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

October 10, 2024

Melody Campbell, Fred Hutch

Why are we doing this?

- Many programs and computing clusters don't have nice looking files that you can click and drag to get around
- You now have many of the tools you will need to write scripts and routine and thus automate your work
  - for example: it is much nicer to write a few lines of code that to manually change caats01.jpgs to cats01.jpgs 10,000 times
- Shell scripting helps you get files into the correct format to feed into more sophisticated programs

Extra Commands That Were Requested Last Year

- How to rename:
  - mv input.txt output.txt
- Copy files
  - cp file.txt file2.txt
- How to delete a file
  - rm file.txt
- How to delete a dir
  - rm -r directory
- Print out contents of dir into new text file
  - ls * > contents.txt

Terminate & Cursor shortcuts

- Say you accidentally typed a bunch of things in the terminal that you don't want anymore…
  - Control + C
  - this will not run the command, and give you a fresh new line
- The cursor can be annoying. You have to move via arrows
- Say you typed a bunch of things in the terminal and you made one mistake
  - at the beginning
    - Control + A
  - at the end
    - Control + E

fournova

COMMAND LINE CHEAT SHEET

presented by **TOWER** › Version control with Git - made easy

DIRECTORIES

```
$ pwd
```
Display path of current working directory

```
$ cd <directory>
```
Change directory to <directory>

```
$ cd ..
```
Navigate to parent directory

```
$ ls
```
List directory contents

```
$ ls -la
```
List detailed directory contents, including hidden files

```
$ mkdir <directory>
```
Create new directory named <directory>

OUTPUT

```
$ cat <file>
```
Output the contents of <file>

```
$ less <file>
```
Output the contents of <file> using the less command (which supports pagination etc.)

```
$ head <file>
```
Output the first 10 lines of <file>

```
$ <cmd> > <file>
```
Direct the output of <cmd> into <file>

```
$ <cmd> >> <file>
```
Append the output of <cmd> to <file>

```
$ <cmd1> | <cmd2>
```
Direct the output of <cmd1> to <cmd2>

```
$ clear
```
Clear the command line window

FILES

```
$ rm <file>
```
Delete <file>

```
$ rm -r <directory>
```
Delete <directory>

```
$ rm -f <file>
```
Force-delete <file> (add `-r` to force-delete a directory)

```
$ mv <file-old> <file-new>
```
Rename <file-old> to <file-new>

```
$ mv <file> <directory>
```
Move <file> to <directory> (possibly overwriting an existing file)

```
$ cp <file> <directory>
```
Copy <file> to <directory> (possibly overwriting an existing file)

```
$ cp -r <directory1>
  <directory2>
```
Copy <directory1> and its contents to <directory2> (possibly overwriting files in an existing directory)

```
$ touch <file>
```
Update file access & modification time (and create <file> if it doesn't exist)

PERMISSIONS

```
$ chmod 755 <file>
```
Change permissions of <file> to 755

```
$ chmod -R 600 <directory>
```
Change permissions of <directory> (and its contents) to 600

```
$ chown <user>:<group> <file>
```
Change ownership of <file> to <user> and <group> (add `-R` to include a directory's contents)

SEARCH

```
$ find <dir> -name "<file>"
```
Find all files named <file> inside <dir> (use wildcards [*] to search for parts of filenames, e.g. "file.*")

```
$ grep "<text>" <file>
```
Output all occurrences of <text> inside <file> (add `-i` for case-insensitivity)

```
$ grep -rl "<text>" <dir>
```
Search for all files containing <text> inside <dir>

NETWORK

```
$ ping <host>
```
Ping <host> and display status

```
$ whois <domain>
```
Output whois information for <domain>

```
$ curl -O <url/to/file>
```
Download <file> (via HTTP[S] or FTP)

```
$ ssh <username>@<host>
```
Establish an SSH connection to <host> with user <username>

```
$ scp <file>
  <user>@<host>:/remote/path
```
Copy <file> to a remote <host>

PROCESSES

```
$ ps ax
```
Output currently running processes

```
$ top
```
Display live information about currently running processes

```
$ kill <pid>
```
Quit process with ID <pid>

fournova

COMMAND LINE TIPS & TRICKS

presented by **TOWER** › Version control with Git - made easy

GETTING HELP

On the command line, help is always at hand: you can either type `man <command>` or `<command> --help` to receive detailed documentation about the command in question.

FILE PERMISSIONS

On Unix systems, file permissions are set using three digits: the first one representing the permissions for the owning user, the second one for its group, and the third one for anyone else.

Add up the desired access rights for each digit as following:

`4` – access/read (r)
`2` – modify/write (w)
`1` – execute (x)

For example, `755` means "rwx" for owner and "rx" for both group and anyone. `740` represents "rwx" for owner, "r" for group and no rights for other users.

COMBINING COMMANDS

If you plan to run a series of commands after another, it might be useful to combine them instead of waiting for each command to finish before typing the next one. To do so, simply separate the commands with a semicolon (`;`) on the same line.

Additionally, it is possble to execute a command only if its predecessor produces a certain result. Code placed after the `&&` operator will only be run if the previous command completes successfully, while the opposite `||` operator only continues if the previous command fails. The following command will create the folder "videos" only if the `cd` command fails (and the folder therefore doesn't exist):

```
$ cd ~/videos || mkdir ~/videos
```

THE "CTRL" KEY

Various keyboard shortcuts can assist you when entering text: Hitting `CTRL+A` moves the caret to the beginning and `CTRL+E` to the end of the line.

In a similar fashion, `CTRL+K` deletes all characters after and `CTRL+U` all characters in front of the caret.

Pressing `CTRL+L` clears the screen (similarly to the `clear` command). If you should ever want to abort a running command, `CTRL+C` will cancel it.

THE "TAB" KEY

Whenever entering paths and file names, the `TAB` key comes in very handy. It autocompletes what you've written, reducing typos quite efficiently. E.g. when you want to switch to a different directory, you can either type every component of the path by hand:

```
$ cd ~/projects/acmedesign/docs/
```

...or use the `TAB` key (try this yourself):

```
$ cd ~/pr[TAB]ojects/
  ac[TAB]medesign/d[TAB]ocs/
```

In case your typed characters are ambiguous (because "ac" could point to the "acmedesign" or the "actionscript" folder), the command line won't be able to autocomplete. In that case, you can hit `TAB` twice to view all possible matches and then type a few more characters.

THE ARROW KEYS

The command line keeps a history of the most recent commands you executed. By pressing the `ARROW UP` key, you can step through the last called commands (starting with the most recent). `ARROW DOWN` will move forward in history towards the most recent call.

Bonus tip: Calling the `history` command prints a list of all recent commands.

HOME FOLDER

File and directory paths can get long and awkward. If you're addressing a path inside of your home folder though, you can make things easier by using the `~` character. So instead of writing `cd /Users/your-username/projects/`, a simple `cd ~/projects/` will do.

And in case you should forget your user name, `whoami` will remind you.

OUTPUT WITH "LESS"

The `less` command can display *and paginate* output. This means that it only displays one page full of content and then waits for your explicit instructions. You'll know you have `less` in front of you if the last line of your screen either shows the file's name or just a colon (`:`).

Apart from the arrow keys, hitting `SPACE` will scroll one page forward, `b` will scroll one page backward, and `q` will quit the `less` program.

DIRECTING OUTPUT

The output of a command does not necessarily have to be printed to the command line. Instead, you can decide to direct it to somewhere else.

Using the `>` operator, for example, output can be directed to a file. The following command will save the running processes to a text file in your home folder:

```
$ ps ax > ~/processes.txt
```

It is also possible to pass output to another command using the `|` (pipe) operator, which makes it very easy to create complex operations. E.g., this chain of commands will list the current directory's contents, search the list for PDF files and display the results with the `less` command:

```
$ ls | grep ".pdf" | less
```

30-day free trial available at
www.git-tower.com

TOWER
Version control with Git - made easy

Any New Command Requests?

- How to create many sequentially numbered folders/ directories

Teaching Goals

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

Syntax (Structure)

command -flag(s) argument

ls -ltr tfcb_2024

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 numbers

- 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.lst, 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
  - esc (to exit insert mode)
  - :wq (write and quit)
- More in the tutorial!

Now let’s start the tutorial

- Go here:
  - https://github.com/FredHutch/tfcb_2024
  - navigate to lectures/lecture05
  - go through the readme to gitclone and cd into **lecture04**

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 l awk -F '_' '{print $2}' l awk -F '.' '{print $1}' > greece.lst
- You can check how many files there are like this
  - cat greece.lst l 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!!