Introduction to UNIX

Genome 559: Introduction to Statistical and Computational Genomics

Seungsoo Kim

What is UNIX?

- Unix is a family of operating systems (like Linux)
- The language of the command-line (except on PCs, until Windows 10; see http://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/)
- Set of programs/commands for:
  - Navigating file directories
  - Manipulating text files
  - (and much more we won't cover)

Why should you care about UNIX?

- Have you ever spent a long time systematically renaming files one by one, only to miss some?
- Do you ever have to work with files too large for your text editor?
- Will you have to work on a computer cluster?

You’ve already been using UNIX!

```
$ cd Documents
$ python hello.py
Hello, World!
```

Where are we?

Print working directory

```
$ pwd
/Users/seungsoo
```

(this is called a path)

List files and subdirectories

```
$ ls
Applications
Desktop
Documents
…
```

Let’s go somewhere …

Change directory

```
$ cd Documents
$ pwd
/Users/seungsoo/Documents
```

We’re now in a subdirectory.

Go up one level

```
$ cd ..
$ pwd
/Users/seungsoo
```

Let’s make a new folder (directory)

Make directory

```
$ mkdir GS559
```

Move into that directory

```
$ cd GS559
```

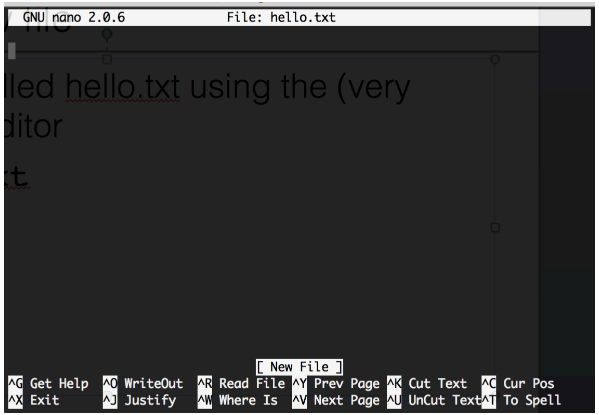
Go back home

```
$ cd
```

Let’s write a new file

Write a new file called hello.txt using the (very basic) nano text editor

```
$ nano hello.txt
```



```
GNU nano 2.0.6                File: hello.txt

[ New File ]
```

nano text editor

- Instructions are at the bottom of the screen
- ^X means Control-X, etc.
- To save: ^X, then y, then Enter

Renaming/moving files

Rename hello.txt to greetings.txt

```
$ mv hello.txt greetings.txt
```

Move greetings.txt to the GS559 folder

```
$ mv greetings.txt GS559
```

Move greetings.txt back to the current folder

```
$ mv GS559/greetings.txt .
```

The single dot “.” is a shortcut for the current directory

Copying and deleting files

Make a copy of greetings.txt called greetings2.txt

```
$ cp greetings.txt greetings2.txt
```

Remove (delete) greetings2.txt

```
$ rm greetings2.txt
```

Be super careful with `rm` – unlike files put in the “Recycling Bin”, files deleted with `rm` are permanently gone.

Remember, you can use `ls` to check what files are in your current location.

Summary of part 1 (file navigation)

`pwd`	print working directory
`cd dir`	change directory to dir
`cd ..`	go up one level
`ls`	list directory contents
`nano file1`	edit file1 using text editor nano
`mv file1 file2`	move/rename file1 to file2
`cp file1 file2`	copy file1 and save as file2
`rm file1`	delete file1

The wildcards: * and ?

- UNIX is particularly powerful because of its wildcards
  - * indicates any string of characters (including none)
  - ? indicates any single character
- `ls` shows all files in the directory (except for some hidden files … check out ls -a)
- `ls *.py` shows all files in the directory that end in .py
- `ls D*` shows all files/directories that start with D (case-sensitive)
- Warning: be particularly careful using wildcards with `rm`! A good practice is to check which files you would delete with a command by first using `ls` in place of `rm`

More wildcard examples

If you had the following files in your directory:

- `PS1.txt`
- `PS1.py`
- `PS2.txt`
- `PS2.py`
- `PS3.txt`
- `PS3.py`
- `Lecture1.pptx`
- `Lecture1A.pptx`
- `Lecture1B.pptx`
- `Lecture2A.pptx`
- `Lecture2B.pptx`

How would you move all files ending in .txt to a new folder?

Which files would **`rm Lecture1?.pptx`** delete?

More wildcard examples - solutions

If you had the following files in your directory:

- `PS1.txt`
- `PS1.py`
- `PS2.txt`
- `PS2.py`
- `PS3.txt`
- `PS3.py`
- `Lecture1.pptx`
- `Lecture1A.pptx`
- `Lecture1B.pptx`
- `Lecture2A.pptx`
- `Lecture2B.pptx`

How would you move all files ending in .txt to a new folder?
`mv *.txt newfolder` (folder must already exist)

Which files would **`rm Lecture1?.pptx`** delete?
Lecture1A.pptx and Lecture 1B.pptx (not Lecture1.pptx)

A couple of handy shortcuts

- Tab-completion
  - if there's only one file/directory that starts with the set of characters you've typed, hitting Tab will complete it

```
$ ls gree<Tab>
$ ls greetings.txt
```

  - if there are multiple such files, hitting Tab twice will list them all
- Command history: use the up/down arrow keys to get your previously entered commands

Viewing/manipulating files

- UNIX is a text-based system – most files are flat (not fancy like Word) text files
- UNIX contains a lot of useful programs for working with text files
- UNIX programs read in files and write out to the standard out (and error) stream, unless redirected to a file
  - In general, they do not edit files in place

Print the beginning of the file

Print the top (head) of the file PS3_chr21.txt (by default, first 10 lines)

```
$ head PS3_chr21.txt
ctccaaagaaattgtagttttcttctggcttagaggtagatcatcttggt
ccaatcagactgaaatgccttgaggctagatttcagtctttgtggcagct
ggtgaatttctagtttgccttttcagctagggattagctttttaggggtc
ccaatgcctagggagatttctaggtcctctgttccttgctgacctccaat
tttgtctatccttttgctgagaggtctgcttaacttccttttagtcaggt
agctccattttatgctaagcttcttagttgctcaccttctgcagctaaag
aatcagaaaatgctgtgaaggaaaaacaaaatgaaattgcattgtttcta
ccggccctttatcaagccctggccaccatgatagtcatgaattccaattg
ttgtctatgcaggcctaccagatttctaacatctctgagctaccattttc
ttcttagctatctgctcagcaaatgtatccaaatgaaaggctgtggagaa
```

Print the first line in the file

```
$ head –n 1 PS3_chr21.txt
ctccaaagaaattgtagttttcttctggcttagaggtagatcatcttggt
```

Print the beginning of the file

Print the top (head) of the file PS3_chr21.txt (by default, first 10 lines)

```
$ head PS3_chr21.txt
ctccaaagaaattgtagttttcttctggcttagaggtagatcatcttggt
ccaatcagactgaaatgccttgaggctagatttcagtctttgtggcagct
ggtgaatttctagtttgccttttcagctagggattagctttttaggggtc
ccaatgcctagggagatttctaggtcctctgttccttgctgacctccaat
tttgtctatccttttgctgagaggtctgcttaacttccttttagtcaggt
agctccattttatgctaagcttcttagttgctcaccttctgcagctaaag
aatcagaaaatgctgtgaaggaaaaacaaaatgaaattgcattgtttcta
ccggccctttatcaagccctggccaccatgatagtcatgaattccaattg
ttgtctatgcaggcctaccagatttctaacatctctgagctaccattttc
ttcttagctatctgctcagcaaatgtatccaaatgaaaggctgtggagaa
```

Print the first line in the file

```
$ head –n 1 PS3_chr21.txt
ctccaaagaaattgtagttttcttctggcttagaggtagatcatcttggt
```

This is an option, specifying how many lines to print

Print the end of the file

Print the end (tail) of the file PS3_chr21.txt (by default, last 10 lines)

```
$ tail PS3_chr21.txt
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

Print the last line in the file

```
$ tail –n 1 PS3_chr21.txt
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

Print an entire file (or multiple, concatenated) to the screen

```
$ cat greetings.txt
hello


$ cat greetings.txt greetings.txt
hello
hello
```

Redirecting to standard in, standard out, and standard error

```
prog1 < file1
```

means feed file1 into the standard input of the program prog1

```
prog1 arg1 > file1
```

means run prog1 with argument arg1 and save the output to file1

```
prog1 arg1 | prog2
```

means run prog1 with argument arg1 and feed the output as the standard input to program prog2

Using the left arrow to replace an argument expecting a file with the output of a program

```
prog2 <(prog1 arg1)
```

is (mostly) equivalent to

```
prog1 arg1 > file1
prog2 file1
```

You can string these together!

```
prog3 <(prog1 arg1) <(prog2 arg2)
```

(mostly) equivalent to

```
prog1 arg1 > file1
prog2 arg2 > file2
prog2 file1 file2
```

Exercises

Create a new file twogreetings.txt that contains the contents of greetings.txt twice in a row.

Concatenate the first 10 lines of PS3_chr21.txt with the last 10 lines of PS3_chr21.txt and print to the screen.

Exercises - solutions

Create a new file twogreetings.txt that contains the contents of greetings.txt twice in a row.

```
$ cat greetings.txt greetings.txt > twogreetings.txt
```

Concatenate the first 10 lines of PS3_chr21.txt with the last 10 lines of PS3_chr21.txt and print to the screen.

```
$ cat <(head PS3_chr21.txt) <(tail PS3_chr21.txt)
```

(How many lines of Python would this take?)

How big is the file?

wc counts the number of lines, words, and characters (bytes) in a file

```
$ wc PS3_chr21.txt
 774374  774374 40267443 PS3_chr21.txt
```

Just print the number of lines

```
$ wc -l PS3_chr21.txt
774374
```

less: a better viewer for looking at big files

less works with files one screen at a time
Try `less PS3_chr21.txt`

You can search for strings in the file:
type: `/GATT` to search “GATT” and highlight all matches
then hit “n” to go to the next hit

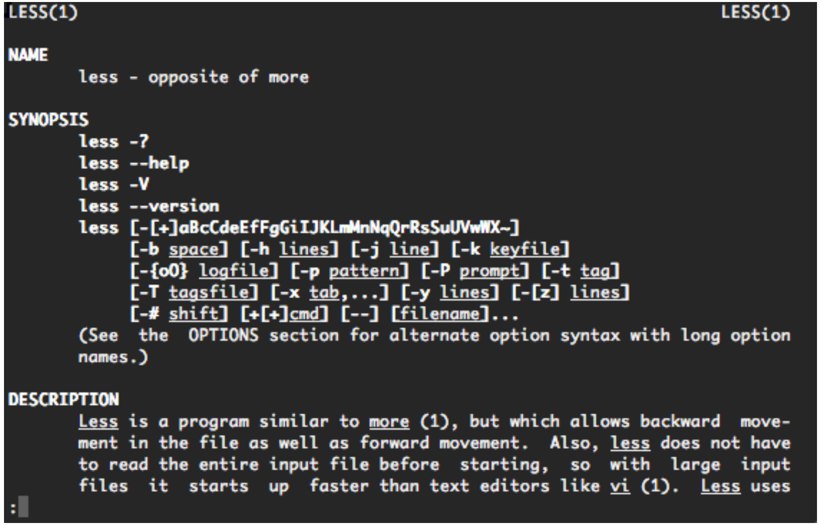
Hit arrow keys to navigate
Hit Space to go a page down
Hit “q” to exit

How do I remember all those options?

Every command has a manual page. Access it with the command **`man`**

```
$ man less
```

Read through manuals using the `less` commands!



```
LESS(1)                                                            LESS(1)

NAME
       less - opposite of more

SYNOPSIS
       less -?
       less --help
       less -V
       less --version
       less [-[+]aBcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX~]
            [-b space] [-h lines] [-j line] [-k keyfile]
            [-{oO} logfile] [-p pattern] [-P prompt] [-t tag]
            [-T tagsfile] [-x tab,...] [-y lines] [-[z] lines]
            [-# shift] [+[+]cmd] [--] [filename]...
       (See  the  OPTIONS section for alternate option syntax with long option
       names.)

DESCRIPTION
       Less is a program similar to more (1), but which allows backward  move-
       ment in the file as well as forward movement.  Also, less does not have
       to read the entire input file before  starting,  so  with  large  input
       files  it  starts  up  faster than text editors like vi (1).  Less uses
:
```

Working with big files - grep

Print all lines in PS3_chr21.txt that contain a string of interest, here GATT

```
$ grep GATT PS3_chr21.txt
```

Print all lines in PS3_chr21.txt that do NOT contain “N”

```
$ grep —v N PS3_chr21.txt
```

Some options:

-f: instead of just a string, take a file with a list of query sequences

-w: require the match to be a word (have whitespace on either side)

Working with big files - cut

We often work with tables, with columns separated by tabs (or spaces, commas, etc.)

Print the 3rd, 4th, 5th, and 9th columns (fields) of file1.txt

```
$cut –f 3-5,9 file1.txt
```

some options:

-d: specify delimiter - comma, space, tab (default)

-c: get characters rather than fields

How can we keep a record of these kinds of complex commands, and rerun them later?

- Shell scripts are programs that can be run by the UNIX interpreter, as if you had typed each line directly on the command-line.
- They can run other programs (e.g. Python programs), so they’re useful for building complex programs (or analysis pipelines) that use programs other people have written (like BLAST)
- Like Python programs, they can take arguments, use loops and conditional statements, etc.
- They end in .sh and are executable pieces of text

Shell scripts

Suppose you had a Python program called hello-n.py (what does it do?)

```
import sys
for i in range(int(sys.argv[1])):
   print “Hello!”
```

And a shell script called five.sh

```
python hello.py 5
```

Then running five.sh would print “Hello!” five times by running hello-n.py with the argument 5.

How do we run a shell script?

We first have to make the script *executable*, with the command **`chmod`**

```
$ chmod +x five.sh
```

Then we can enter the name of the program, five.sh, preceded by “./” (strictly speaking, needs to be a path – e.g. could be myfolder/five.sh)

```
$ ./five.sh
Hello!
Hello!
Hello!
Hello!
Hello!
```