Lesson 2: Files and Directories

Challenge 1

Many ways to do the same thing - absolute vs relative paths

Starting from a filesystem location of `/Users/amanda/data/` , which of the following commands could Amanda use to navigate to her home directory, which is `/Users/amanda` ?

1. `cd .`
2. `cd /`
3. `cd /home/amanda`
4. `cd ../..`
5. **`cd ~`**
6. `cd home`
7. **`cd ~/data/..`**
8. **`cd`**
9. **`cd ..`**

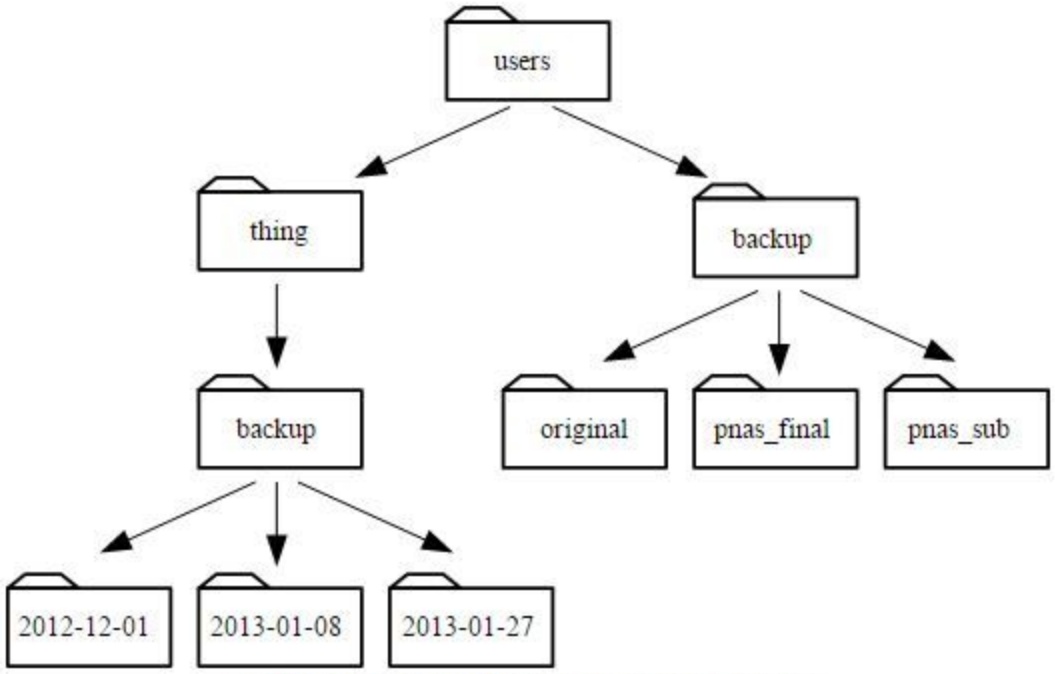

Figure: File System for Challange Questions

Challenge 2

Relative path resolution

Using the filesystem diagram below, if `pwd` displays `/Users/thing` , what will `ls ../backup` display?

1. `../backup: No such file or directory`
2. `2012-12-01 2013-01-08 2013-01-27`
3. `2012-12-01/ 2013-01-08/ 2013-01-27/`

4. **`original pnas_final pnas_sub`**

File System for Challenge Questions

Challenge 3

`ls` reading comprehension

Assuming a directory structure as in the above Figure (File System for Challenge Questions), if `pwd` displays `/Users/backup`, and `-r` tells `ls` to display things in reverse order, what command will display:

```
pnas_sub/ pnas_final/ original/
```

1. `ls pwd`
2. `ls -r -F`
3. `ls -r -F /Users/backup`
4. **Either #2 or #3 above, but not #1.**

Challenge 4

Exploring more `ls` arguments

What does the command `ls` do when used with the `-s` and `-h` arguments?

Answer: -s size ; -h human readable

http://explainshell.com/

Answer: Try `man ls` to get help about `ls`. You read about what `-s` and `-h` do.

Lesson 3: Creating Things

Challenge 1

Renaming files

Suppose that you created a `.txt` file in your current directory to contain a list of the statistical tests you will need to do to analyze your data, and named it: `statstics.txt`

After creating and saving this file you realize you misspelled the filename! You want to correct the mistake, which of the following commands could you use to do so?

1. `cp statstics.txt statistics.txt`
2. **`mv statstics.txt statistics.txt`**
3. `mv statstics.txt .`
4. `cp statstics.txt .`

Challenge 2

Moving and Copying

What is the output of the closing `ls` command in the sequence shown below?

```
$ pwd
/Users/jamie/data
$ ls
proteins.dat
$ mkdir recombine
$ mv proteins.dat recombine
$ cp recombine/proteins.dat ../proteins-saved.dat
$ ls
```

1. `proteins-saved.dat recombine`
2. **`recombine`**
3. `proteins.dat recombine`
4. `proteins-saved.dat`

Challenge 3

Organizing Directories and Files

Jamie is working on a project and she sees that her files aren't very well organized:

```
$ ls -F
analyzed/  fructose.dat    raw/   sucrose.dat
```

The `fructose.dat` and `sucrose.dat` files contain output from her data analysis. What command(s) covered in this lesson does she need to run so that the commands below will produce the output shown?

```
$ ls -F
analyzed/   raw/
$ ls analyzed
fructose.dat    sucrose.dat
```

To get the necessary output she could run `mv *.dat analyzed`

```
Move 2 dat files to analyzed folder
```

Challenge 4

Copy with Multiple Filenames

What does `cp` do when given several filenames and a directory name, as in:

```
$ mkdir backup
$ cp thesis/citations.txt thesis/quotations.txt backup
```

`Answer:` `cp` will copy both files in `thesis` to `backup`

What does `cp` do when given three or more filenames, as in:

```
$ ls -F
intro.txt    methods.txt    survey.txt
$ cp intro.txt methods.txt survey.txt
```

Answer:

`cp` throws an error because it doesn't know where to put the `*.txt` files

Challenge 5

Listing Recursively and By Time

The command `ls -R` lists the contents of directories recursively, i.e., lists their sub-directories, sub-sub-directories, and so on in alphabetical order at each level. The command `ls -t` lists things by time of last change, with most recently changed files or directories first. In what order does `ls -R -t` display things?

Answer: `ls -R -t` will list directories by time and recursively

Lesson 4: Pipes and filters

Using wildcards

When run in the `molecules` directory, which `ls` command will produce this output?

```
ethane.pdb methane.pdb
```

1. `ls *t*ane.pdb`
2. `ls *t?ne.*`
3. **`ls *t??ne.pdb`**
4. `ls ethane.*`

Challenge 2:

What does `sort -n` do?

If we run `sort` on this file:

```
10
2
19
22
6
```

the output is:

```
10
19
2
22
6
```

If we run `sort -n` on the same input, we get this instead:

```
2
6
10
19
22
```

Explain why `-n` has this effect.

Answer:

`sort` with out `-n` will assume that what you are sorting are characters. When sorting characters, "10" comes before "2" because "1" come before "2". In other words, you can think of sort as sorting on the first character and then

on the second character, etc. So "300000" would come before "8" because "3" comes before "8". With the `-n` flag, sort considers 10 a number rather than a character string and sorts the numbers as you'd expect.

Challenge 3:

What does < mean?

What is the difference between:

```
wc -l < mydata.dat
```

and:

```
wc -l mydata.dat
```

`Answer:` They both take in `mydata.dat` as input, but one command take it as an argument and the other takes it as standard input.

Challenge 4:

What does >> mean?

What is the difference between:

```
echo hello > testfile01.txt
```

and:

```
echo hello >> testfile02.txt
```

Hint: Try executing each command twice in a row and then examining the output files.

Answer:

`echo hello > testfile01.txt` will always overwrite `testfile01.txt` while `echo hello >> testfile02.txt` will create the file if it doesn't exist and append to the file otherwise.

Challenge 5

Piping commands together

In our current directory, we want to find the 3 files which have the least number of lines. Which command listed below would work?

1. `wc -l * > sort -n > head -n 3`
2. `wc -l * | sort -n | head -n 1-3`
3. `wc -l * | head -n 3 | sort -n`
4. **`wc -l * | sort -n | head -n 3`**

Challenge 6:

Why does `uniq` only remove adjacent duplicates?

The command `uniq` removes adjacent duplicated lines from its input. For example, if a file `salmon.txt` contains:

```
coho
coho
steelhead
coho
steelhead
steelhead
```

then `uniq salmon.txt` produces:

```
coho
steelhead
coho
steelhead
```

Why do you think `uniq` only removes *adjacent* duplicated lines? (Hint: think about very large data sets.) What other command could you combine with it in a pipe to remove all duplicated lines?

Answer:

Removing adjacent duplicates is completely in-line with the Unix philosophy: build small programs that do things really well and combine them to do bigger things. Combining `uniq` and `sort` will return only the unique fish in the list.

Uniq - Discard all but one of successive identical lines from INPUT (or standard input), writing to OUTPUT (or standard output).

Challenge 7:

Pipe reading comprehension

A file called `animals.txt` contains the following data:

```
2012-11-05,deer
2012-11-05,rabbit
2012-11-05,raccoon
2012-11-06,rabbit
2012-11-06,deer
2012-11-06,fox
2012-11-07,rabbit
2012-11-07,bear
```

What text passes through each of the pipes and the final redirect in the pipeline below?

```
cat animals.txt | head -n 5 | tail -n 3 | sort -r > final.txt
```

`Answer final.txt should contain:`

2012-11-06,rabbit
2012-11-06,deer
2012-11-05,raccoon

Challenge 8:

Pipe construction

For the file `animals.txt` from the previous exercise, the command:

```
$ cut -d , -f 2 animals.txt
```

produces the following output:

```
deer
rabbit
raccoon
rabbit
deer
fox
rabbit
bear
```

What other command(s) could be added to this in a pipeline to find out what animals the file contains (without any duplicates in their names)?

Answer: `cut -d , -f 2 animals.txt | sort  | uniq`

Lesson 5: Loops

Challenge 1:

Variables in Loops

Suppose that `ls` initially displays:

```
fructose.dat    glucose.dat   sucrose.dat
```

What is the output of:

```
for datafile in *.dat
do
    ls *.dat
done
```

Now, what is the output of:

```
for datafile in *.dat
do
  ls $datafile
done
```

Why do these two loops give you different outputs?

Answer:

You can try this out using the following commands

```
touch fructose.dat glucose.dat sucrose.dat
for datafile in *.dat; do ls *.dat; done
```

it prints

```
fructose.dat   glucose.dat sucrose.dat

fructose.dat glucose.dat sucrose.dat
fructose.dat glucose.dat sucrose.dat
```

because each time `ls *.dat* displays all the files with a `.dat` extension.

```
for datafile in *.dat; do ls $datafile; done
```

prints

```
fructose.dat
glucose.dat
sucrose.dat
```

because each iteration of the loop prints one item in `*.dat` which expands to the three `.dat` files.

Challenge 2:

Saving to a File in a Loop - Part One

In the same directory, what is the effect of this loop?

```
for sugar in *.dat
do
    echo $sugar
    cat $sugar > xylose.dat
done
```

1. Prints `fructose.dat`, `glucose.dat`, and `sucrose.dat`, and the text from`sucrose.dat` will be saved to a file called `xylose.dat`.
2. **Prints `fructose.dat`, `glucose.dat`, and `sucrose.dat`, and the text from all three files would be concatenated and saved to a file called `xylose.dat`.**

3. Prints `fructose.dat`, `glucose.dat`, `sucrose.dat`, and `xylose.dat`, and the text from `sucrose.dat` will be saved to a file called `xylose.dat`.
4. None of the above.

Challenge 3:

Saving to a File in a Loop - Part Two

In another directory, where `ls` returns:

```
fructose.dat    glucose.dat   sucrose.dat   maltose.txt
```

What would be the output of the following loop?

```
for datafile in *.dat
do
    cat $datafile >> sugar.dat
done
```

1. **All of the text from `fructose.dat`, `glucose.dat` and `sucrose.dat` would be concatenated and saved to a file called `sugar.dat`.**
2. The text from `sucrose.dat` will be saved to a file called `sugar.dat`.
3. All of the text from `fructose.dat`, `glucose.dat`, `sucrose.dat` and`maltose.txt` would be concatenated and saved to a file called `sugar.dat`.
4. All of the text from `fructose.dat`, `glucose.dat` and `sucrose.dat` would be printed to the screen and saved to a file called `sugar.dat`

Challenge 4:

Doing a Dry Run

Suppose we want to preview the commands the following loop will execute without actually running those commands:

```
for file in *.dat
do
  analyze $file > analyzed-$file
done
```

What is the difference between the two loops below, and which one would we want to run?

```
# Version 1
for file in *.dat
do
  echo analyze $file > analyzed-$file
done

# Version 2
for file in *.dat
do
  echo "analyze $file > analyzed-$file"
done
```

Answer:

Version 1 would echo analyze $file and save the result to the file analyzed-$file. We would not see anything on the screen

Version 2 is the one we want and it would print the expanded string "analyze $file > analyzed-$file" to the console for each iteration of the for loop.

Challenge 5:

Nested Loops

Suppose we want to set up up a directory structure to organize some experiments measuring the growth rate under different sugar types *and* different temperatures. What would be the result of the following code:

```
for sugar in fructose glucose sucrose
do
    for temperature in 25 30 37 40
    do
        mkdir $sugar-$temperature
    done
Done
```

Answer:

First, a sugar is extracts from [fructose, glucose, sugar]. For this sugar four directories are made that have the pattern $sugar-$temperature where $temperature is on of the 4 temps given. This is repeated for each sugar so that 12 new directories are made.

Lesson 6: Shells Scripts

Challenge 1:

Variables in shell scripts

In the `molecules` directory, imagine you have a shell script called `script.sh` containing the following commands:

```
head -n $2 $1
tail -n $3 $1
```

While you are in the `molecules` directory, you type the following command:

```
bash script.sh '*.pdb' 1 1
```

Which of the following outputs would you expect to see?

1. All of the lines between the first and the last lines of each file ending in `.pdb` in the`molecules` directory
2. The first and the last line of each file ending in `.pdb` in the `molecules` directory
3. The first and the last line of each file in the `molecules` directory
4. An error because of the quotes around `*.pdb`

Challenge 2:

List unique species

Leah has several hundred data files, each of which is formatted like this:

```
2013-11-05,deer,5
2013-11-05,rabbit,22
2013-11-05,raccoon,7
2013-11-06,rabbit,19
2013-11-06,deer,2
2013-11-06,fox,1
```

```
2013-11-07,rabbit,18
2013-11-07,bear,1
```

Write a shell script called `species.sh` that takes any number of filenames as command-line parameters, and uses `cut`, `sort`, and `uniq` to print a list of the unique species appearing in each of those files separately.

Challenge 3:

Find the longest file with a given extension

Write a shell script called `longest.sh` that takes the name of a directory and a filename extension as its parameters, and prints out the name of the file with the most lines in that directory with that extension. For example:

```
$ bash longest.sh /tmp/data pdb
```

would print the name of the `.pdb` file in `/tmp/data` that has the most lines.

Challenge 4:

Why record commands in the history before running them?

If you run the command:

```
history | tail -n 5 > recent.sh
```

the last command in the file is the `history` command itself, i.e., the shell has added`history` to the command log before actually running it. In fact, the shell *always* adds commands to the log before running them. Why do you think it does this?

Challenge 5:

Script reading comprehension

Joel's `data` directory contains three files: `fructose.dat`, `glucose.dat`, and`sucrose.dat`. Explain what a script called `example.sh` would do when run as `bash example.sh *.dat` if it contained the following lines:

```
# Script 1
echo *.*
```

```
# Script 2
for filename in $1 $2 $3
do
```

```
    cat $filename
done

# Script 3
echo $@.dat
```

Lesson 7: Finding Things

Challenge 1

Using grep

```
The Tao that is seen
Is not the true Tao, until
You bring fresh toner.

With searching comes loss
and the presence of absence:
"My Thesis" not found.

Yesterday it worked
Today it is not working
Software is like that.
```

From the above text, contained in the file `haiku.txt`, which command would result in the following output:

```
and the presence of absence:
```

1. `grep "of" haiku.txt`
2. `grep -E "of" haiku.txt`
3. `grep -w "of" haiku.txt`
4. `grep -i "of" haiku.txt`

Challenge 2

`find` pipeline reading comprehension

Write a short explanatory comment for the following shell script:

```
wc -l $(find . -name '*.dat') | sort -n
```

Challenge 3

Matching `ose.dat` but not `temp`

The `-v` flag to `grep` inverts pattern matching, so that only lines which do *not* match the pattern are printed. Given that, which of the following commands will find all files in `/data` whose names end in `ose.dat` (e.g., `sucrose.dat` or `maltose.dat`), but do *not* contain the word `temp`?

1. `find /data -name '*.dat' | grep ose | grep -v temp`
2. `find /data -name ose.dat | grep -v temp`
3. `grep -v "temp" $(find /data -name '*ose.dat')`
4. None of the above.

Challenge 4

Little Women

You and your friend, having just finished reading *Little Women* by Louisa May Alcott, are in an argument. Of the four sisters in the book, Jo, Meg, Beth, and Amy, your friend thinks that Jo was the most mentioned. You, however, are certain it was Amy. Luckily, you have a file `LittleWomen.txt` containing the full text of the novel. Using a `for` loop, how would you tabulate the number of times each of the four sisters is mentioned? Hint: one solution might employ the commands `grep` and `wc` and a `|`, while another might utilize `grep` options.