EPBI 414 (Fall 2016) - Assignment 5

Unix / Linux Command Line

Overview

This homework consists of two parts. For the first part, you'll answer some questions about the material covered this week. Then, you'll write a shell script to process some text data, and you'll improve it by changing some of the commands.

Logistics

Submit your assignment in a .zip file labeled in the following matter:

<Case ID>_EPBI414_Fall2016_A5.zip

So, if your Case ID is tar9, you would submit the following zip file:

tar9_EPBI414_Fall2016_A5.zip

Part 1

Submit a written document with your answers to the following questions.

1. Log into `hal` using SSH. In your home directory, create a subdirectory called `epbi414_hw3`. Change to that directory. Then, create a new subdirectory within `epbi414_hw3`, called `hw3_1`. Change to that directory. Then, created a final subdirectory within `hw3_1` called `hw3_1_2`. Change to that directory. Write the commands you used to accomplish this. [6 points]

2. Go back to your home directory, and remove the directory `epbi414_hw3`. Next, using a single command, create the same directory structure that you had before (`~/epbi414_hw3/hw3_1/hw3_1_2/`). Using a single command, change to the lowest level directory (`hw3_1_2`). Write the commands you used to accomplish this. [6 points]

3. Within `hw3_1_2`, create a text file called `greetings.txt` with the text "Hello, EPBI 414" on the first line. Do not use a text editor to do this - accomplish it from the command line! Then, change the permissions of the file so that the owner can read and write the file and the group can read the file; all other users should be denied all permissions. Write the commands you used to accomplish this. [6 points]

4. Go back to your home directory, and then go one level up from there. Generate a list (with file permissions) of all the files in your home directory and all its subdirectories (hint: you will need to use recursion with `ls`; consult the manual page for help). Use a redirect to save this file to `epbi414_hw3`, with the name `home_dir.lst`. Write the

commands you used to accomplish this. [6 points]

5. Download the file `list_of_jedi.txt` off of Blackboard. Since there's a new Star Wars movie coming out this Christmas, we'll use some data from the Expanded Universe for homework. Using SFTP or SCP, move this file to your home directory on `hal`. Then, place it in the directory `epbi414_hw3`, and set the file's permissions to `rw-r-----`. Using a single command, count how many lines there are in the file and store this count value into a file named `jedi_line_count.txt` in the same directory. Write the commands you used to accomplish this (starting with moving the file from your home directory and ending with producing the line count). [6 points]

6. Using a single command on the command line, take the last 15 lines of the `list_of_jedi.txt` file. Sort those lines and output them into a file called `sorted_jedi_subset.txt`. Write the command you used to accomplish this. [6 points]

7. Finally, use the `date` command to append the date you generated the file `sorted_jedi_subset.txt` to the end of the file. Write the command you used to accomplish this. [4 points]

8. Using the `zip` command, create a zip file containing the `epbi414_hw3` directory, called `epbi414_hw3.zip`. Copy this to your local computer, and include it in your homework submission. Be sure that your zip file contains all the subdirectories (you will need to ensure it is recursive).

Part 2

The next part will require you to use the program `sed`, which is a **s**tream **ed**itor. `sed` is a very powerful program that can help you manipulate text files easily, but it can have a tough learning curve. You should expect to do some research on how it works in order to solve this problem.

Download the file `database_query_results.txt` from Blackboard and copy it to your home directory on `hal`. If you look in this file, you'll see that it appears to contain multiple query results from different databases, but the ordering of the lines is all jumbled. This is fairly common when you use a command-line program to execute multiple queries in parallel (i.e. going out to more than one server at the same time) - since queries can take differing amounts of time to complete, when they come back and are written to the file, you can end up with lines that are out of order.

You need to turn this into a simple, comma-separated value list of the databases and the dates. Fortunately, this is actually a pretty easy task with `sed`. For this part, you need to write a simple shell script which will take in a file like `database_query_results.txt` and will produce a date-stamped, comma-separated values file of just the data you want.

Your script must do the following:

- It must be reasonably ***commented*** so that it is documented. You will lose points if you do not comment.

- It must read in the file to be processed as a ***command line argument***.

- It must include a simple check to make sure that the person who ran the program actually supplied the command line argument. This will require you to use a simple ***if statement*** in bash.

    - If the person does not give a command line argument, the program needs to print an error message to the terminal, and then quit with an ***exit status code*** of 1.

- It must print a message to the terminal stating that it is processing the file, listing the file by name. For instance, "Processing file database_query_results.txt..."

- It must store the date that the program is run in a variable, in the format YYYYMMDD_HHMMSS (year month day hour minute second). For this you will use the ***date*** command. The reason it is wise to store the date in your program is because, if your program takes a long time to run, the date and time when you start it might not be the date and time when it finishes.

- It must output its results to a new file, where the first row is a header that defines the columns. The columns should be named DB and Event_Date.

- It must do all the text processing using `sed`. All the `sed` calls should be done in a single line, with the output being piped between calls. The end of the line should be where the text is sorted before being written to a file.

- It must output the data sorted in the correct order, from lowest to highest, by database number.

- It must name the output file using the timestamp you created before (in the format given above). For instance, `20160920_131211_output.csv`, or something like that. It's fine to write the output to one file and then to rename it at the end of the program.

The Blackboard assignment also has an example of what your file should look like if your program is working correctly.

Here are a few hints which will get you headed in the right direction.

- There are many different ways to check whether or not your script received the right amount of command line arguments. Using Google, you should be able to find plenty.

- Since you need your file to have a header, it might make sense to create the file using one operation, and then to append the results to it.

- When using `sed`, you will want to mostly make use of the ***substitute*** and ***delete*** options. It is likely that you'll need to use some ***regular expressions*** when dealing with the spaces in the file (since you need to replace them with commas).

    - Note that doing this only using `sed` tends to result in multiple calls, so don't be worried if you feel like you are using it a lot. We'll work on making it more concise in the next part.

- Make sure that when you sort the file, the `sort` command is doing what you expect.

Include your finalized script with your submission from Part 1. [45 points]

Part 3

You might have noticed that you used a lot of calls to `sed` in the previous part. Let's make your script more efficient by combining `grep` and `sed` together. Take the program you created for Part 2, and replace your calls to `sed` with a combination of `grep` and `sed`. You can do this by using `grep` to select only the lines that you are interested in, and then piping them through `sed` and `sort` to get the final output. Your final version should use each command only once. Hint: you will want to make use of a ***line anchor*** in your regular expression with `grep`. As before, save and include your finalized script with your submission from Part 1. [15 points]