GNU/Linux File System and Basic Commands

0.1 Files, directories, and pwd

The GNU/Linux operating system is much different from your typical Microsoft Windows PC, and probably looks different from Apple OS X. GNU/Linux is an Unix-like operating system meaning it acts similar to traditional Unix systems. Other Unix-like operating systems include Solaris, BSD (and its deviants), and Apple OS X. The Free Software Foundation (FSF) wrote independent software to mimic a Unix system so it could be made free to all users (the word "free" here referring to "freedom", not the monetary "free"). This GNU software along with the Linux kernel, written by Linus Trovalds, makes up the GNU/Linux operating system we will be using.

In GNU/Linux, and other Unix-like operating systems, files are organized hierarchically in directories. Directories themselves are actually files and so is everything else on the computer, even the monitor! (at least they are represented as files). This detail is not very important and it can help to think of directories as folders that hold files, and other directories.

The base directory is called the root directory, denoted with a forward-slash '/' and this holds many other directories. From the root directory we see other directories such as `dev bin home mnt usr` and more. Each of these directories has purposes ranging from the commands on the terminal, to accessing hardware devices (which are represented as files!).

The directory we are concerned about for now is our home directory. On personal computers this is likely located under `/home/USERNAME/` where 'USERNAME' is your login name. On the school computers; however, a more advanced set up is needed to keep track of all the undergrads, graduate students, staff, faculty, and miscellaneous servers. To see where your home directory is located open up a terminal and type the command `pwd` and you will see output similar to this:

```
denver:~$ pwd
/s/bach/l/under/username
denver:~$
```

You are encouraged to try commands as they are introduced as there is little danger with most of the commands, and ones that are dangerous will be flagged as such. Also every section builds on the previous one as we will be creating files and directories so try to enter all of the commands! Your output for some commands will be different but these differences will be noted and likely unimportant.

The above command stands for **p**rint **w**orking **d**irectory and tells you where on the computer you are. When a terminal opens it will usually start you in you *home* directory, which is named after your username. Your output for this command will look similar but leading directories may be slightly different, and your username will replace the emphasized text.

Questions

1 The GNU/Linux operating system is more closely related to which of the following?
A. Microsoft Windows B. Apple OS X

2 In which way are files organized in the GNU/Linux file system?
A. Linearly B. Hierarchically C. Exponentially D. Chronologically

3 Which of the following are treated as files in a GNU/Linux system?
A. Programs B. The monitor C. Text documents D. All of the above E. None of the above

0.2 ls

When working in a directory we often want to view what other files and directories are within your working directory. This can be accomplished with the `ls` command, which is short for **list**. Try it out in your terminal. (Note that your output will depend on what files and directories have already been made in your account.)

```
denver:~$ ls
Desktop Downloads Public public_html Welcome
denver:~$
```

Many terminal emulators will color directories differently from normal text files. Here the directories are blue and the text file appears normal.

It is possible to list the contents of another directory by passing that directory as an argument to `ls`. Arguments are everything that come after the command. To list the contents of our `public_html` directory we created in the last lab we may issue the following command:

```
denver:~$ ls public_html
index.html
```

Note you may have created more files or directories inside of `public_html` which would mean a different output.

There are many useful options we can pass to `ls` to change its behavior. The first one we will try is the long flag. Flags are special arguments that usually begin with a hyphen. Try typing `ls -l` and examine the output. This flag gives us a *long* output.

```
denver:~$ ls -l
total 16
drwxr-xr-x 2 con under 4096 Jul 08 11:07 Downloads
...
```

The output is of the form:
`[file permissions] [links] [owner] [group] [size] [month] [day] [time] [file name]`

Where the month, day, and time refer to when the file was last accessed. Note that your output will have information specific to your files (such as the owner being you).

The first series of characters represent the file *permissions* of the form:
`[directory(d/-)][owner][group][other]`

For `owner, group` and `other` there are three characters, 'r' for *read*, 'w' for *write*, and 'x' for *executing* files, or searching directories. If these permissions are granted for one of the three types the letter will appear, else a hyphen.

Thus in the above example `Downloads` is a directory, the owner can read, write, and search it, members of the same group as the owner may read and search the directory, and everyone else on the system may read and search the directory.

The next option worth mentioning is the *all* flag. Try it out with `ls -a`

```
denver:~$ ls -a
.  .gnome2 Desktop .bash_history .bash_profile .bashrc
..  Documents Downloads
```

This flag shows all of the files. Normally `ls` only shows the non hidden files. Hidden files begin with a period, such as the directory `.gnome2` that you may see. These hidden files are usually configuration files and do not need to be shown normally. Note your output will likely be different from this in both formatting and contents.

The two files which probably look a little weird are the single period, and the double period, which seem to both be directories. This is because they are directories! The period refers to the directory you are in and the double period refers to the parent directory. A parent directory is the directory that holds the directory in question.

Questions

4 What flag is best for displaying hidden files with the `ls` command?
A. `-a` B. `-h` C. `-U` D. `-x`

5 When displaying the hidden file what is the single dot directory?
A. Parent directory B. Home directory C. Current directory D. Root directory
E. It's not a directory

6 What group do you belong to in the CS computers? (Hint: look at the output from `ls -l`)
A. studs B. over C. fresh D. under E. null

0.3 man

We only scratched the surface of all the options for the `ls` command. If we would like to see more we can look at the manual entry. There is a command to do this in the terminal also. Type `man ls` to see its **man**ual entry. This will bring up a whole new screen with lots of information on the `ls` command.

There are nine different sections of `man` pages ranging from programs, library calls, kernel routines and even games (although these have been discontinued on many systems). To specify which section you would like (although this is rarely needed) put the number directly after `man`.

To move down press 'j', to move up press 'k', to page down press `<space>`, and if you'd like to leave the page press 'q' for quit.

```
denver:~$ man ls
(you will now be in a new page until you hit 'q')
denver:~$ man 3 printf
(you will now be in a new page until you hit 'q')
```

You can even call `man man` to see the manual entry for itself!

Questions

7 Which manual section contains information on special files? (Hint: look at the manual entry for `man`.)
A. `1` B. `2` C. `4` D. `8`

8 Which button do you press to search a man page? (Hint: try these in an actual `man` page.)
A. `s` B. `<Control>f` C. `/` D. `f` E. `<alt><space>`

9 How would you display the files of the root directory? (Hint: the first section touched on how the root directory is denoted.)
A. `ls root` B. `ls .` C. `ls /` D. `ls \` E. `ls ~`

0.4 cd

When we open the terminal it starts in the home directory (denoted by a tilda, ~) but sometimes we want to work from a different directory. To **c**hange **d**irectory we use the `cd` command. Try it out by typing `cd Documents` or some other directory.

Remember if you are ever lost and far from home you can type `cd ~` to go home! It is akin to tapping your heels three times and saying "There's no place like home."

In the last section we mentioned that period and double period were directories, this means that we can `cd` to them.

```
denver:~$ cd public_html
denver:~/public_html$ pwd
/s/bach/l/under/con/public_html
denver:~/public_html$ cd .
denver:~/public_html$ pwd
/s/bach/l/under/con/public_html
denver:~/public_html$ cd ..
denver:~$ pwd
/s/bach/l/under/con
denver:~$
```

Your output of `pwd` will depend on your account.

Don't forget to `man cd` to check out all of the options.

Questions

10 How would you change to the directory two above the current directory (assuming there are at least two above you)?
A. `cd /home/USERNAME` B. `cd ../` C. `cd ../../` D. `cd .../`

11 Which command changes you to the directory you were just in?
A. `cd $LAST` B. `cd ../` C. `cd -` D. `cd $BACK` E. `undo`

12 What happens if you try to `cd` to a normal text file?
A. It will be treated as a directory by `cd` and be entered B. An error is given

0.5 touch

Whenever a file is modified/accessed on a computer its time-stamp is updated. We saw some time-stamps when we ran the command `ls -l` which told us the last time the file was modified.

Programs can read these time-stamps and perform certain actions on files if they are older or younger than a certain time. Sometimes we want to update the time stamps without opening the file. We do this with the `touch` command. To use it type `touch` then the name of a file.

If the file that is named does not exist, an empty file is created. Thus if you wanted to create an empty Java source file (program source files are just plain text files!) for a class you could enter:

```
denver:~$ ls
Downloads Documents public_html Welcome
denver:~$ touch P3.java
denver:~$ ls
Downloads Documents public_html P3.java Welcome
```

Don't forget to `man touch` to check out all of the options.

Questions

13 What happens if the `touch` command is used on a file that already exists?
A. The old file is deleted in place of a new one B. Nothing C. The command gives an error D. The files time stamp gets updated

14 Is it possible to set the time stamp of a file to sometime previous to its current time stamp using the `touch` command? (Hint: look in the manual page.)
A. No, GNU/Linux systems protect against this B. No, `touch` does not have any additional flags C. Yes, with the aid of a time machine D. Yes, using the correct flag

15 What will the command `touch man` do?
A. Update the time stamp of the `man` command B. Give an error because you can not call two commands at once C. It is the same as the `man touch` command D. It will create a file named `man` (or update its times stamp if it already exists) E. It will try to overwrite the `man` program but fail

0.6 mkdir

We have seen how to make files from the terminal, the command `mkdir` will **m**a**k**e **dir**ectories.

To organize files for all your computer science classes we could create the following directories:

```
denver:~$ ls
Downloads Documents public_html Welcome
denver:~$ mkdir classes
denver:~$ ls
classes Downloads Documents public_html Welcome
denver:~$ cd classes
denver:~/classes$ pwd
/s/bach/l/under/con/classes
denver:~/classes$ mkdir cs192 math160 cs160
denver:~/classes$ ls
cs192 cs160 math160
```

Notice we can give more than one argument to `mkdir` and it will create a directory for each of them. Organizing your files is a personal choice, however, like cleaning your room, if you neglect it too long it becomes a large clean up chore!

Don't forget to `man mkdir` to check out all of the options.

Questions

16 How would you create two directories in one command where one directory is created inside the other? (Hint: try them out!)
A. `mkdir one two` B. `mkdir -n one two` C. `mkdir -n one/two`
D. `mkdir -p one/two` E. It is not possible

17 Like the `touch` command, will using `mkdir` on an existing directory update its time stamp?
A. No, it will give an error B. No, it will overwrite the old one C. No, directories do not have time stamps D. Yes

18 Can you create directories anywhere on the computer with `mkdir`?
A. Yes, but only if you use the `-a` flag B. Yes, but only if you have permissions to do so
C. No, `mkdir` only works on directories under current working directory D. No, `mkdir` only lets you create directories under your home directory

0.7 cp

To **co**p**y** a file to a new file we use the `cp` command. This command takes two arguments, first the file to copy, then the place to copy it to.

Copying files is useful if you want to create backups, or preserve a document while having a copy to edit. Lets say we want to create a backup of a very important CS192 paper that we are working on. (Note: we will use the `touch` command to create the document before copying it. If the file already existed, as would make more sense for this scenario, we could skip the `touch` command.)

```
denver:~$ ls
Downloads Documents classes public_html Welcome
denver:~$ cd classes/cs192
denver:~/classes/cs192$ pwd
/s/bach/l/under/con/classes/cs192
denver:~/classes/cs192$ ls
denver:~/classes/cs192$ touch important_paper.tex
denver:~/classes/cs192$ ls
important_paper.tex
denver:~/classes/cs192$ cp important_paper.tex important_paper.tex.bak
denver:~/classes/cs192$ ls
important_paper.tex important_paper.tex.bak
```

In this example we gave file name another extension, this does nothing special to file except change the name.

We can copy a file to a directory also, and in this case we can omit the name and it will keep its original name. Download this PDF from:

`http://www.cs.colostate.edu/~cs192/.Fall15/assignments/linux_lab_1.pdf`

If we enter that link in Firefox we will see this PDF. There should be button on the upper right hand corner of the embedded PDF viewer that looks like a sheet of paper with a down

arrow on it. Click that button to download the PDF, and make sure to select the `Save File` radio button before clicking `OK`. This will save the PDF in the `Downloads` directory. Lets copy it over to our `cs192` directory!

```
denver:~$ ls
Downloads Documents classes public_html Welcome
denver:~$ ls Downloads
linux_lab_1.pdf
denver:~$ cp Downloads/linux_lab_1.pdf classes/cs192/
denver:~$ ls classes/cs192/
important_paper.tex important_paper.tex.bak linux_lab_1.pdf
```

Don't forget to `man cp` to check out all of the options.

Questions

19 If a file already exists at the destination of a `cp` will it be overwritten (assume no flags)?
A. Yes, but there will be a prompt B. Yes, so be careful! C. No, it will give an error
D. Yes, but a back up will be created with a `.bak` extension

20 What flag would you use to copy a directory and all its contents?
A. `-p` B. `-l` C. `-R` D. `-C` E. `-i`

21 Which of the following commands makes use of an arbitrary number of arguments? (Hint: try the commands with different amounts of arguments.)
A. `cd` B. `mkdir` C. `touch` D. `A and B` E. `B and C` F. None

0.8 mv

The command to **m**o**v**e a file is `mv` which takes two arguments, first the file to be moved, next where to move it. This can be used to rename files as well as moving them around.

We can see, already, our `cs192` directory is getting a little cluttered. Lets create more directories to better organize it, then move the current files to their new directories. (Note that `ls` output may be ordered slightly different on your terminal.)

```
denver:~$ ls
Downloads Documents classes public_html Welcome
denver:~$ cd classes/cs192
denver:~/classes/cs192$ ls
important_paper.tex important_paper.tex.bak linux_lab_1.pdf
denver:~/classes/cs192$ mkdir pdfs papers backups
denver:~/classes/cs192$ ls
backups important_paper.tex important_paper.tex.bak linux_lab_1.pdf papers
pdfs
denver:~/classes/cs192$ mv important_paper.tex papers
denver:~/classes/cs192$ mv important_paper.tex.bak backups/paper.bak
denver:~/classes/cs192$ mv linux_lab_1.pdf pdfs
denver:~/classes/cs192$ ls
backups papers pdfs
denver:~/classes/cs192$ ls backups
paper.bak
denver:~/classes/cs192$ ls papers
important_paper.tex
denver:~/classes/cs192$ ls pdfs
linux_lab_1.pdf
```

Notice that when `important_paper.tex.bak` was moved, it was also renamed. This is a very useful way to rename files!

Don't forget to `man mv` to check out all of the options.

Questions

22 Is it possible to create a backup of the destination file when using `mv`? (Hint: read the manual entry.)
A. No, `mv` is a primitive command B. No, only `cp` can do that C. Yes, but this is done automatically D. Yes, with the appropriate flag

23 Is `mv` the only way to rename files on the schools GNU/Linux systems? (Hint: try the commands out!)
A. No, `cp` then `rm` the source file B. No, there is a `rename` command C. No, it can be done in the GUI file manager D. All of the above E. Yes, `mv` is the only way

24 What happens when both the `-i` and `-n` flags are used with `cp`? (Hint: look in the manual page.)
A. An error is given B. Only the first takes effect C. Only the last takes effect D. They both take effect E. There are no flags for `cp`

0.9 rm

The command to **rem**ove is `rm` which takes an arbitrary number of arguments, all of which will be removed from the computer (except for directories). This is *not* the same as dragging files

to the trash in Windows; once removed the files are gone. One should use extreme caution with this command.

Some of the flags for `rm` make it even more dangerous. Make sure you know what your command will do before issuing it.

If we now wanted to remove the backup we created for the paper we can use `rm`!

```
denver:~$ ls
Downloads Documents classes public_html Welcome
denver:~$ ls classes/cs192/backups
paper.bak
denver:~$ rm classes/cs192/backups/paper.bak
denver:~$ ls classes/cs192/backups
denver:~$
denver:~$ echo "Oh no!  The backup is gone"
Oh no!  The backup is gone
```

Don't forget to `man rm` to check out all of the options.

0.10 rmdir

The command to **rem**ove **dir**ectory is `rmdir` which takes an arbitrary amount of arguments, all of which will be removed from the computer if they are *empty* directories.

We previously removed the only file from our `backups` directory, let us now remove the directory too.

```
denver:~$ ls
Downloads Documents classes public_html Welcome
denver:~$ ls classes/cs192/
backups papers pdfs
denver:~$ rmdir classes/cs192/backups
denver:~$ ls classes/cs192
papers pdfs denver:~$
```

Don't forget to `man rmdir` to check out all of the options.
Questions

25 Is the `rm` command similar to dragging something to the trash on other OS's?
A. No, `rm` removes the file permanently B. Yes, but the recycling bin for `rm` is only emptied when space is needed in GNU/Linux C. Yes, but files can be accessed with the `recover` command D. Yes, the file will be located in the recycling bin

26 Will using `rm` on a directory, without any flags, remove everything in it?
A. No, only empty directories can be removed with `rm` B. No `rm` can not remove directories C. Yes, so be careful D. Yes, but only with the appropriate flags

27 What command mimics `rmdir`?
A. `rm -f` B. `rm -d` C. `unmkdir` D. None, `rmdir` is the only command to remove directories

0.11 echo

We saw in the `rm` section the command **echo** which echoes back whatever comes after it. Quotes are not always necessary, however, they help to remove special meaning from certain characters. There are two different types of quotes, which act differently. The double quotes (") will expand shell variables within them, but the single quotes (') will not. We will cover shell variables in the next lab, for now just know shell variables start with a dollar sign and hold useful information.

```
denver:~$ echo hello world
hello world
denver:~$ echo "my home is $HOME"
my home is /s/bach/l/under/con
denver:~$ echo 'my home is $HOME'
my home is $HOME
denver:~$ echo Hello, twitter #csRocks
Hello, twitter
denver:~$ echo "Hello, twitter #csRocks"
Hello, twitter #csRocks
```

We will talk about what the hashtag does later, for now just notice that when using special characters it is best to use quotes.

Don't forget to `man echo` to check out all of the options.

Questions

28 Does using single and double quotes with the `echo` command have the same effect?
A. No, single quotes expand shell variables B. No, double quotes expand shell variables C. Yes, but convention suggests only using double quotes D. Yes, but convention suggests only using single quotes

29 What is produced by the following command? (Hint: type this into the terminal, then google "ascii hex table" for a little understanding.)
`echo -e "\x6a"`
A. The *exact* string; `\x6a` B. An unprintable character C. The character 'j' D. The number 106

0.12 Editing files

We have seen how to make and remove files, now lets look at ways to edit them. We can edit files with a text editor. The easiest text editor to use is `gedit`. This editor is a lot like `notepad` and the controls are very simple. We can open a file with gedit on the terminal with `gedit` *`file`* however this will mean we cannot use our terminal until we are done editing it. It is better to put an ampersand after the command so we can continue using the terminal.

If we want to update our webpage we can use `gedit`.

```
denver:~$ ls
Downloads Documents classes public_html Welcome
denver:~$ ls public_html
index.html
denver:~$ gedit public_html/index.html &
[1] 16534
denver:~$
```

The ampersand forks the process into a background subshell. The number that is in the square brackets is the variable that holds the second number. This is so one could kill the process if need be.

Don't forget to `man gedit` to check out all of the options.

Questions

30 What happens when the gedit command is invoked with a filename that does not exist?
A. An error is given and gedit does not open B. A new file is created with the name that was given C. An empty file is opened and created only when the file is saved

31 Is it possible to change the encoding of characters that `gedit` uses?
A. No, it can only encode its files in ASCII B. No, it can only encode its files in UTF-8
C. Yes, if the appropriate flag is given

32 What flag would one use to set the size and location of `gedit`?
A. `--window-size` B. `-g` C. `-w` D. `--set-location` E. A and D

0.13 Shell Globbing

It is common to invoke a shell command on every file in a directory, or at least every file of a given type. One example is when trying to compile multiple Java files in a directory. It becomes tedious to type out every file name, and as discussed computer scientists are quite lazy.

The way to group files together is called globbing, or filename expansion. These filename expansions are similar *regular expressions* but, as you will see in the next lab, are not the same.

First we will need to download two Java files. Navigate to this page on Firefox (the current assignment):

http://www.cs.colostate.edu/~cs192/.Fall15/assignments/lLinuxFileSystemLab.php

In the instructions, under item two, there should be two links to two different Java source files. For both of the files click on the link, then right click on the page that shows the source files, select `Save Page As..`, then save the file (maintaining its name) to your home directory.

```
denver:~$ ls
Hello.java Downloads Documents classes public_html Sum.java Welcome
denver:~$ mkdir java_fun
denver:~$ ls
Hello.java Downloads Documents classes java_fun public_html Sum.java
Welcome
denver:~$ mv *.java java_fun
denver:~$ ls
Downloads Documents classes java_fun public_html Welcome
denver:~$ cd java_fun
denver:~/java_fun$ ls
Sum.java Hello.java
denver:~/java_fun$ javac *.java
denver:~/java_fun$ ls
Sum.java Sum.class Hello.java Hello.class
denver:~/java_fun$ java Hello
Hello world
denver:~/java_fun$ java Sum
Enter a number:  20
Enter another number:  30
The sum of 20 and 30 is 50
```

Here the `*.java` matches anything that ends with the characters `.java` (in this case `Sum.java` `Hello.java`) but the star can be used at the end too! The asterisk expands to anything, or nothing. Thus `st*` would match "star", "staff", "st" but not "rest" because the star comes at the end.

In bash the asterisk expands to fill an arbitrary number of characters. For more information look in `man bash` and search for the section titled "Pathname Expansion" (remember how to search?)

Here is a brief overview of a few types of filename expansions.

Special Character	Meaning
`?`	Wild card for matching any single character
`*`	Matches any sequence of characters
`[`*`list of characters`*`]`	Matches one character from the list
\	Escape (remove special meaning) from the next special character

Questions

For the following questions assume the commands are to be run in a directory containing the following files (Hint: create a directory and `touch` all these files so you can actually run these commands.):

```
aac abc acc aabc main.c this.c this.h thxs.p thrs.c photo128.jpg photo129.jpg
photography.pdf photo130.jpg
```

33 What command would remove only the photos from this directory? (assume all files with the `.jpg` extension are files)
A. `rm phot*` B. `rm *jpg` C. `rm photo12*`

34 How many files would this command display? `echo th?s.?`
A. One B. Two C. Three D. Four E. None, the echo command needs quotes around the argument

35 What would the following command display (order be slightly different) `echo` $a[ab]c$
A. `aac abc acc` B. `aabc` C. `aac abc` D. `aac abc aac aabc` E. Nothing

0.14 Tab completion

Computer scientists and programmers tend to be very lazy, as such we create ways to do things with as little buttons pushed as possible. Using the tab button will attempt to fill in the rest of a file name, and if multiple exist will fill as much as possible. A second tab press when multiple files exist will print a list of the files to the terminal.

```
denver:~$ ls
Downloads Documents dir_one dir_two Welcome
denver:~$ cd d*tab*ir_ *tab*
dir_one dir_two
denver:~$ cd dir_
```

The more one uses this the better one begins to understand the behavior.