this Lab

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

Log in to the Course Linux Server to do all commands in this lab. Set your bash **`PS1`** shell prompt to show your login name, computer name, and the **basename** of your current directory, just as you did in the previous Lab. **Leave your finished work on the server; do not delete it when you are finished the worksheet**.

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`**

`*`	zero (0) or more characters
`?`	any one (1) character (including blanks and unprintable characters!)
`[aeAd]`	exactly one (1) character in the list between brackets: **`a`**, **`e`**, **`A`**, or **`d`**
`[^BbCc]`	exactly one (1) character that is **not** in the list of chars (also written as **`[!BbCc]`**)
`[0-9]`	exactly one character in a range of characters determined by **LOCALE**
`[^0-9]`	exactly one character **not** in the **LOCALE**-dependent range of characters

Do not use **character ranges**, especially alphabetic ranges, without first understanding your system **LOCALE** setting. The range **`[a-z]`** often includes the upper-case letters from **`B`** through **`Z`** and may not include **`A`**.

Use the **`echo`** command to see how patterns expand **first**, **before** you use them in a real command line.

Lab5.1: 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 lab5.1 ; mkdir lab5.1 ; cd lab5.1
[user@localhost lab5.1]$ touch  f1   f2   f3
[user@localhost lab5.1]$ touch  f10  f20  f30
[user@localhost lab5.1]$ touch  f11  f12  f13
[user@localhost lab5.1]$ touch  f33  fffff
[user@localhost lab5.1]$ mkdir  dir1  dir2  dir3
[user@localhost lab5.1]$ ls
dir1  dir2  dir3  f1  f10  f11  f12  f13  f2  f20  f3  f30  f33  fffff
[user@localhost lab5.1]$ ls | sum
33442    1              ( make sure you get this number, otherwise start over at the beginning)
```

Use the **echo** command to see how patterns expand **first**, **before** you use them in a real command line.

```
[user@localhost lab5.1]$ cp  f?  dir1
```

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

_____

```
[user@localhost lab5.1]$ cp  f?0  dir2
```

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

_____

```
[user@localhost lab5.1]$ cp  f?3  dir3
```

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

_____

```
[user@localhost lab5.1]$ echo  ?  ??  ?????
```

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

_____

_____

_____

```
[user@localhost lab5.1]$ cp  ?  ??  ?????  dir3
```

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

_____

_____

```
[user@localhost lab5.1]$ 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.?

_____

Lab5.2: 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 lab5.1]$ cd ; rm -rf lab5.2 ; mkdir lab5.2 ; cd lab5.2
[user@localhost lab5.2]$ cp -a ../lab5.1/.  .   (note the single dot destination directory)
[user@localhost lab5.2]$ ls
dir1  dir2  dir3  f1  f10  f11  f12  f13  f2  f20  f3  f30  f33  fffff
[user@localhost lab5.2]$ ls | sum
33442    1            ( make sure you get this number, otherwise start over at the beginning of 1.1)
```

Use the **echo** command to see how patterns expand **first**, **before** you use them in a real command line.

```
[user@localhost lab5.2]$ rm  dir?/*          (clean out all files in these directories)
[user@localhost lab5.2]$ ls  f*
```

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

```
[user@localhost lab5.2]$ ls  -d  d*
```

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

```
[user@localhost lab5.2]$ 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 lab5.2]$ 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 lab5.2]$ 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.)

Lab5.3: Using [ ] to match a single character from a list of characters

```
[user@localhost ]$ cd ; rm -rf lab5.3 ; mkdir lab5.3 ; cd lab5.3
[user@localhost lab5.3]$ touch  hat  help  hit  hot  hut
[user@localhost lab5.3]$ ls | sum
07916    1            ( make sure you get this number, otherwise start over at the beginning of 1.3)
```

Use the **echo** command to see how patterns expand **first**, **before** you use them in a real command line.

```
[user@localhost lab5.3]$ ls  h[ao]t
```

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

```
[user@localhost lab5.3]$ ls  h[aeiou]t
```

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

```
[user@localhost lab5.3]$ ls  h[aeiou]*
```

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

i) Why? _____

_____

_____

```
[user@localhost lab5.3]$ cd ; rm -rf lab5.3b ; mkdir lab5.3b ; cd lab5.3b
[user@localhost lab5.3b]$ touch  sda  sdb  sdc  sdd
[user@localhost lab5.3b]$ touch  sda1  sdb2  sdc3  sdd4
[user@localhost lab5.3b]$ ls | sum
61395    1            ( make sure you get this number, otherwise start over at the beginning of 1.3)

[user@localhost lab5.3b]$ ls  sd[abc]
```

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

_____

```
[user@localhost lab5.3b]$ ls  sda[1-4]
```

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

_____

```
[user@localhost lab5.3b]$ ls  sd[bc][1-3]
```

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

_____

```
[user@localhost lab5.3b]$ echo  sd[^ab][^12]
```

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

_____

Character ranges [a-z] are dangerous! Always test them first!

Do not use **character ranges**, especially alphabetic ranges, without first understanding your system **LOCALE** setting. The range **`[a-z]`** often includes the upper-case letters from **`B`** through **`Z`** and may not include **`A`**. Try the following test on your **Fedora 12** system and also on the **Course Linux Server** and note the differences:

1. `$ touch a A b B y Y z Z`
2. `fedora$ echo [a-z]` # *Fedora gives:* **`a b y z`**
3. `fedora$ echo [A-Z]` # *Fedora gives:* **`A B Y Z`**
4. `ubuntu$ echo [a-z]` # *Ubuntu gives:* **`a B b Y y Z z`**
5. `ubuntu$ echo [A-Z]` # *Ubuntu gives:* **`A a B b Y y Z`**

Fedora uses an old English-only LOCALE that expects ASCII characters where Ubuntu uses a modern International LOCALE that also includes Unicode characters. Do not use **character ranges**, especially alphabetic ranges, without first understanding your system **LOCALE** setting.

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**.

Lab5.4: Redirection happens *first*, *before* the command runs

```
[user@localhost ]$ cd ; rm -rf lab5.4 ; mkdir lab5.4 ; cd lab5.4
[user@localhost lab5.4]$ date > foo              (erase foo and then put the date in it)
[user@localhost lab5.4]$ cat foo foo foo         (show the file contents three times)
[user@localhost lab5.4]$ cat foo foo foo >foo    (send foo into foo three times)
[user@localhost lab5.4]$ 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:

Creating a text file from keyboard input using keyboard EOF ^D

- 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 lab5.4]$ 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 lab5.4]$ cat myfile                    (This is Output One.)
```

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

```
[user@localhost lab5.4]$ 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 lab5.4]$ cat myfile                    (This is Output Two.)
```

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

Appending to or Overwriting an existing file using > and >>

```
[user@localhost lab5.4]$ date >  bar          (note only one > on this line!)
[user@localhost lab5.4]$ date >> bar
[user@localhost lab5.4]$ date >  bar          (note only one > on this line!)
[user@localhost lab5.4]$ date >> bar
[user@localhost lab5.4]$ date >> bar
```

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

Concatenating multiple files using `cat`

```
[user@localhost lab5.4]$ cat > f1
```

Enter the following keyboard text (and **close** the file as before using EOF): **Hello everybody**

```
[user@localhost lab5.4]$ cat > f2
```
Enter the following keyboard text (and **close** the file as before using EOF): **My name is *My Name***
```
[user@localhost lab5.4]$ cat > f3
```
Enter the following keyboard text (and **close** the file as before using EOF): **Good-bye**
```
[user@localhost lab5.4]$ cat f1 f2 f3 > f4
```

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

Lab5.5: Using shell pipelines | to connect commands

```
[user@localhost ]$ cd ; rm -rf lab5.5 ; mkdir lab5.5 ; cd lab5.5
[user@localhost lab5.5]$ ls -l /bin          (too much output to see on one screen)
[user@localhost lab5.5]$ 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 lab5.5]$ grep 'the' /etc/fstab
[user @localhost lab5.5]$ grep 'none' /etc/fstab
[user @localhost lab5.5]$ grep 'the' /etc/fstab | grep 'none'
```

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

```
[user@localhost lab5.5]$ nl /etc/passwd
[user@localhost lab5.5]$ nl /etc/passwd | head -n 5
[user@localhost lab5.5]$ 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):

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. These predefined aliases may be set in system RC files (under **/etc**) or in a **.bashrc** file placed into your account HOME directory.

Listing all current shell aliases

```
[user@localhost lab5.5]$ alias                (list all current aliases - may be none)
[user@localhost lab5.5]$ grep -w 'alias' /etc/skel/.bashrc
[user@localhost lab5.5]$ grep -w 'alias ls=' /etc/skel/.bashrc
```

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

b) Give a two-command shell **pipeline** that would count the number of aliases in **/etc/skel/.bashrc**

Creating a shell alias (for this shell only)

```
[user@localhost lab5.5]$ alias myls='ls -alF --color=auto'
[user@localhost lab5.5]$ alias | grep 'myls'
[user@localhost lab5.5]$ ls  ..                    (two dots - parent directory)
[user@localhost lab5.5]$ myls  ..                  (two dots - parent directory)
```

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

_____

_____

Removing a single shell alias (for this shell only)

```
[user@localhost lab5.5]$ alias myls='ls -alF --color=auto'
[user@localhost lab5.5]$ myls
[user@localhost lab5.5]$ unalias myls
[user@localhost lab5.5]$ alias | grep 'myls'
[user@localhost lab5.5]$ myls
```

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

_____

Temporarily bypassing a current alias using backslash

Putting a backslash in front of a command name causes the shell to ignore any alias for that command.

```
[user@localhost lab5.5]$ alias ls='ls -alF --color=auto'
[user@localhost lab5.5]$ ls  ..                    (two dots - parent directory)
[user@localhost lab5.5]$ \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): _____

Lab5.6: Curly Brace Expansion - { . . . , . . . }

Curly braces surround comma-delimited text that you want the shell to repeat. The pattern **foo{1,2,3}** is expanded to be three arguments **foo1 foo2 foo3** by the shell. Blanks inside the braces need to be quoted to hide them from the shell. You can nest braces: **foo{bar{1,2,3},more{a,b},end}**

Use the **echo** command to see how patterns expand **first**, **before** you use them in a real command line.

```
[user@localhost ]$ cd ; rm -rf lab5.6 ; mkdir lab5.6 ; cd lab5.6
[user@localhost lab5.6]$ echo  foo{1,2,3}  bar{fly,maid,tender,stool}
[user@localhost lab5.6]$ echo  foo{bar{1,2,3},more{a,b},end}
[user@localhost lab5.6]$ touch  file{a,b,c}  in{1,2,3}days
```

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

_____

```
[user@localhost lab5.6]$ rm  fil{ea,eb}  i{n1,n2}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 lab5/{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 **`lab5`** 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!)

_____

Review: Basic Commands

e) How do **`mv`** and **`cp`** differ?

_____

_____

f) **Describe** the purpose of the **`whoami`** command (read the man page):

_____

g) **Describe** the purpose of the command: **`head`**

_____

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

_____

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

_____

j) **Describe** the purpose of the command: **`grep`**

_____

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

_____

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

_____

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

_____

n) 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?

_____

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

_____

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

_____

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

_____

r) How do you create a directory **`a/b/c/d/e`** including all the parent directories, in one command line?

_____

s) Use a **`head`** and **`tail`** pipeline to show only lines 10-20 from the file **`/etc/passwd`**:

_____

t) Use a **`head`** and **`tail`** and **`sort`** pipeline to show only lines 10-20 from the file **`/etc/passwd`**, sorted in descending (not ascending) order (read the **`sort`** man page):

_____

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

_____