MCB 536: Tools for Computational Biology
Lecture 05: Intro to Command Line pt II

October 12, 2023

Melody Campbell, Fred Hutch

Teaching Goals

- Interacting with the command line
    - Review
        - Syntax
    - Scripting
    - For-loops
- Tutorial

Syntax (Structure)

command -flag(s) argument

ls -ltr tfcb_2022

what do you want me to do?	what options do you want?	what should i perform it on?

verb adverb noun

english: list out time sorted backwards and fully what is in this folder

Pipes

- Pipes are a form redirection
- They let you use the output of one command and pass it on to a new command

- Two new commands:
  - head file.txt prints first 10 lines of a file
  - tail file.txt prints last 10 lines of a file
    - head -5 file.txt prints first 5 lines of a file

- What if we only want to print line 5?
  - head -5 file.txt | tail -1

Semicolon

- Semicolons allow you to execute two separate commands on the same line. In functions in a similar way to pressing the 'return' key
- Try:
  - pwd
  - ls
- or
  - pwd ; ls
  - spaces don't matter they are ignored
- not the same as pipe, try
  - head -5 file.txt ; tail -1
  - ^ this will hang so use ctrl + C to kill it

Variables

- Variables are shown by having a dollar sign
- Some are set by most systems ($USER $HOME)
- Others you can set on your own to personalize your computer ~OR~ for writing simple scripts
  - They can update and change!
    - they can be commands or flags or arguments
  - Example: `today_is=october ; echo $today_is`

For Loops

- A ‘for loop’ lets you iterate a process
- It allows you to set a variable and to change it over a repeating process
- The variable $i is often used, but you can use anything

- just using numbers, try:
- for i in {1..25}
  - this opens open the command sequence and you’ll see a > at the beginning of your line
- do echo $i
- done
  - (this ends the command sequence)

For Loops

- Alternatively you can do it all on one line with the semi colon
  - for i in {1..32} ; do echo $i ; done
  - for i in {1..32} ; do echo I have $i files in this directory ; done
- any variable works (except a few words that already have assigned meanings, and as always don't use special characters)
  - for pineapple in {1..32} ; do echo I have $pineapple files in this directory ; done

For Loops with Numbers

- Let’s use this to create a directory with some fake files
  - mkdir texts
  - cd texts

- Loop:
- for i in {1..32} ; do echo text_$i > text_file_$i.txt ; done
  - read one of the files. What does it say?

For Loops using ls

- Let’s say all of these texts are of meaningful, and we want to add that prefix to all of them
- for pineapple in `ls *.txt` ; do mv $pineapple important_$pineapple ; done
  - pineapple = the new variable. instead of being numbers counting up, it is now the output of ‘ls *.txt’ (so it is the list of files in your dir ending in .txt)
  - you are now using the mv command to change the name from text_file_1.txt to important_text_file_1.txt
    - note the extension is already in the variable
- now delete all these files, and use the recall command to re-make them!

For Loops using ls

- another way to do the exact same thing renaming would be:
- for i in {1..32} ; do mv text_file_$i.txt important_text_file_$i.txt ; done
    - note that when you use numbers ONLY the number is the variable so you need to put in the name & file extension

    - there are many ways to do the same thing when you script

Another useful loop example

- for i in {1..15} ; do mv important_text_file_$i.txt firsthalf_important_text_file_$i.txt; done

- for i in {16..32} ; do mv important_text_file_$i.txt secondhalf_important_text_file_$i.txt; done

- these names are getting long. to make them shorter:
  - for i in {1..15} ; do mv firsthalf_important_text_file_$i.txt firsthalf_$i.txt; done
  - for i in {16..32} ; do mv secondhalf_important_text_file_$i.txt secondhalf_$i.txt; done

For loop using cat

- Let's say we have a file (number_list.lst) with specific numbers that we want to name files after
  - for i in `cat number_list.lst` ; do echo $i ; done
    - make sure you use those very specific apostrophes
  - for i in `cat number_list.lst` ; do echo text_$i.txt > random_$i.txt ; done

  - number_list.txt, example

223

4324

67

71

112

434

35

562

Put this together

- mkdir texts ; cd texts ; for i in {1..32} ; do echo text_$i > text_file_$i.txt ; done ; for pineapple in `ls *.txt` ; do mv $pineapple important_$pineapple ; done

- Wow that's ugly.
- Let's make it into a script instead

Put this together using an editor

- open a new file in vs editor, copy the single line script
- take out all the “;”

```
mkdir texts
cd texts

for i in {1..32}
do echo text_$i > text_file_$i.txt
done

for pineapple in `ls *.txt`
do mv $pineapple important_$pineapple
done
```

- run using
  - bash text_script.sh

Now make it stand alone

```
#!/bin/bash
mkdir texts
cd texts

for i in {1..32}
do echo text_$i > text_file_$i.txt
done

for pineapple in `ls *.txt`
do mv $pineapple important_$pineapple
done
```

- change the permissions so you can execute this file
  - `chmod a+x script.sh`
  - run with `./text_script.sh`

Put this together w/o and editor

- be clever about the outputs
- use escape backslash wisely (note: escape works differently in quotations)

```
echo mkdir texts >text_script2.sh
echo cd texts >>text_script2.sh

echo for i in \{1..32\} >>text_script2.sh
echo do echo text_\$i.txt \> text_\$i.txt >>text_script2.sh
echo done >>text_script2.sh

echo for pineapple in \`ls\ \*.txt\`  >>text_script2.sh
echo do mv \$pineapple important_\$pineapple >>text_script2.sh
echo done >>text_script2.sh
```

- wow, all putting in all of those escape characters was really painful… if only there was an easier way…

vi

- vi (or vim) is a text editor
- while right now it just seems like a complicated way to edit a document it can be useful when:
  - you have a huge file and you want to navigate quickly and specifically
  - you want to find/replace very specific patterns
  - you're on a cluster or another computer without fancy software like vs code
- Usage
  - vi script.sh
  - "i" for insert mode
  - ctrl + v for paste
  - :wq (write and quit)
- More in the tutorial!

Now let's start the tutorial

- Go here:
  - https://github.com/FredHutch/tfcb_2023
  - navigate to lectures/lecture05
  - go through the readme to gitclone and cd into **lecture04** (sorry for this mismatch)

hint: use echo

- When you’re testing loops and variable outputs, or any code ‘echo; can be your bestie
- This way you can ensure your desired outputs are correct and don’t accidentally move overwrite files when you’re in the testing phase

- example:
- NO (for testing)
  - `for text in *.txt ; do mv $text important_$text ; done`
- YES (for testing)
  - `for text in *.txt ; do echo $text important_$text ; done`

for funsies

- Last year someone wanted to rename iPhone photos that all had the name IMG_####.jpeg
  - For example, you have a folder of photos where you know the first 22 were taken in one location and the rest were taken somewhere else
  - You want the image name to reflect this and you want to name the first 22 santorini_####.jpeg and the remaining to be called crete_####.jpeg
- To do this (read: the way I would do this), you would need a scripting language called "awk"
- In lecture05 there is a folder called "greece_photos" where you can try it out

for funsies

- For the scope of this class, it didn’t make sense to go into all of the meanings of how this command works, but you can extract the numbers of and make a list like this:

  -
    ```
    ls *.jpeg | awk -F '_' '{print $2}' | awk -F '.' '{print $1}' > greece.lst
    ```
- You can check how many files there are like this

  -
    ```
    cat greece.lst | wc -l
    ```
- And the rest you have learned how to do in the class:

  -
    ```
    head -22 greece.lst > first22.lst
    ```

  -
    ```
    tail -10 greece.lst > last10.lst
    ```

  -
    ```
    for i in `cat first22.lst `; do mv IMG_$i.jpeg santorini_'$i'.jpeg ; done
    ```

  -
    ```
    for i in `cat last10.lst `; do mv IMG_$i.jpeg crete_'$i'.jpeg ; don
    ```

As I mentioned, if you do this with your real files, either make a copy first or use the echo command so you don’t overwrite files!!