More Linux Commands

0.1 `wc`

The Linux command for acquiring size statistics on a file is `wc`. This command can provide information from line count, to bytes in a file.

Open up a terminal, make sure you are in your home directory, and run the command.

```
wc ~/public_html/index.html
```

Examine the three numbers outputted. Next run the following command.

```
wc -l ~/public_html/index.html
```

Notice that only one number was outputted. Compare this number with the contents of your `public_html/index.html` file.

Note that your output will likely differ from the following.

```
denver:~$ wc ~/public_html/index.html
6 12 81 public_html/index.html
denver:~$ wc -l ~/public_html/index.html
6 public_html/index.html
denver:~$
```

Question

1 What statistic did the `wc -l` command show?
A. The number of bytes B. The number of lines C. The number of words D. The number of characters

0.2 `cat`

We have used editors to edit files, however, this is tedious if we would just like to view their contents. To catenate a file to the terminal use the `cat` command.

Try it out by issuing the following command, from you home directory.

```
denver:~$ cat public_html/index.html
<html>
<head></head>
<body>
Hello my name is John Doe
</body>
</html>
```

Now use `gedit` to create a file named `test01`, in a new directory, and fill it with some text. Then `cat` the contents of it.

```
denver:~$ mkdir cs192_lab2
denver:~$ gedit cs192_lab2/test01
(now add content to this file, then save and close it)
denver:~$ cat ~/public_html/index.html ~/cs192_lab2/test01
```

Question

2 What does listing more than one file with the `cat` command do?
A. It only displays the first file B. It displays each file separated by the file name
C. It gives an error D. It displays each file in the order they were listed, no separation

0.3 `clear`

As we use commands like `cat`, our console can become very cluttered. It is sometimes helpful to clear the screen. The Linux command for this is `clear` and is demonstrated below.

```
denver:~$ cat ~/public_html/index.html
(lines of output)
denver:~$ clear
(screen now only shows a prompt)
```

0.4 `diff`

Another useful command for Linux is `diff`. This command can be used for testing your program's output with expected output.

Now lets copy over `~/cs192_lab2/test01` to another file. Change the second file, adding one line, and replacing a word in another line to all caps. Then use the `diff` command to compare them.

```
denver:~$ cp ~/cs192_lab2/test01 ~/cs192_lab2/test02
denver:~$ gedit ~/cs192_lab2/test02
(change one line to all caps, add another line; save and quit)
denver:~$ diff ~/cs192_lab2/test01 ~/cs192_lab2/test02
(output of the diff command, showing lines that differ)
```

Now try the same command with the `-i` flag.

```
diff -i ~/cs192_lab2/test01 ~/cs192_lab2/test02
```

Question

3 What does the `-i` flag do to the `diff` command?
A. It is not a valid flag B. It suppresses all output C. It only produces output when files are different D. It ignores lines that only differ in case

0.5 `date`

It is common in shell scripts, and programs, do perform certain actions depending on the date and time. The Linux command for acquiring this information is `date`. Issue the following command (note output will obviously depend on the date, although the format will stay the same).

```
denver:~$ date
Wed Aug 19 23:26:16 MDT 2015
```

Next try issuing the following commands and notice how the output changes.

```
denver:~$ date +"%m"
(output here)
denver:~$ date +"%M"
(output here)
denver:~$ date +"The month is %B and the nano seconds are %N"
(output here)
```

Question

4 What does the plus sign, followed by a quoted string, do for the `date` command?
A. Allows you to specify output format B. It will produce an error C. Extra arguments are ignored D. The provided text will proceed the output of the date command

0.6 `less`

We have seen the `cat` command to look at files, however, a better command, for longer files, is the `less` command. Issue the following command to view information on the CS department's computers using the `less` command.

```
denver:~$ less ~info/machines
(press 'q' to quit the page)
```

The controls for the `less` pager are described below. Note that these controls are the same as the `man` command, because `man` uses `less` to view the manual pages.

Command	Description
ENTER, e, j	Scroll forward one line
y, k	Scroll backwards one line
SPACE, f	Scroll forward one window
b	Scroll backwards one screen
/	Search forward for a pattern
?	Search backwards for pattern
n	Move to next pattern match
h	Display help screen
v	Opens file in text editor, defaults to the `vi` editor
q	Quit

Now let's give two file arguments to `less`.

```
denver:~$ less ~info/machines ~/public_html/index.html
(Inside man page.  Now press :n and see what happens)
denver:~$
```

Question

5 What did pressing `:n` inside `less` do, when the command was invoked with multiple files as arguments?
A. It quit the program B. It moved on to the next file C. It moved back to the previous file D. It paged down to the bottom

0.7 Piping and Redirection

Some commands may produce a lot of output. It is beneficial to have a way to use the output of one command as the input to another. In Linux there is a piping utility that allows this.

Issue the following commands, keeping track of what is changing in the file in question after every command.

```
denver:~$ echo "Hello world" > cs192_lab2/file03
denver:~$ cat cs192_lab2/file03
(output)
denver:~$ date > cs192_lab2/file03
denver:~$ cat cs192_lab2/file03
(output)
```

Question

6 What happened to the file `file03` in the previous example?
A. It was filled with the output of both commands B. It was filled with the output of the first command C. It was filled with the output of the first command, then overwritten with the output of the second command D. An error was given after the second command

Now issue the following two commands and notice how `file03` changes.

```
denver:~$ echo "Hiya, world" >> cs192_lab2/file03
denver:~$ cat cs192_lab2/file03
(output)
```

Question

7 What does using the double greater-than sign (>>) do?
A. It prevents the first file from being overwritten B. It prompts the user with a warning before overwriting the file C. It appended the output of the command to the file given D. It resulted in an error

Lastly it is possible to use the output of one command as the input for another. Issue the following two commands and note the affect.

```
denver:~$ ls /
(output)
denver:~$ ls / | wc
(output)
```

0.8 `grep`

Searching through large files is another task that you will likely be faced with at some time. The command Linux provides to accomplish this is `grep`. This command takes a pattern, followed

by an arbitrary amount of files to search for the given pattern.

Try `grep` out with the following command.

```
denver:~$ grep hp ~info/machines
(lots of output)
denver:~$ grep hp ~info/machines | wc -l
23
denver:~$ grep -i hp ~info/machines
(lots of output, notice what else is being matched now)
denver:~$ grep -i hp ~info/machines | wc -l
552
```

Now try the following command, taking note that the `-n` flag is being used now.

```
grep -n hp ~info/machines
```

Question

8 What did using the `-i` flag appear to do to the `grep` command?
A. It made the search case-insensitive B. It inverted the match, making lines that aren't a match, a match C. It counted the total matches D. It is not a valid flag

9 What does the `-n` flag do to the `grep` command?
A. It displays the line number with the match B. It displays no output C. It displays only matching lines, as they appear in the file D. It is not a valid flag

0.9 `whoami`

Many processes on the computer need to execute differently depending on who is executing it. In Linux the command we use to check which user is logged in is `whoami`. Try it out on your terminal, noting that the response will depend on your user name.

```
denver:~$ whoami
con
```

0.10 `echo`

We have seen a few uses for the `echo` command already, in this lab. Now that we know a few more commands let's see how to interpolate command output into the echo string. Type the following two commands.

```
denver:~$ echo "My user name is whoami"
(output here)
denver:~$ echo "My user name is $(whoami)"
(output here)
```

Question

10 What did surrounding the `whoami` command with `$(...)` do inside the `echo` command?
A. It printed the literal string B. It substituted the output of `whoami` into the echoed string C. It gave an error D. Everything inside the `$(...)` was ignored

0.11 `find`

Another common problem computer scientist face is finding files. Linux provides a very powerful command for this called `find`. This command takes as arguments a base directory, options, and arguments to the options for searching for the file. Issue the following command to search for your `index.html` from your home directory.

```
denver:~$ find .  -name "index.html"
./public_html/index.html
```

Now use the `touch` command to create the file `~/public_html/public_html` then issue the following command.

```
find .  -type f -name "public_html"
```

Question

11 What does the `-type f` option and argument do to the `find` command?
A. Only matches directories B. Gives an error C. Looks for files starting with an 'f'
D. Only matches files (not directories)

0.12 `chmod`

When using the command `ls -l`, one output section is the file permissions. Each file has *owner*, *group*, and *other* permissions (and they appear in that order). The three possible permissions are *read* (worth 4 points), *write* (worth 2 points) and *execute* (or search for directories) (worth 1 point). The sum of any three of these options will always be unique.

To change these permissions, we can use the `chmod` command. Try it out by issuing the two following commands.

```
denver:~$ touch my_new_chmod
denver:~$ chmod 754 my_new_chmod
denver:~$ ls -l my_new_chmod
-rwxr-xr-- 1 con under 0 Sep 6 17:05 my_new_chmod
```

As we can see this file was given read, write, and execute permissions to the owner; read and execute permissions to the group; and read permissions to everyone else.

Question

12 Which set of numbers would be accepted by `chmod` to allow the owner to read, write, and execute the file; the group to only read and execute it; and everyone else to only execute it?
A. 157 B. 764 C. 755 D. 751

Shell Scripting

Creating a Script

It can be useful to group commands into a *script* so that they may be run multiple times. A script is just a collection of commands which are passed to the shell one at a time.

Let's now create a script! Open a terminal, and in the home directory issue the following command.

```
gedit my_script &
```

Inside the `gedit` window type:

```
#!/bin/bash
echo "Hello, this is my first script"
```

Save the file (do not close it) and type the following into your terminal.

```
denver:~$ bash my_script
Hello, this is my first script
denver:~$ chmod +x my_script
denver:~$ ./my_script
Hello, this is my first script
```

Once the script has been written it may be invoked in one of two ways. The first is by calling a shell (such as `bash`) then listing the script file as an argument. The next method is to set the execution bit of the file's permissions. For this method we also have to let the computer know,

in the file, which shell to use. That is what the first line of the script does. The two methods are shown by example above on the file named `my_script`.

Notice that in the second example, when invoking a script in the same directory as you, a dot forward slash proceeds the script name. This is to disambiguate the script from a shell command.

Lastly shell scripts may have comments in them that do not effect the other code. Comments start with a hash (#) sign (usually at the start of a line) and continue to the end of a line.

```
# I am a comment and do not effect the script!
```

Add another line to your script with the following command.

```
echo "The person running this script is $(whoami)"
```

Save your script again, and run it using one of the two methods described above.

Assignment

Create a shell script that utilizes the following commands, with additional flags for at least three, such as `-l` for `ls -l`. Also try to have some logical structure/purpose for the script.

1. `date`
2. `ls`
3. `grep`
4. Use of a pipe, `|`
5. `echo`
6. `whoami`
7. `cat`
8. `wc`
9. `mkdir`
10. `touch`
11. Redirection using `>` to create a file
12. Redirection using `>>` to append to a file