This is Lab Worksheet 4 - 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`** – (**`help alias`**) built-in **`bash`** command to create synonyms for command names.
- **`bash`** – Shell **aliases**
- **`bash`** – Shell **wildcard** (GLOB char) patterns: **`*  ?  [...] [!...]`**
- **`sum`** – generate a checksum
- **`unalias`** – remove an alias, or all aliases using **`-a`**

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, punctuation, and unprintable characters!)
`[aeAd]`	exactly one (1) character in the list between brackets: **`a`**, **`e`**, **`A`**, or **`d`**
`[!abc]`	exactly one (1) character that is **not** in the list of characters (sometimes written as **`[^abc]`**)
`[0-9]`	**SEE WARNING**! exactly one character in a **range** of characters determined by **LOCALE**
`[!0-9]`	**SEE WARNING**! exactly one character **not** in the **LOCALE**-dependent **range** of characters

WARNING: DO NOT USE LETTER RANGES: Do not use letter **ranges** separated by a dash, e.g. **`[a-z]`**, without first understanding your system **LOCALE** setting. The dash range **`[a-z]`** often includes the upper-case letters from **`B`** through **`Z`** and may not include **`A`**. Using digits **`[0-9]`** is the only safe range allowed.

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

```
$ echo /some/pattern*      # echo the pattern on the screen before using the real command name!
$ rm /some/pattern*        # now use the real command name with the verified pattern
```

Lab 4.1: 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 of 4.1)
```

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

```
[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.?

Lab 4.2: Using * to match zero or more characters

This section **depends** on the files created at the start of the previous section. **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 4.1)
```

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

```
[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.)

_____

Lab 4.3: Using [ ] to match a single character from a list of characters

Square brackets are shell metacharacters that indicate a GLOB pattern match for any **one** character in the list of characters between the square brackets. You can also match any **one** character that is *not* in the list by starting the list with an exclamation point **!**, e.g. **[!abc]** Circumflex is sometimes also allowed but doesn't work in all shells, so the exclamation point is preferred instead of using **[^abc]**

```
[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 4.3)
```

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

```
[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 with rm above)

[user@localhost lab4.3b]$ ls  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  sda[1-4]
```

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

```
[user@localhost lab4.3b]$ 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 lab4.3b]$ echo  sd[!ab][!12]
```

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

List complements sometimes work using a leading circumflex, as **`[^abc]`**, to mimic the behaviour of Regular Expressions, but the circumflex doesn't work in all shells so **`[!abc]`** is preferred in GLOB patterns.

Dashed character ranges [a-z] are not always predictable! Always test them first!

DO NOT USE DASHED RANGES: Do not use **character ranges** separated by a dash, 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`**. Using only digits in the range, e.g. **`[0-9]`**, may be safe.

The following shows the difference on two example Linux systems, depending on the **LOCALE** environment:

1. `$ touch a A b B y Y z Z`
2. `server$ echo [a-z]` *# Server gives only lower-case:* **`a b y z`**
3. `server$ echo [A-Z]` *# Server gives only upper-case:* **`A B Y Z`**
4. `desktop$ echo [a-z]` *# Desktop gives an incomplete mix:* **`a B b Y y Z z`**
5. `desktop$ echo [A-Z]` *# Desktop gives an incomplete mix:* **`A a B b Y y Z`**

The Server uses a traditional English-only LOCALE that expects only ASCII (English) characters where the Desktop uses a modern International LOCALE (e.g. **`LC_CTYPE=en_CA.utf8`**) that also includes Unicode characters in the range. Do not use dashed **character ranges**, especially alphabetic ranges, without first understanding your system **LOCALE** setting. Verify GLOB patterns using **`echo`** ! For a fuller explanation on Internationalization problems, including the use of POSIX character classes such as **`[:lower:]`** and **`[:upper:]`** read:

http://teaching.idallen.com/cst8177/15w/notes/000_character_sets.html

Lab 4.4: Using shell aliases

Aliases are **built-in to your current shell**. They are not separate commands like **`date`** and **`ls.`** Aliases let you define your **own** command names. You can also **redefine** existing command names and/or add your **favourite** options to existing command names. Many Linux systems come with some shell 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. Know what aliases are set in your account before you start typing!

Listing all current shell aliases

```
[user@localhost ~]$ alias                    (list all current aliases - may be none)
```

a) In which man page are shell aliases documented? : _____

You can also use the **`help`** feature of your shell to get help on any built-in command, e.g. **`help alias`**

Creating a shell alias (for this shell only)

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

e) Why doesn't **`man alias`** work? _____

Review: Basic Commands

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

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

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

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

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

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

g) 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?

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

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

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

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

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