

## 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.

### ***Commands, topics, and features covered***

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

- **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
- **grep** – search for patterns and print the lines containing the patterns
- **head** – select lines at the start (head) of the input (default is to select first 10 lines)
- **history** – (**man bash**) built-in **bash** command to show history of commands typed
- **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. Keep your work.

### **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**

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

### **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)
[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)
[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)
[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 -i 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 -i sd[a-c]
```

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

---

```
[user@localhost lab5.3b]$ 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 lab5.3b]$ echo sd[a-d] [1-4]
```

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

---

## 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

- 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

```
[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 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 on creation.

### 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: \_\_\_\_\_

### Creating a shell alias

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

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

---



---



---

### Removing a single shell alias

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

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

---



---



---

### Temporarily bypassing a current alias for an existing command

```
[user@localhost lab5.5]$ alias ls='ls -aF --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}`

```
[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!)

---

## Using shell command line history (similar to DOS doskey)

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

## 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?
- 

## 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 **“..”**:
-