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

- **`bash`** – Shell command **pipelining**: **`|`**
- **`bash`** – Shell I/O **redirection**: **`>  <  >>`**
- **`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`**)
- **`tail`** – select lines at the end (tail) of the input (default is to select last 10 lines)
- **`tr`** – translate characters from standard input, e.g. lower-case to upper-case, etc.
- **`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 Shell I/O Redirection

- Redirection happens **first**, before the command runs. That means shell I/O redirection can appear *anywhere* in the command line, even at the very beginning: **`$ >out date`**
- 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 files are **erased** to **empty** even if the command is **not found** or has **no output**. This truncation happens **before the command runs**.
- You can only redirect what you can ***see***. If you don't ***see*** any output on your screen, adding redirection won't create any.
- Redirection goes into only ***one*** output file.

Lab 5.1: Redirection happens *first*, *before* the command runs

```
[user@localhost ]$ cd ; rm -rf lab5.1 ; mkdir lab5.1 ; cd lab5.1
[user@localhost lab5.1]$ date > foo               (erase foo and then put the date in it)
[user@localhost lab5.1]$ cat foo foo foo          (show the file contents three times)
[user@localhost lab5.1]$ cat foo foo foo >foo     (send foo into foo three times)
[user@localhost lab5.1]$ 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:

```
[user@localhost lab5.1]$ date > foo          (erase foo and then put the date in it)
[user@localhost lab5.1]$ cat foo             (show the file contents on the screen)
```

```
[user@localhost lab5.1]$ cp foo bar >foo       (copy the file with some added redirection)
[user@localhost lab5.1]$ cat bar                       (what is in bar now?)
```

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

Lab 5.2: You can only redirect output that you can see

```
[user@localhost ]$ cd ; rm -rf lab5.2 ; mkdir lab5.2 ; cd lab5.2
[user@localhost lab5.2]$ date >foo                (erase foo and then put the date in it)
[user@localhost lab5.2]$ cat foo                  (show the contents of the file)
[user@localhost lab5.2]$ cp foo bar               (copy command has no output to see)
[user@localhost lab5.2]$ cp foo bar >out          (redirect output of copy command)
[user@localhost lab5.2]$ cat out                  (what is in out now?)
```

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

Lab 5.3: Redirection goes only to one place

```
[user@localhost ]$ cd ; rm -rf lab5.3 ; mkdir lab5.3 ; cd lab5.3
[user@localhost lab5.3]$ date >a           (erase a and then put the date in it)
[user@localhost lab5.3]$ cat a             (show the contents of the file on the screen)
[user@localhost lab5.3]$ date >a >b >c     (erase three files; put date only in last one)
[user@localhost lab5.3]$ cat a             (show content of file a  now)
[user@localhost lab5.3]$ cat b             (show content of file b  now)
[user@localhost lab5.3]$ cat c             (show content of file c  now)
```

a) Explain in **detail** what you see inside the three files and **exactly** how the contents got that way:

Lab 5.4: Only standard output is redirected, by default

```
[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 nofile    (will generate error message for missing file)
[user@localhost lab5.4]$ cat foo nofile >out       (redirect stdout only)
[user@localhost lab5.4]$ cat out
[user@localhost lab5.4]$ cat foo nofile >both 2>&1      (redirect stderr with stdout)
[user@localhost lab5.4]$ cat both
```

b) Explain in **detail** what you see inside the files **out** and **both** and **exactly** how the contents got that way:

Lab 5.5a: 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`**, **`fgrep`**

```
[user@localhost ]$ cd ; rm -rf lab5.5 ; mkdir lab5.5 ; cd lab5.5
[user@localhost lab5.5]$ 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 lab5.5]$ cat myfile
```
*(This is **Output One**.)*

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

```
[user@localhost lab5.5]$ 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 lab5.5]$ cat myfile
```
(This is Output Two.)

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

____

____

Lab 5.5b: Appending to or Overwriting an existing file using > and >>

```
[user@localhost lab5.5]$ date >  bar        (note only one > on this line!)
[user@localhost lab5.5]$ date >> bar
[user@localhost lab5.5]$ date >  bar        (note only one > on this line!)
[user@localhost lab5.5]$ date >> bar
[user@localhost lab5.5]$ date >> bar
```

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

Lab 5.5c: Concatenating multiple files using `cat`

In this section you will be using **`cat`** without any file name argument, which means it will be reading your **keyboard**. Type the given text and use the EOF character to complete the input. The output from **`cat`** will be redirected into the given output file. (Nothing will appear on your screen – look in the file.)

```
[user@localhost lab5.5]$ cat > f1
```
Enter the following keyboard text (and **close** the file as before using EOF): **`Hello everybody`**
```
[user@localhost lab5.5]$ cat > f2
```
Enter the following keyboard text (and **close** the file as before using EOF): **`My name is`** ***`My Name`***
```
[user@localhost lab5.5]$ cat > f3
```
Enter the following keyboard text (and **close** the file as before using EOF): **`Good-bye`**
```
[user@localhost lab5.5]$ cat f1 f2 f3
```
(see all three files – three lines – on your screen)

```
[user@localhost lab5.5]$ cat f1 f2 f3 > f4
```

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

Lab 5.5d: Throwing away output using /dev/null

```
[user@localhost lab5.5]$ date >foo                      (erase foo and then put the date in it)
[user@localhost lab5.5]$ cat foo nofile                 (will generate error message for missing file)
[user@localhost lab5.5]$ cat foo nofile 2>/dev/null     (discard error message)
[user@localhost lab5.5]$ find /etc >out                 (generates some error messages)
[user@localhost lab5.5]$ find /etc >out 2>/dev/null     (discard the error messages)
[user@localhost lab5.5]$ find /etc >/dev/null 2>out     (keep only the error messages)
```

a) What is the purpose of the special output file pathname **`/dev/null`** ?

Lab 5.6a: Using shell pipelines | to connect commands

```
[user@localhost ]$ cd ; rm -rf lab5.6 ; mkdir lab5.6 ; cd lab5.6
[user@localhost lab5.6]$ ls -l /bin                (too much output to see on one screen)
[user@localhost lab5.6]$ 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 lab5.6]$ fgrep 'the' /etc/fstab
[user @localhost lab5.6]$ fgrep 'none' /etc/fstab
[user @localhost lab5.6]$ fgrep 'the' /etc/fstab | fgrep 'none'
```

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

```
[user@localhost lab5.6]$ nl /etc/passwd
[user@localhost lab5.6]$ nl /etc/passwd | head -n 5
[user@localhost lab5.6]$ 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):

Lab 5.6b: Pipes and Redirecting standard input (stdin)

```
[user@localhost lab5.6]$ date >foo
[user@localhost lab5.6]$ cat foo
[user@localhost lab5.6]$ echo hi | cat
```

```
[user@localhost lab5.6]$ echo hi | cat foo
[user@localhost lab5.6]$ echo hi | cat <foo
[user@localhost lab5.6]$ echo hi | ls
[user@localhost lab5.6]$ echo hi | pwd
```

a) Explain in detail why **`hi`** does not show on your screen in the last four commands, above?

```
[user@localhost lab5.6]$ date >foo | wc
```

b) Explain why the output of the above command pipeline is: **`0 0 0`**

Lab 5.6c: The `tr` translate command

```
[user@localhost lab5.6]$ echo 'hello world'
[user@localhost lab5.6]$ echo 'hello world' | tr '[:lower:]' '[:upper:]'
```

a) Explain the function of the **`tr`** translate command, as shown above?

b) Does the **`tr`** translate command accept pathnames as command arguments?

Review: Basic Commands

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

b) **Describe** the purpose of the command: **`tail`**

c) **Describe** the purpose of the command: **`fgrep`** and how it differs from **`grep`**

d) **Describe** the purpose and output of the **`wc`** command:

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

f) 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):

g) Give a command pipeline (two commands) that would show the current month calendar on your screen all in UPPER CASE LETTERS, e.g **`SEPTEMBER`** instead of **`September`**:

h) Do you know the **Four Rules for Output Redirection** from the class notes?

i) Do you know the **Three Rules for Pipes** from the class notes?

j) What is wrong with this command: **`$ sort file >file`**