Introduction to Bioinformatics

Lecture Four

UNIX for processing biological data

Lecturer
Gregory R. Grant

CIS4536 Fall 2023

Gregory R. Grant

Genetics Department

ggrant@pennmedicine.upenn.edu

ITMAT Bioinformatics Laboratory
University of Pennsylvania

Teaching Assistants

File Manipulation

- We are going to focus on the manipulation and management of data files.
- What do amateurs do?
  - – Windows Explorer
    - Dragging and dropping and right clicking is convenient
    - But time-consuming operations cannot be automated.
  - – Microsoft Word
    - An ugly monster.
    - Half baked features hacked on top of other half-baked features.
    - Chokes and dies on large files.
  - – Microsoft Excel
    - Limited number of rows.
    - Without learning to program, you can only do what they’ve built in, you cannot follow your heart’s desire.

What do the Pros Do?

- File system control with the UNIX Command Line
- File editing with Emacs and “Scripting”
  - Perl, Python, Ruby
- File manipulation with code
  - Perl, Python, Ruby, Java, C/C++

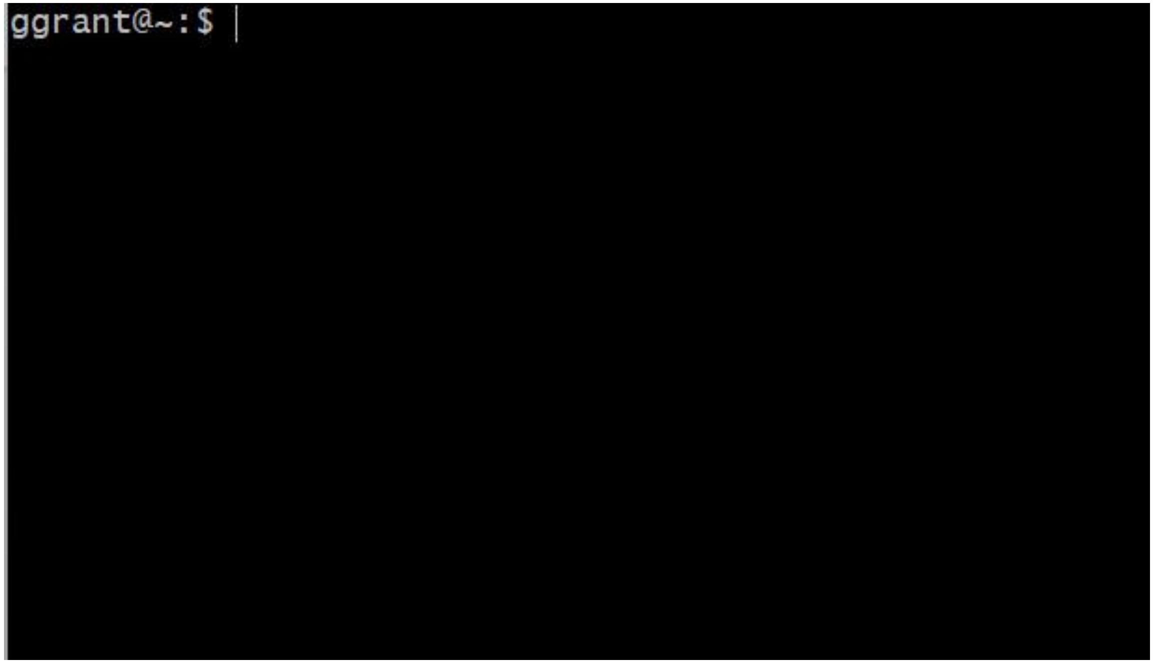
Our Goals

- To gain a working knowledge of Unix and its role in bioinformatics.
- We have given everybody accounts on a Unix machine where we will work and learn.
- We will inspect various “analysis pipelines” this semester that are implemented in a Unix environment.
  - This material will help you know what you’re looking at when we do that.
- And you will perform analysis tasks of your own in Unix.

The Old Days

HERE WE GO AGAIN.
IN MY DAY WE DIDN'T HAVE ANY FANCY GRAPHICS. YOU COULDN'T JUST PAUSE THE GAME. AND IF A GHOST GOT YOU, YOU WERE *DEAD!*

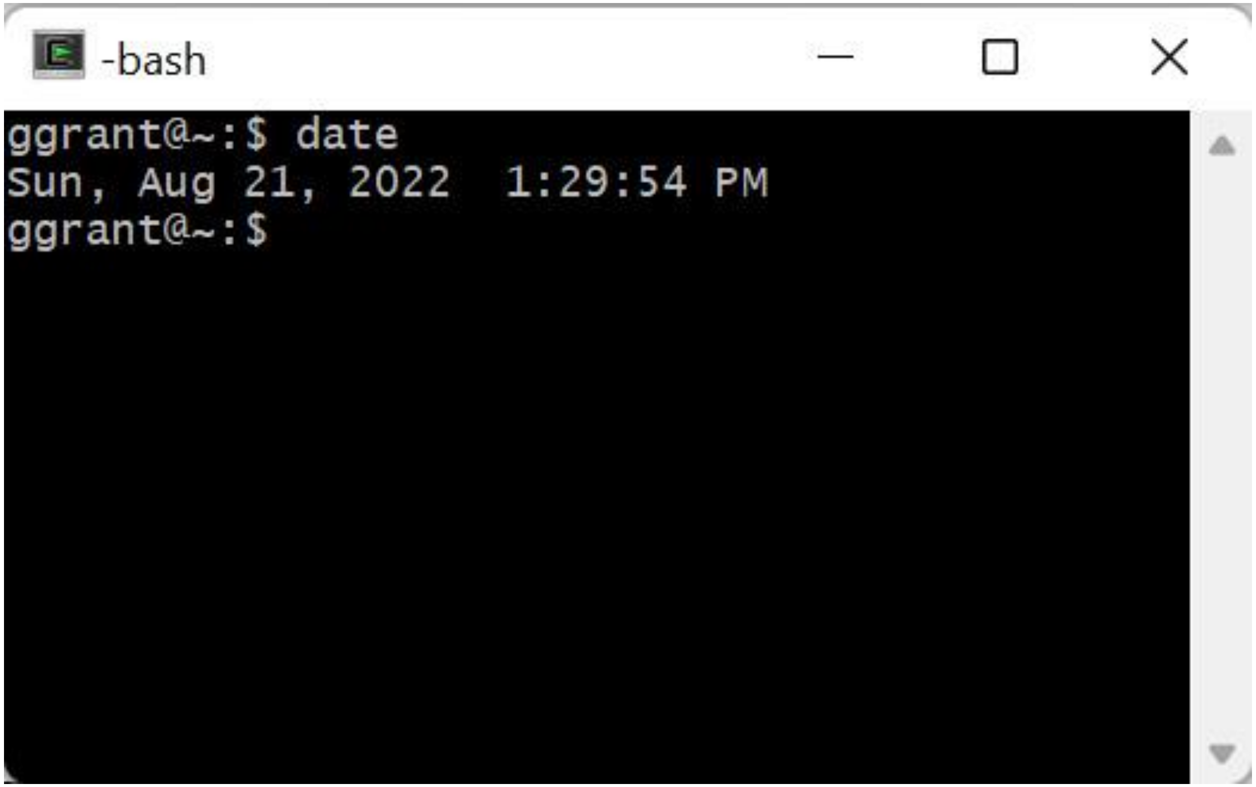
- Fifty years ago, when you turned on a computer, all you would see is something like this.
  - Just a text prompt.
  - There was no mouse, no windows, no images.
  - You interact with it entirely by typing text commands in text and hitting enter.



```
ggrant@~:$ |
```

Example Command

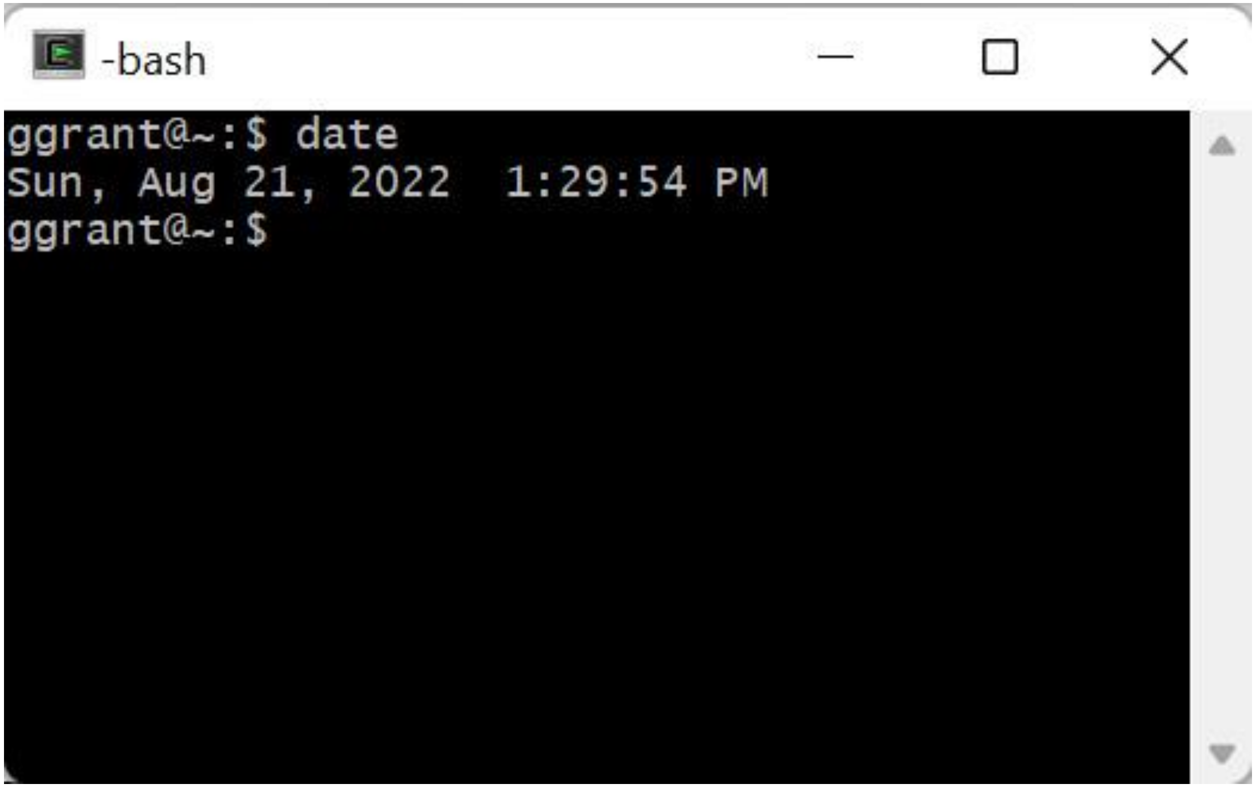
- Here the command ‘date’ was entered.
  - It returns the date and exact time.
- Followed by the prompt, ready for the next command.



```
ggrant@~:$ date
Sun, Aug 21, 2022  1:29:54 PM
ggrant@~:$
```

Text in / Text out

- These text commands return output (as text) to the screen.
- They can also manipulate files as we will see.



```
ggrant@~:$ date
Sun, Aug 21, 2022  1:29:54 PM
ggrant@~:$
```

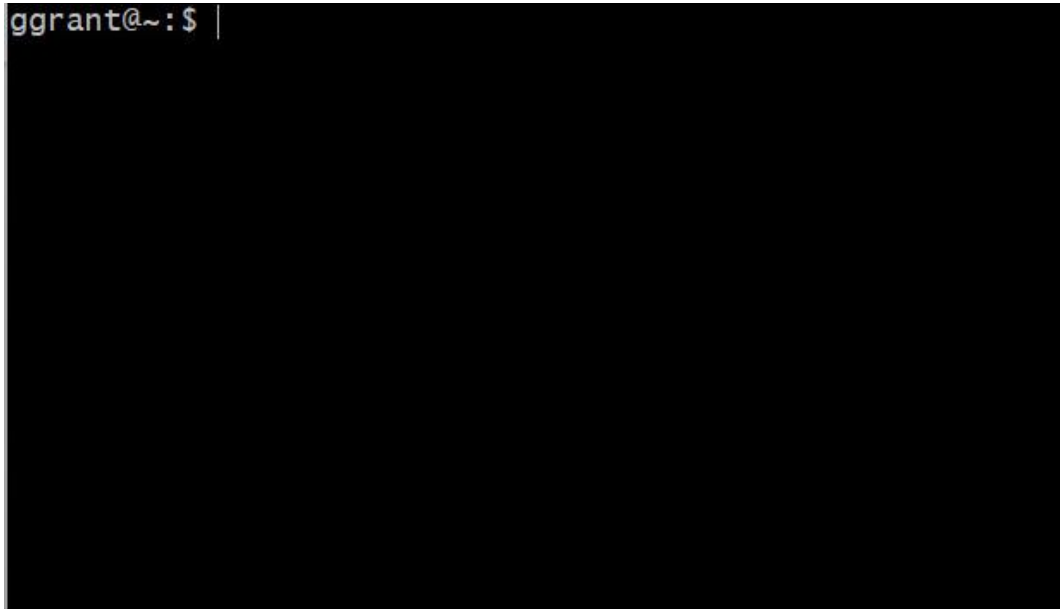
Case Sensitivity

- In most places Unix is case sensitive.
  - – You can have two files named ‘dog’ and ‘Dog’ and Unix will keep them straight.
- The notable exception is Mac which uses a case-insensitive version of Unix.

```
ggrant@~:$
```

Interfaces

- Interfaces have changed dramatically.
  - From text only to a world of graphics and pointing devices.
- But under the hood, computers are still the same.
  - They're organized as a hierarchical file system.
- The difference is simply how we access and manipulate those files.

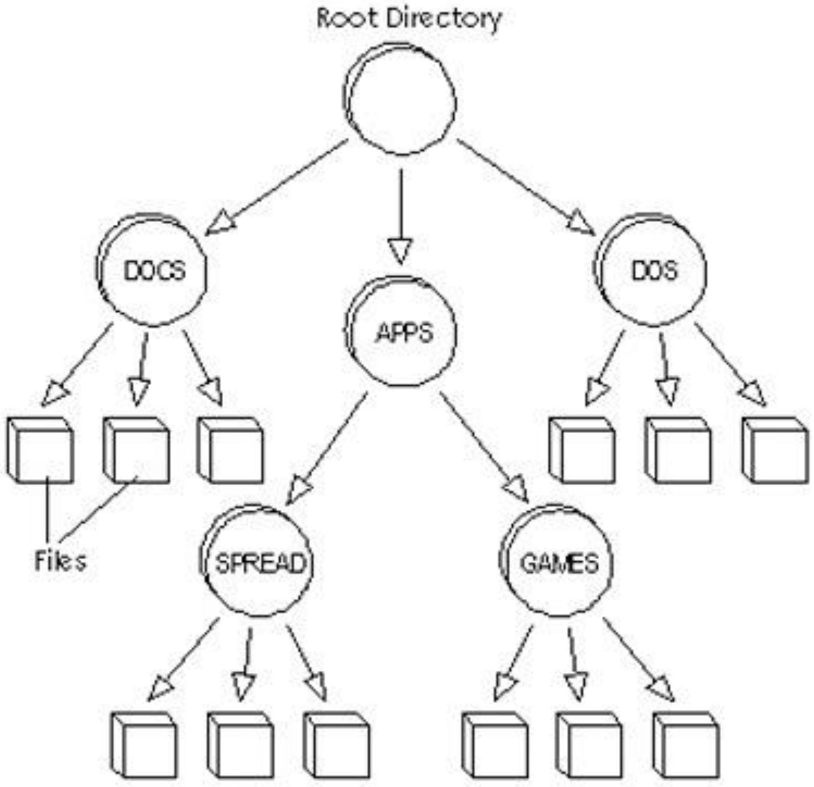


```
ggrant@~:$ |
```

The File System

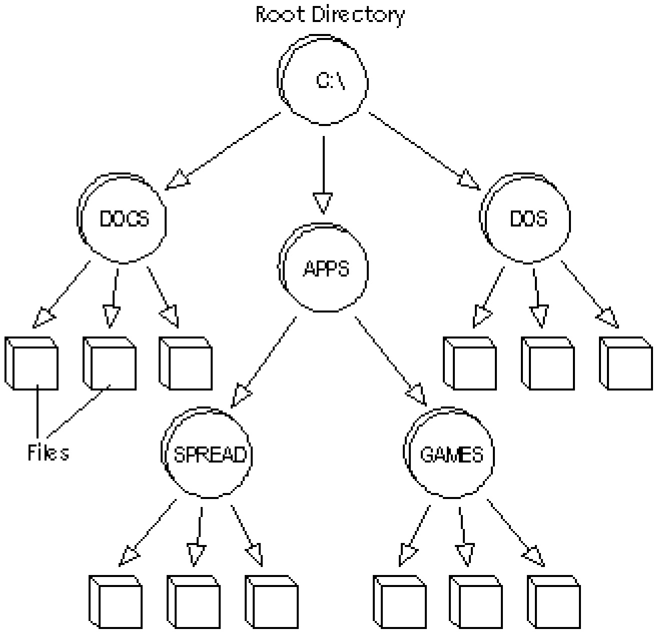
- Every computer has a "File System" which is how the files are organized on the computer.
- File systems contain files and directories (aka 'folders') organized like a tree.
- Directories contain files and subdirectories, which, in turn, can contain their own files and subdirectories, creating a hierarchy that can go to any depth.

- The top-level directory is called **the root directory**, it is the only directory which is not contained in any other directory.

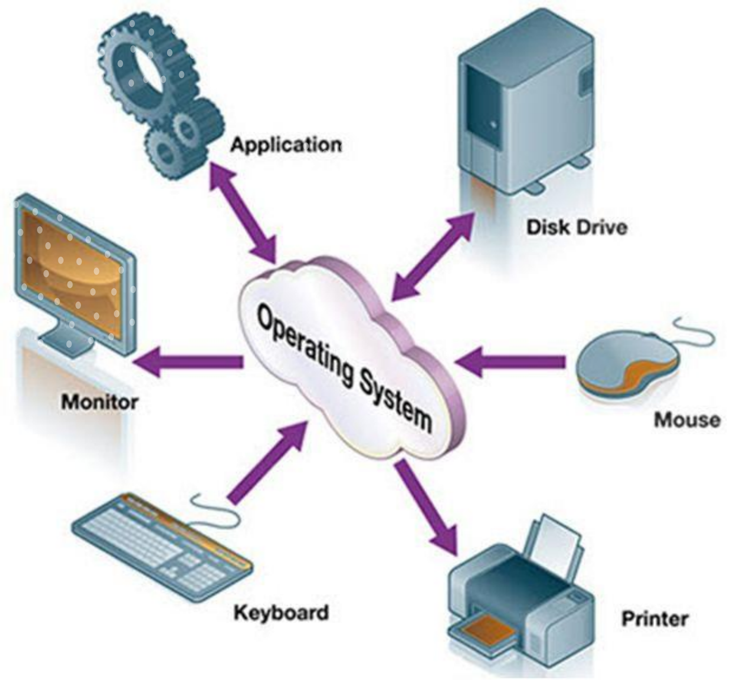


Navigating the File System

- On Windows you navigate this tree using Windows Explorer.
- On a Mac you navigate it using Finder.
- On UNIX, you navigate it using text commands, as we will see.



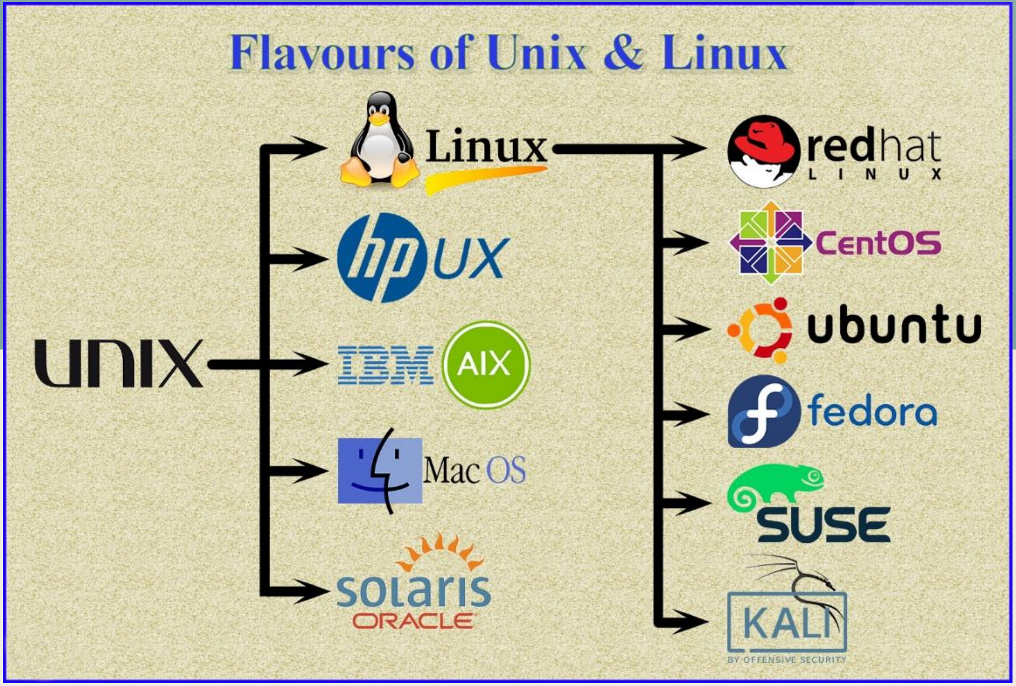
So, what exactly is an Operating System?



- The operating system is what you interact with when you use a computer.
  - The operating system allows you to interact with the file system, to run programs and manage files and users.
- The most popular laptop/desktop operating systems are Mac and Windows.
  - These operating systems are visual and intuitive.
  - In addition to the keyboard, you can use mice, touchpads, touchscreens, pointers and other input devices.
- Another OS you've surely heard of is called "Android"
  - Afterall, your phone is just another computer.

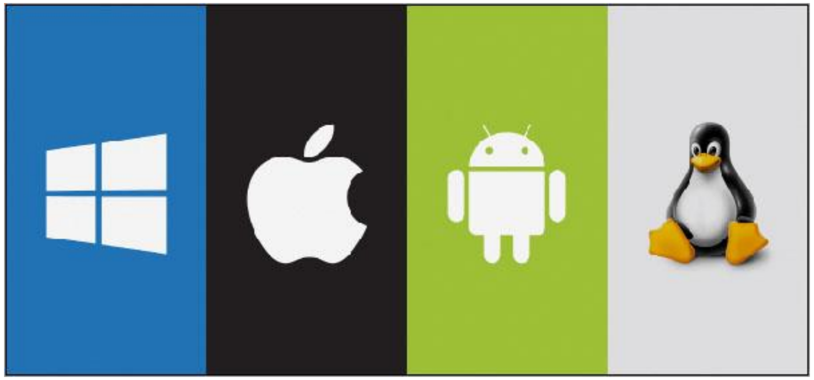
- UNIX is just another option.
  - It is not graphic - it is entirely text-based.
    - You interact solely with the keyboard.
  - Unix is extremely stable; it never crashes.
    - *That’s why the world runs on Unix.*

Many Flavors of Unix



- UNIX is to some extent proprietary.
- But many free alternatives have been developed.
  - All with essentially the same basic functionality as Unix.
  - You may have heard of Linux.
  - MAC OS is another one.
- Linux itself comes in many variants
  - Called Linux Distributions.
  - You'll hear names like Ubuntu, Debian, SUSE, Fedora, etc.
  - These are all just flavors of Linux which itself is a version of Unix.
- *Operating Systems are like cars; everybody seems to drive a different looking one. But we all get from point A to point B.*

Layers



- Often one OS is layered on top of another.
  - You interact with the one on top, the one underneath is hidden.
  - The lowest level operating system is called the “native” OS.
    - Informally “the OS under the hood”.
  - *But (importantly) there’s still only one file system, which both OS’s see simultaneously.*
- For example, the MAC OS is written on top of UNIX.
  - MAC has UNIX under the hood.
  - It didn’t used to be that way, but Apple went Unix around 2001.
- Windows used to be written on top of another text-based operating system called DOS.
  - But Windows itself has been rearchitected to be the native OS, no more layering on top of DOS.

Emulation

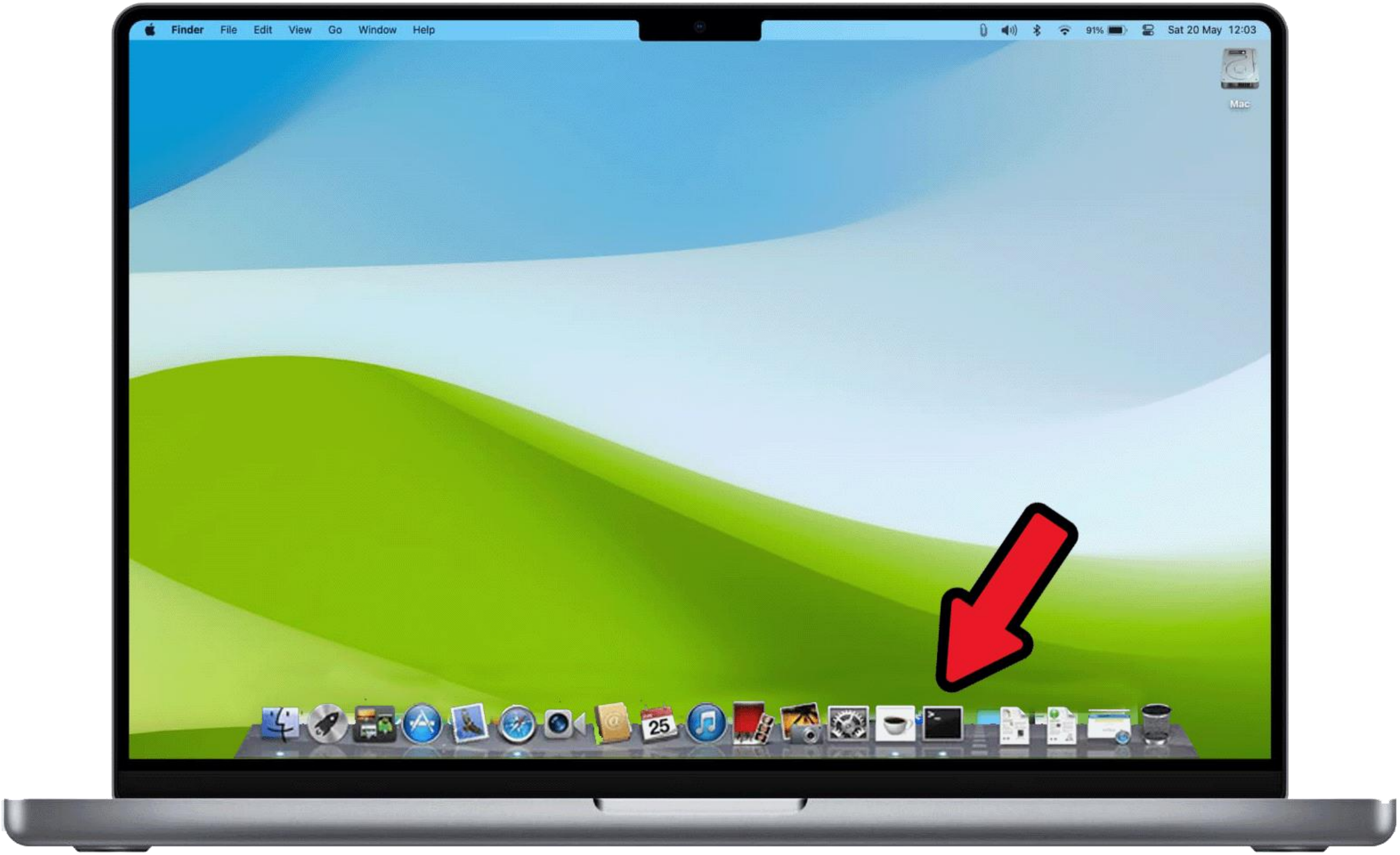
- There are also programs in one OS that can ‘emulate’ another OS.
- For example, here we see Android emulated on Windows.
  - You do this if you want to run an Android program on your laptop.

Android emulated on Windows

Let’s look at some more examples.

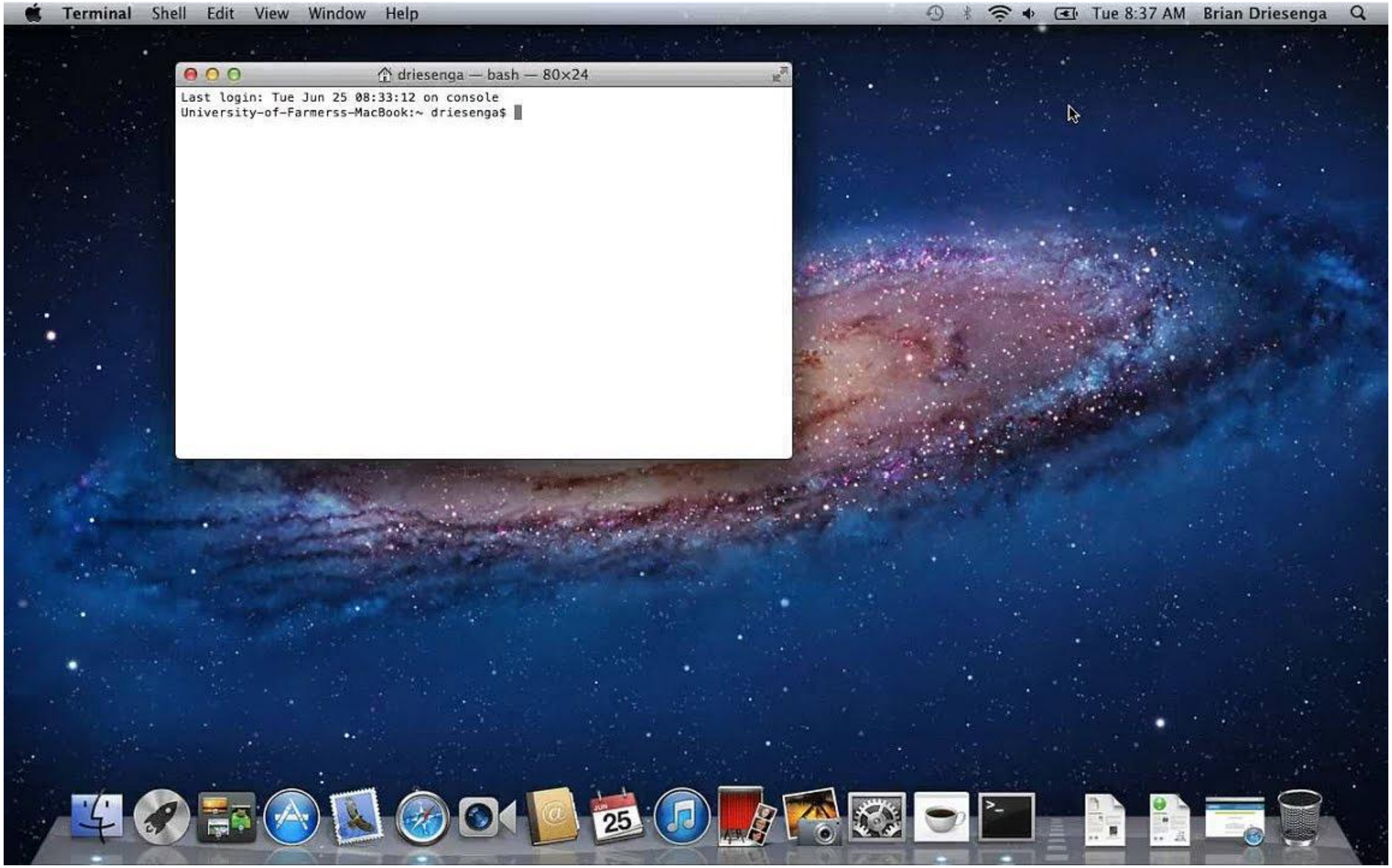
UNIX is Under the Hood of your MAC

- On a Mac, UNIX is the native OS, typically hidden from the user.
  - Mac OS is written on top of Unix.
- But you can peek under the hood by opening a terminal.



Mac Under the Hood

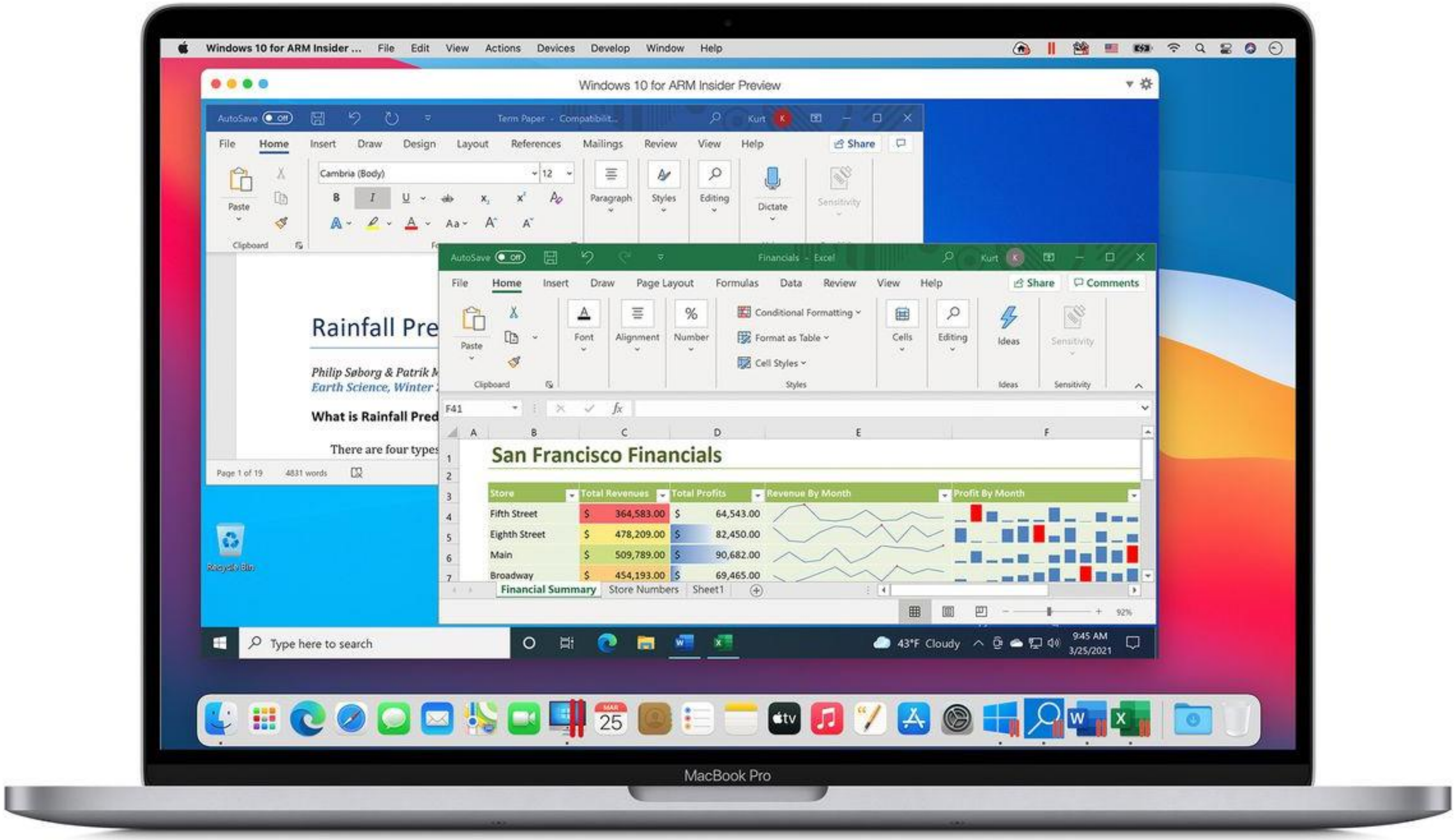
- You can do all the same things in this terminal as you can do using the Finder app.
  - Plus a whole lot more.
  - You just need to know the equivalent text commands, and soon you will.



```
Last login: Tue Jun 25 08:33:12 on console
University-of-Farmerss-MacBook:~ driesenga$ 
```

Windows Emulator on Mac

- Here we see Windows emulated on top of the MAC OS.
  - The whole Windows OS is running in that window.
- In that window you can use Window Explorer to manipulate files.
  - But they're still the same files as you see using Mac's Finder App.



Rainfall Pre

Philip Søborg & Patrik M
Earth Science, Winter

What is Rainfall Pred

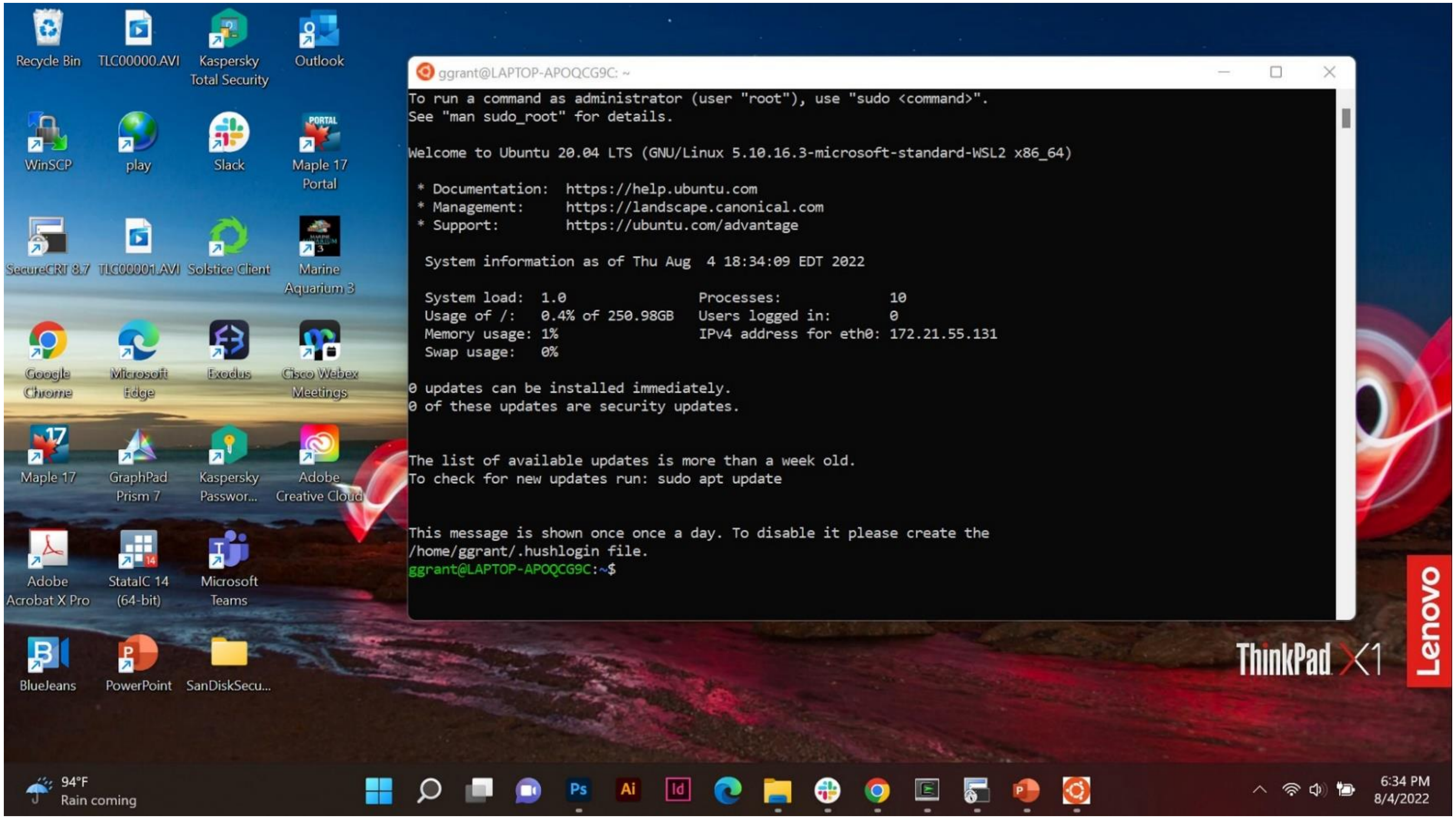
There are four types

San Francisco Financials

Store	Total Revenues	Total Profits	Revenue By Month	Profit By Month
Fifth Street	$ 364,583.00	$ 64,543.00		
Eighth Street	$ 478,209.00	$ 82,450.00		
Main	$ 509,789.00	$ 90,682.00		
Broadway	$ 454,193.00	$ 69,465.00		

UNIX Emulation in Windows

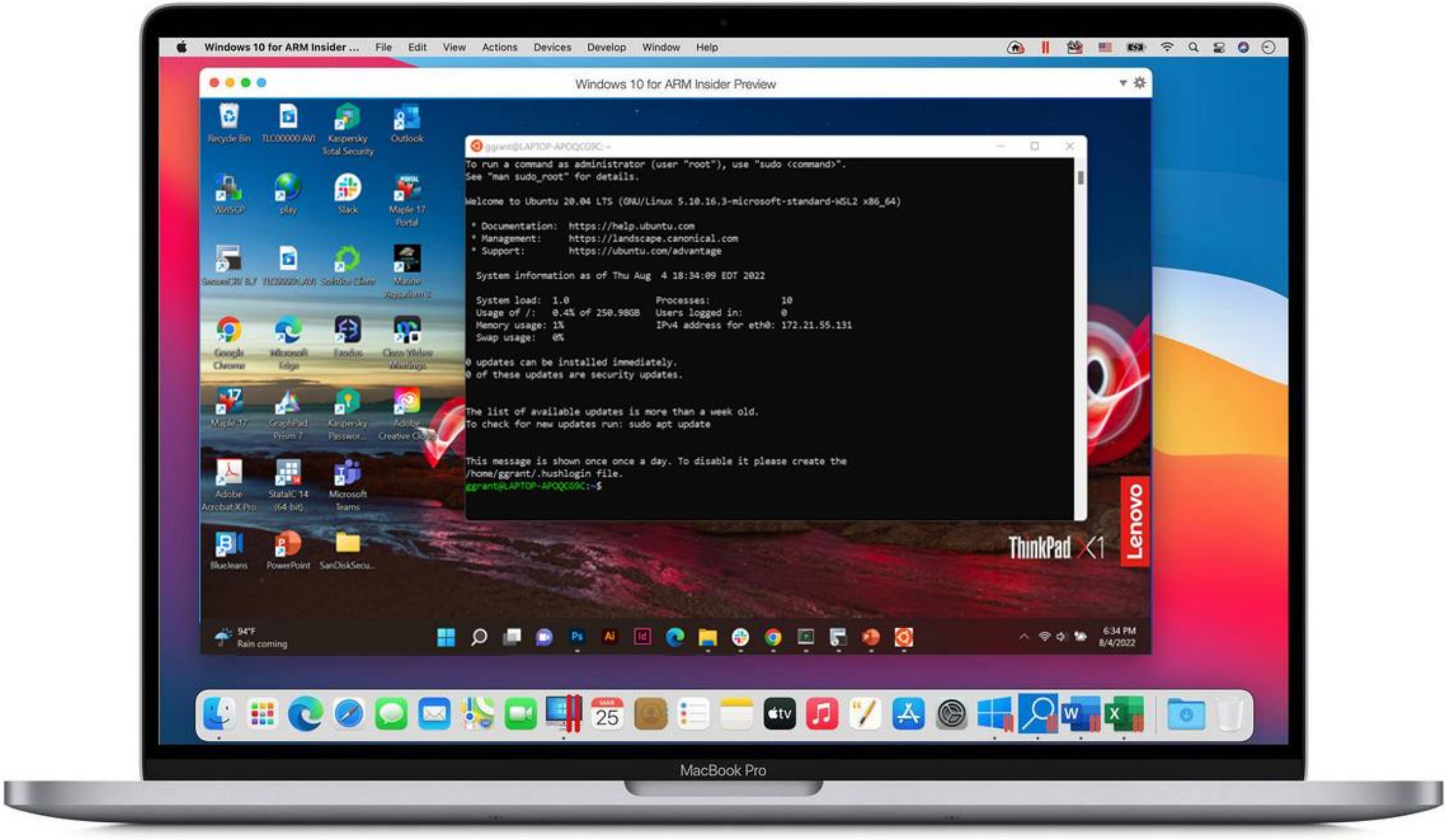
- There are multiple options to emulate UNIX in Windows.
  - WSL (built into Windows) and Cygwin (3rd party) are the two main ways to do it.
- You do not need to emulate Unix on Mac because it already has Unix under the hood.
- The Unix window you see here is *on top* of Windows.



```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Aug  4 18:34:09 EDT 2022

  System load:  1.0                Processes:             10
  Usage of /:   0.4% of 250.98GB   Users logged in:       0
  Memory usage: 1%                 IPv4 address for eth0: 172.21.55.131
  Swap usage:   0%

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


This message is shown once once a day. To disable it please create the
/home/ggrant/.hushlogin file.
ggrant@LAPTOP-APOQCG9C:~$
```

Four Layers

- Here we see Unix emulated in Windows, which itself is emulated in Mac, which itself is written on top of Unix.



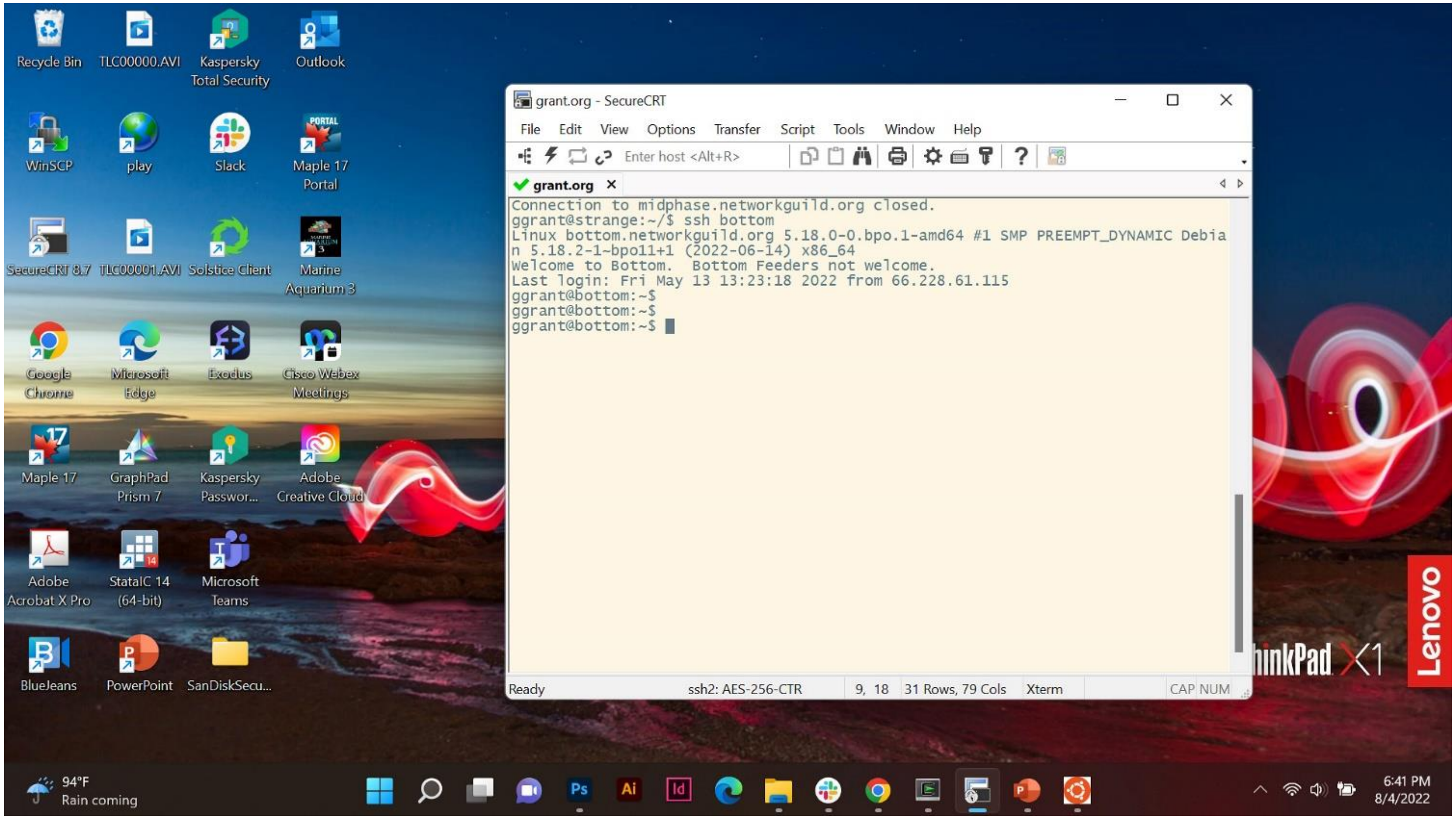
```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Aug  4 18:34:09 EDT 2022

  System load:  1.0                Processes:             10
  Usage of /:   0.4% of 250.98GB   Users logged in:       0
  Memory usage: 1%                 IPv4 address for eth0: 172.21.55.131
  Swap usage:   0%

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


This message is shown once once a day. To disable it please create the
/home/ggrant/.hushlogin file.
ggrant@LAPTOP-APOQCG9C:~$
```

Getting UNIX on Your Computer

- If you are on a Mac then you already have it.
  - Since Mac is built on top of Unix, just open the hood and there it is.
  - To open the hood, simply fire up the 'Terminal' application.
- If you have Windows, there are instructions to install Unix here.
  - *But you don’t need to do that for this class*
    - *You will be provided with a Unix account special for this class that you can access through a web browser.*
  - https://www.onmsft.com/how-to/install-ubuntu-on-windows-10-or-windows-11

Remote Access

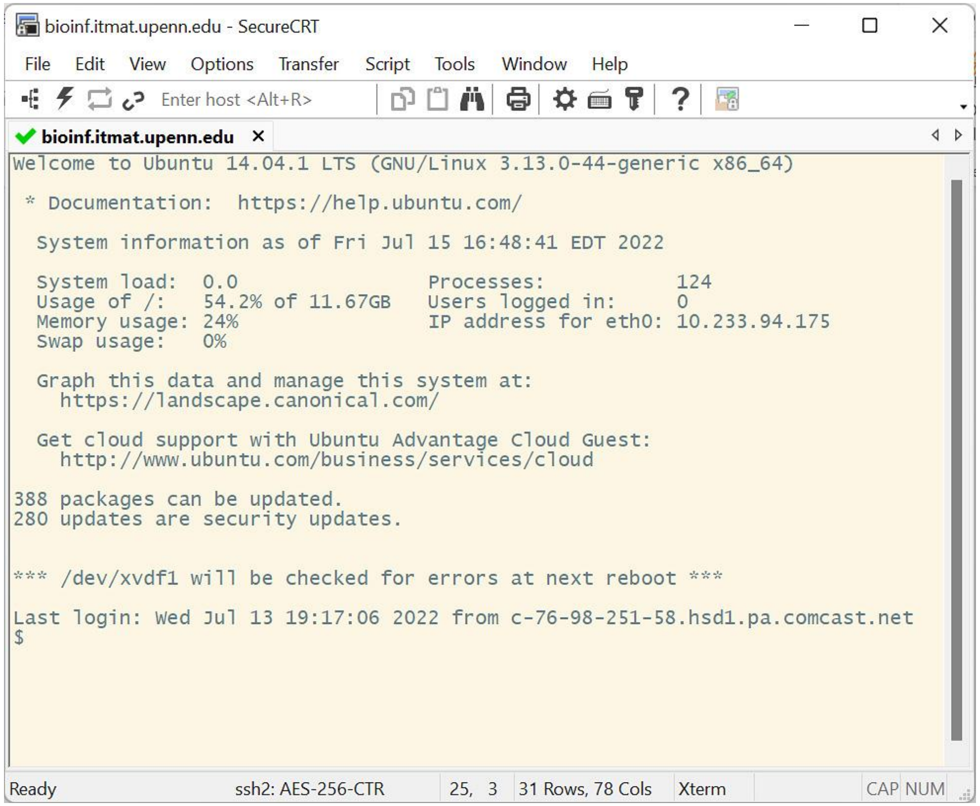
- Unix is also used to connect remotely to other machines.
- This may look the same as a previous slide, but in this case the program you see running in the window is not a Unix emulator controlling my computer.
- Instead, it's a program that connects to a remote machine that could be anywhere in the world and in this window, I control *that* computer, not mine.
  - Notice the program is called "SecureCRT" for encrypted connections to remote (unix) machines.



```
Connection to midphase.networkguild.org closed.
ggrant@strange:~/$ ssh bottom
Linux bottom.networkguild.org 5.18.0-0.bpo.1-amd64 #1 SMP PREEMPT_DYNAMIC Debia
n 5.18.2-1~bpo11+1 (2022-06-14) x86_64
Welcome to Bottom.  Bottom Feeders not welcome.
Last login: Fri May 13 13:23:18 2022 from 66.228.61.115
ggrant@bottom:~$
ggrant@bottom:~$
ggrant@bottom:~$
```

SSH Client

- The program that allows you to connect to a machine remotely is typically called an SSH Client.
- It’s a program that connects to a remote Unix system and allows you to access and control the computer as if you were there.


bioinf.itmat.upenn.edu - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Enter host <Alt+R>
bioinf.itmat.upenn.edu

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-44-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Fri Jul 15 16:48:41 EDT 2022

  System load:  0.0                Processes:           124
  Usage of /:   54.2% of 11.67GB   Users logged in:     0
  Memory usage: 24%                IP address for eth0: 10.233.94.175
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

388 packages can be updated.
280 updates are security updates.


*** /dev/xvdf1 will be checked for errors at next reboot ***

Last login: Wed Jul 13 19:17:06 2022 from c-76-98-251-58.hsd1.pa.comcast.net
$
```

The Web is mostly UNIX

- Most of the time you fetch a web page you are communicating with a Unix machine.
  - Unix machines behind web sites are called "Servers"
- Let's look at a *simple* web address (aka a URL)

`www.cnn.com/2022/08/04/world/mammoth-fossils-early-humans-scn/index.html`

- The first part `www.cnn.com` is the base of the address

`www.cnn.com/2022/08/04/world/mammoth-fossils-early-humans-scn/index.html`

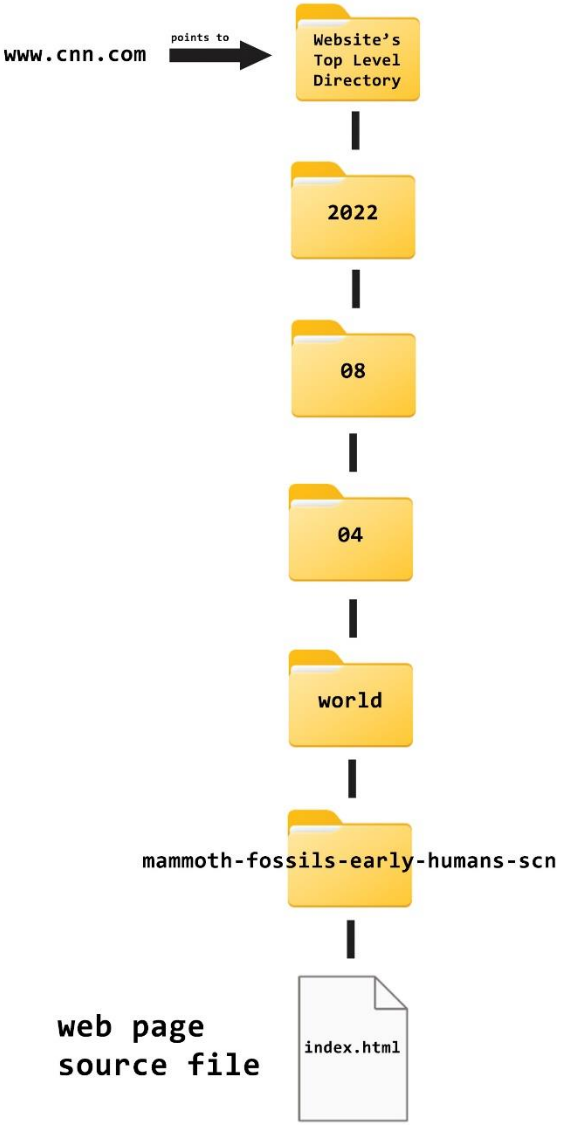
  - It points to a particular directory on a particular Unix machine.
- The next part is a path of subdirectories, separated by slashes:

`www.cnn.com/2022/08/04/world/mammoth-fossils-early-humans-scn/index.html`

- And the last part is a file in the last directory

`www.cnn.com/2022/08/04/world/mammoth-fossils-early-humans-scn/index.html`

  - This file tells the web browser how to render the page.



Clarification: URLs

- We just examined an old-school basic web address for a static web page.
  - And you’ll see plenty of addresses across the web formatted like that.
- But they don’t all conform to that format.
  - Some URL’s use hidden variables to direct you to the right place, and some URL’s have explicit variables specified.
  - Anything that follows a ‘?’ in a URL is a variable.

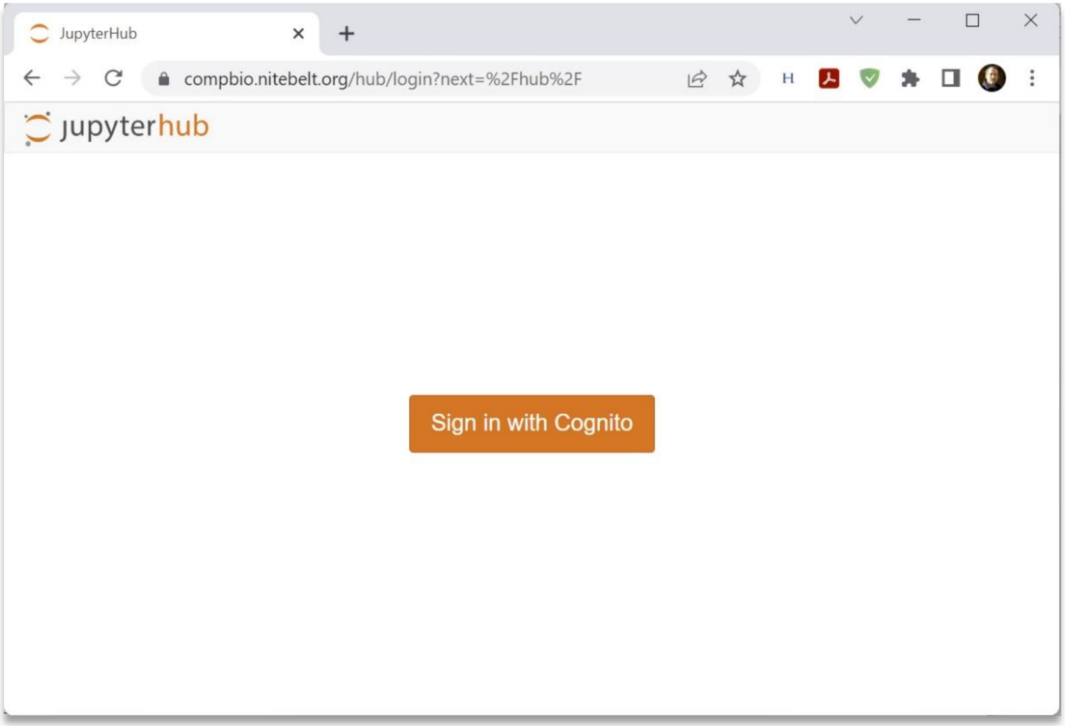
Clarification: Non-Unix Servers

- Non-Unix servers do exist, you may occasionally be fetching a web page from a server running Windows or something else.
- But for the most part, servers tend to be Unix.

- You don’t need to own a powerful computer anymore; you can simply rent one by the hour from Amazon.
  - They don’t mail it to you, you connect to it remotely using an SSH Client.
  - These ephemeral machines are known collectively as The Cloud.
- *For this class we have rented a machine from Amazon that everybody is going to have an account on.*
- Installing an SSH Client can be tricky, you need to set up private and public encryption keys.
  - Fortunately, you won’t need to do that.
  - We have set up a web browser interface instead.

The BIOL4536 Server

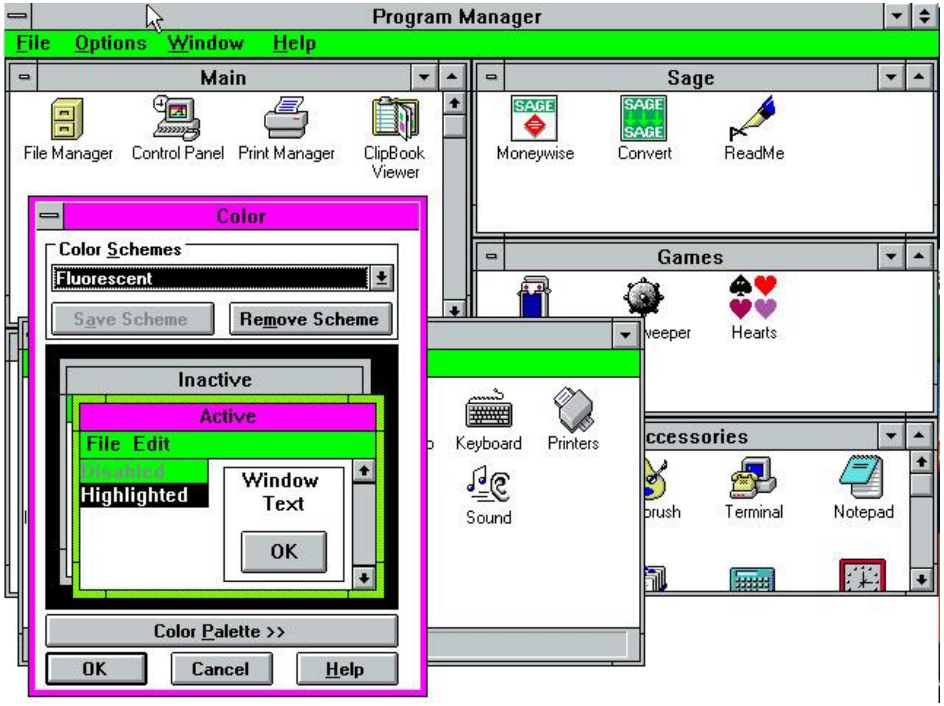
- We will all connect with the same web-based interface.
  - This way we'll all have the same experience.
- The server is here:
  - https://biol4536.itmat.org
  - You will receive a username and password by email.



Why is Unix Text Only?

- When they added slick things like pretty graphics and pointing devices, they also added a whole lot of baggage.
  - – Resulting in a whole lot of inefficiency.

- Particularly if you're running a computer remotely from across the planet, sending all the extra information to render windows and graphics is not practical.
- Text, on the other hand, can be transmitted very fast and efficiently.



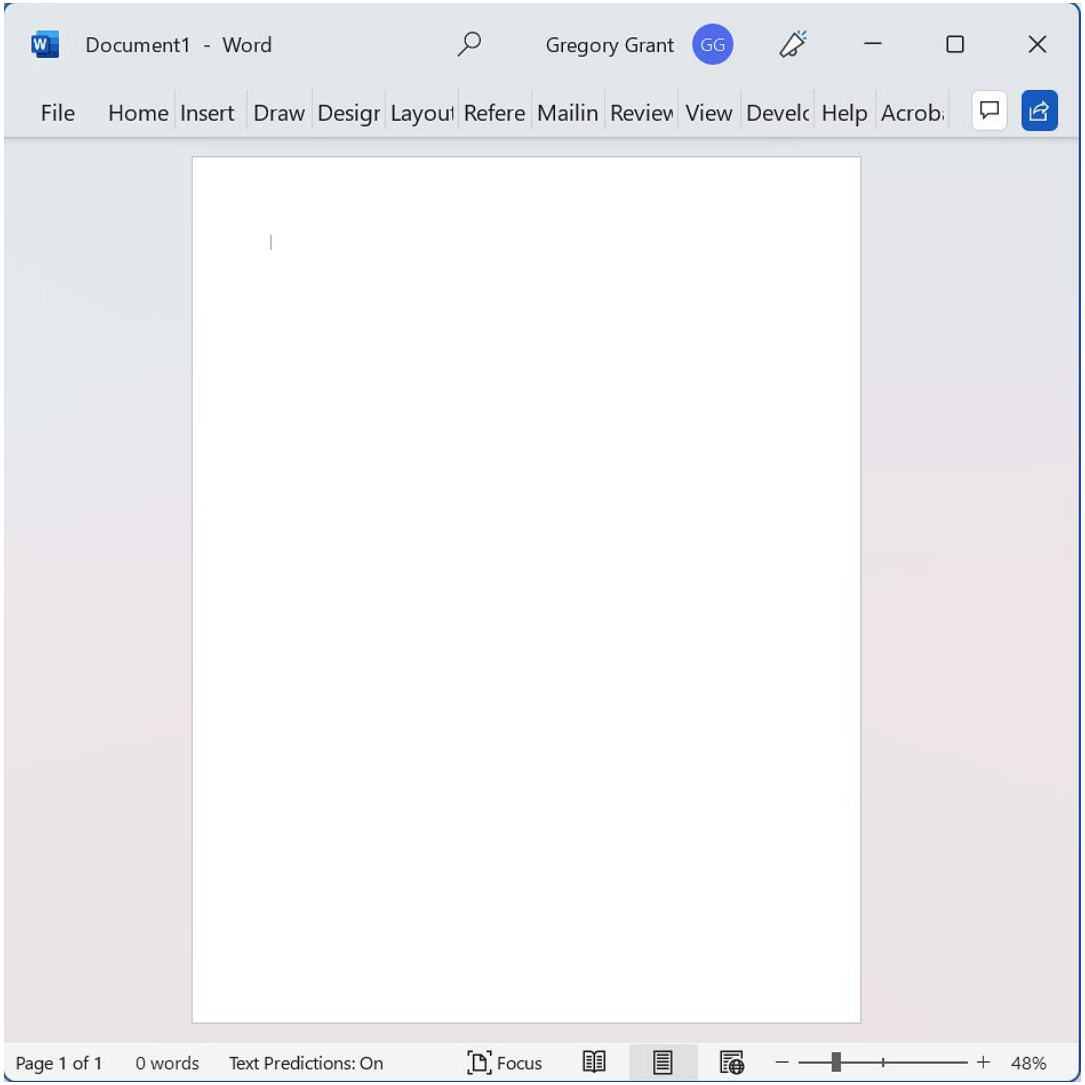
Loss of Functionality

- With graphics we gain aesthetics and make operating systems intuitive.
  - But in the process, we sacrifice a lot of power and functionality that you can only get with a text-based interface.
- For example, suppose you wanted to find all files on your computer with names that start with a 'A', end in a 'T', and do not have an 'R' anywhere in the middle.
  - I'm not sure how you'd even being to do that with Windows Explorer or Finder.
  - Unix makes such things easy.
- Of course, most people never need to do esoteric searches like that.
  - Bioinformatics presents us with all sorts of problems we do not normally encounter.
- UNIX makes many things easy.

A Word on Text Editors

- Many text editors, such as Microsoft Word, save more than just what you type.
  - They save additional formatting information such as the font, whether it is italic, bold, the size the margins, etc..
- Let’s create a simple text file in Word and see what happens.

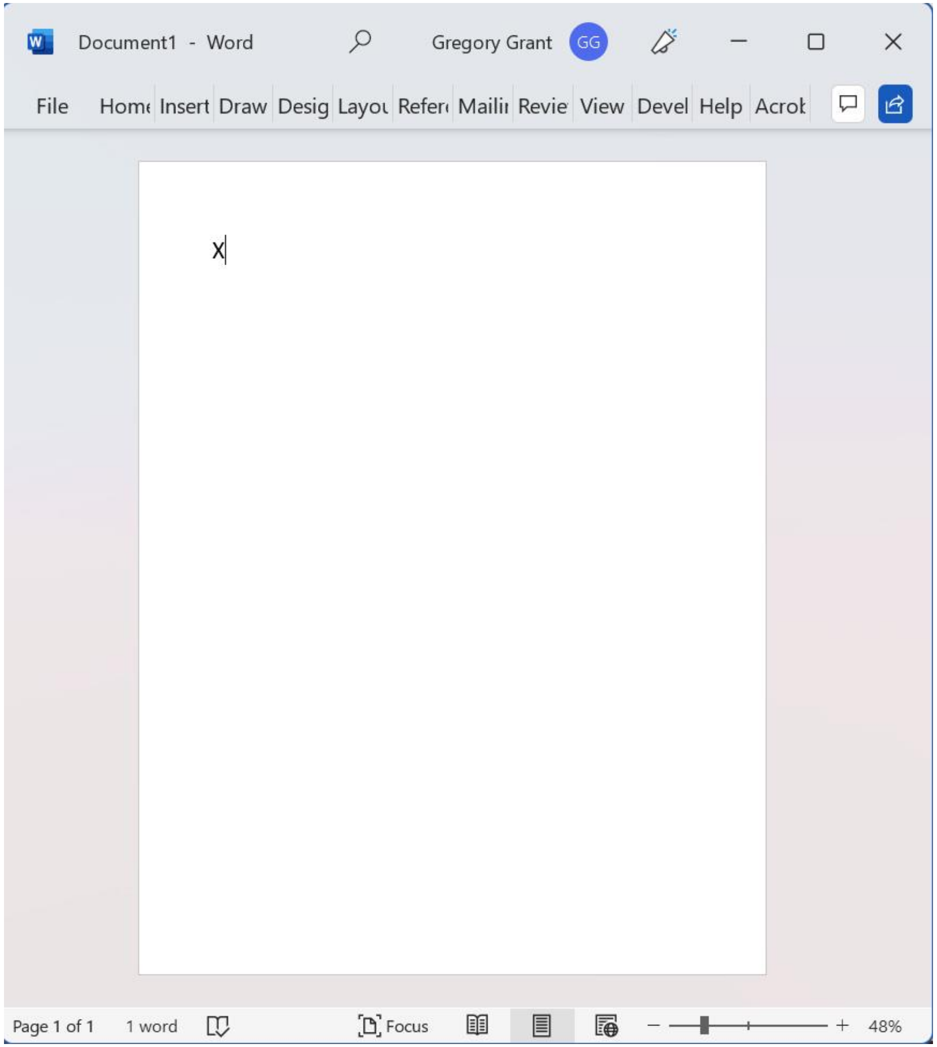
Microsoft Word Blank Document



Text Predictions: On

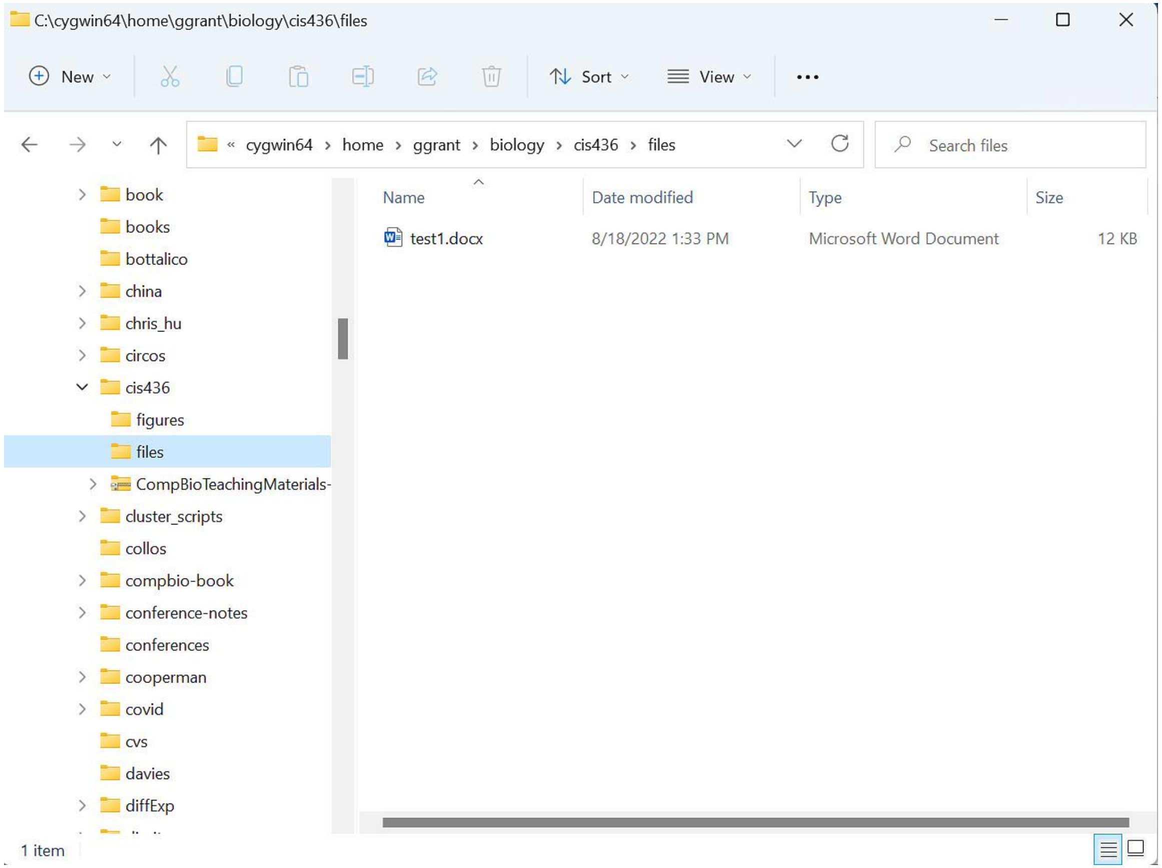
Microsoft Word Small File

- We’ve typed the character ‘X’ and nothing else.
- A file can’t get much smaller than this.



File in Windows Explorer

- We see the file is 12KB in size.
- It’s exactly 11,981 bytes.
- One byte is 8 bits, so 95,848 bits to save ‘X’



C:\cygwin64\home\ggrant\biology\cis436\files

Name	Date modified	Type	Size
test1.docx	8/18/2022 1:33 PM	Microsoft Word Document	12 KB

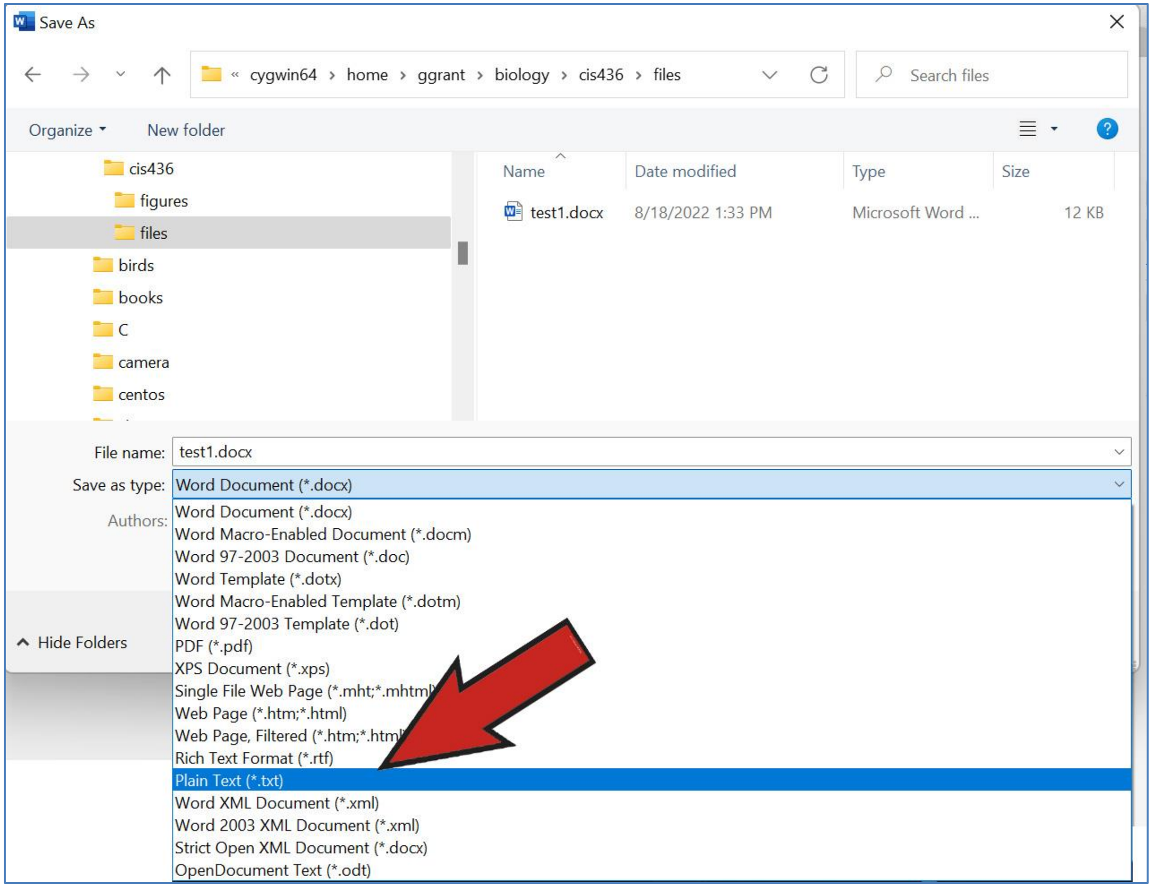
1 item

Baggage

- The ASCII code encodes ‘X’ with one byte.
- So, what is Word doing with the other 11,980 bytes?
- Formatting information
  - Font family, size, color, style, etc.
- Registration information
  - Your name, address and more.
- Other?
  - Who knows, much of it is inscrutable.

Text Files

- We’re going to save the file again, but this time we’re going to save it in ‘text’ format.

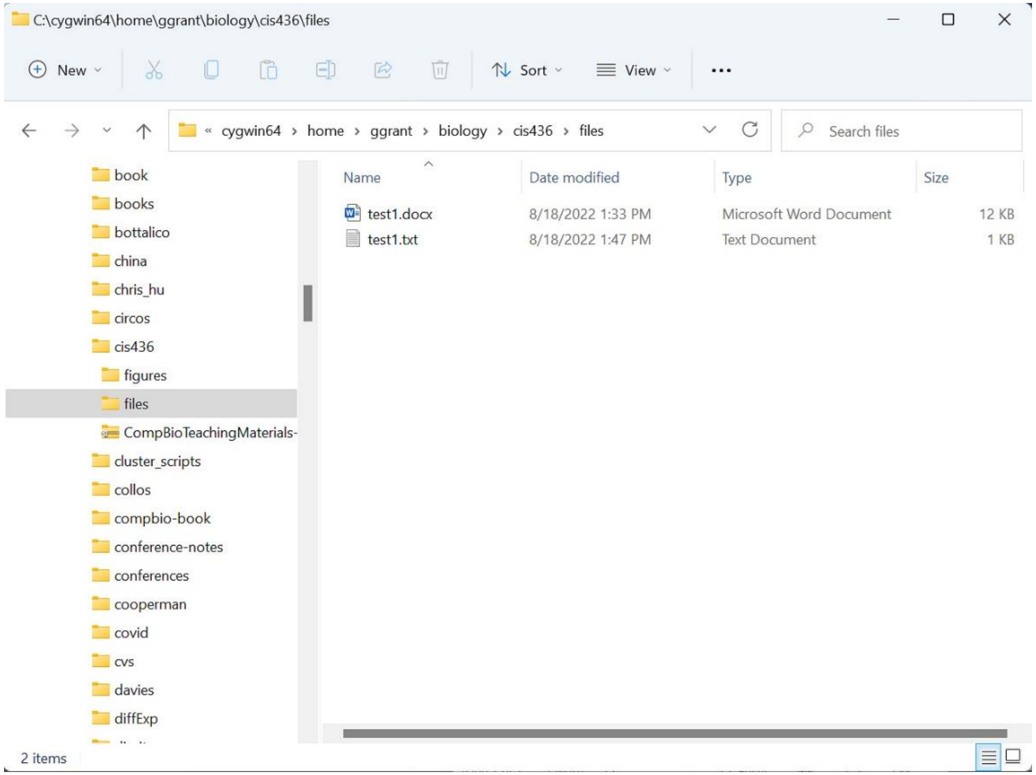

Save As
cygwin64 › home › ggrant › biology › cis436 › files
Search files
Organize
New folder
cis436
figures
files
birds
books
C
camera
centos
Name
Date modified
Type
Size
test1.docx
8/18/2022 1:33 PM
Microsoft Word ...
12 KB
Authors:
Hide Folders
Word Document (*.docx)
Word Macro-Enabled Document (*.docm)
Word 97-2003 Document (*.doc)
Word Template (*.dotx)
Word Macro-Enabled Template (*.dotm)
Word 97-2003 Template (*.dot)
PDF (*.pdf)
XPS Document (*.xps)
Single File Web Page (*.mht;*.mhtml)
Web Page (*.htm;*.html)
Web Page, Filtered (*.htm;*.html)
Rich Text Format (*.rtf)
Plain Text (*.txt)
Word XML Document (*.xml)
Word 2003 XML Document (*.xml)
Strict Open XML Document (*.docx)
OpenDocument Text (*.odt)

File name: test1.docx

Save as type: Word Document (*.docx)

File in Windows Explorer

- Finder now reports the file size as 1KB.
- That’s because 1KB is the smallest thing it will report.
  - In reality, the file size on the disk is 3 bytes.
  - One byte for the ‘X’ and it used two more bytes to indicate the newline character at the end of the line.

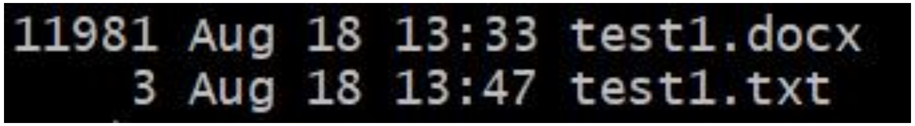


C:\cygwin64\home\ggrant\biology\cis436\files

Name	Date modified	Type	Size
test1.docx	8/18/2022 1:33 PM	Microsoft Word Document	12 KB
test1.txt	8/18/2022 1:47 PM	Text Document	1 KB

Unix Display

- This is how Unix displays the two files.
- It gives the exact file size in bytes.



```
11981 Aug 18 13:33 test1.docx
    3 Aug 18 13:47 test1.txt
```

- If we created the text file in Unix, it would be only 2 bytes, because Unix only uses 1 byte to indicate a newline.

Text Editors

- To work with the command-line, or to program,or to make web pages, or to manipulate files of biological data, you must create and edit text files without the baggage.
  - On Windows, as we have seen, you can make Word save files as plain text.
    - Or you can use NotePad, WordPad, and EditPad.
  - In UNIX all editors are text friendly editors. But the best ones are not intuitive and have steep learning curves.
    - VI, Emacs are popular hardcore ones.
    - Pico and Nano are basic intuitive ones.

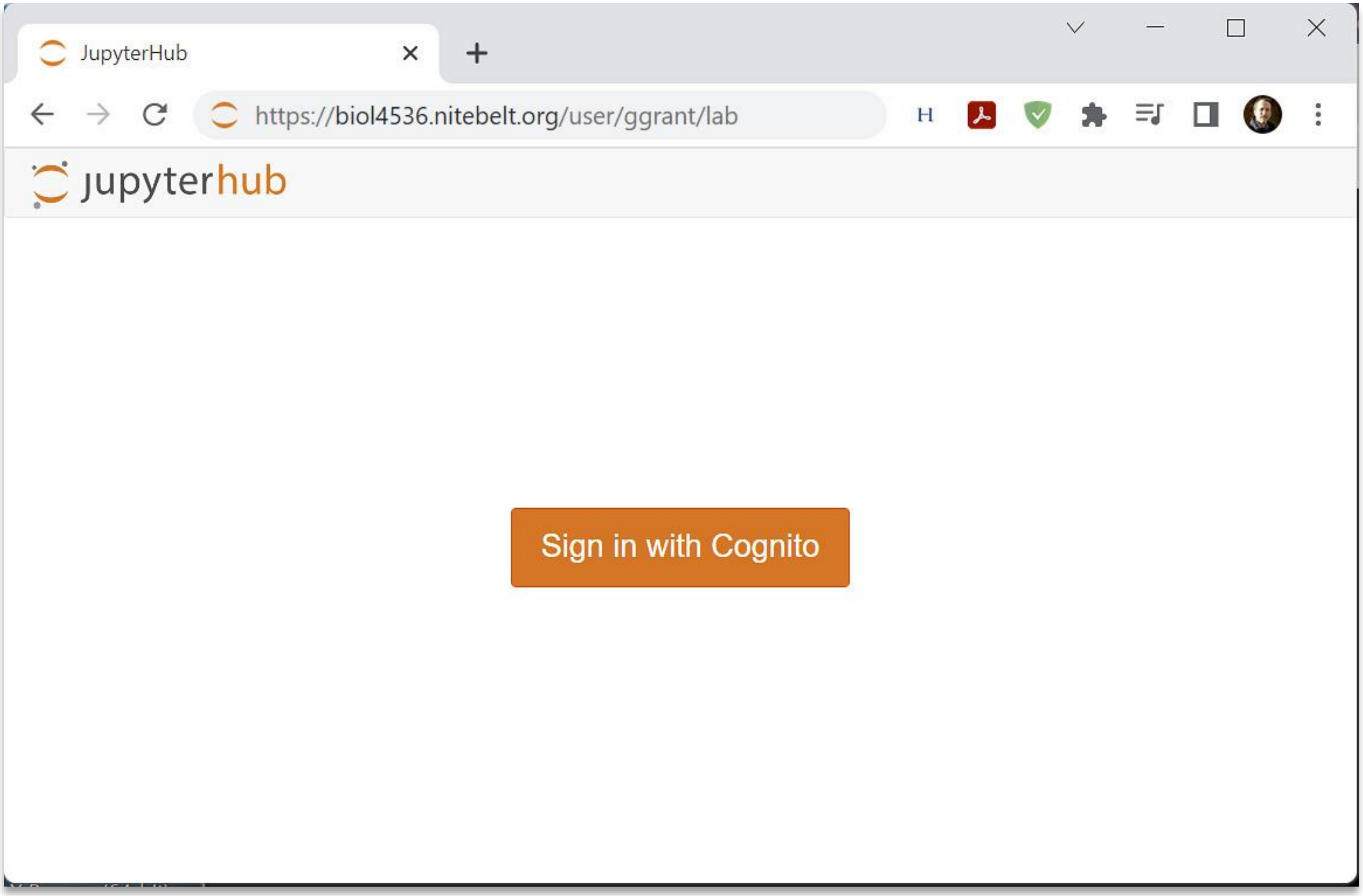
Using the Command Line

- Almost everything you can do with Windows Explorer (or Finder on Mac) you can do command-line, plus a *lot more*.
  - Not absolutely everything is better on the command-line however, so an efficient person must have control of both command-line and GUI access to their system.
    - Windows Explorer (Windows) and Finder (Mac) are **GUI**'s, which stands for "Graphics User Interfaces".
- We can use the command-line to:
  1. Create folders, move files around and rename things.
  2. Run programs
  3. Manipulate and search through text files
  4. Connect remotely to other Unix machines.

  Etc.

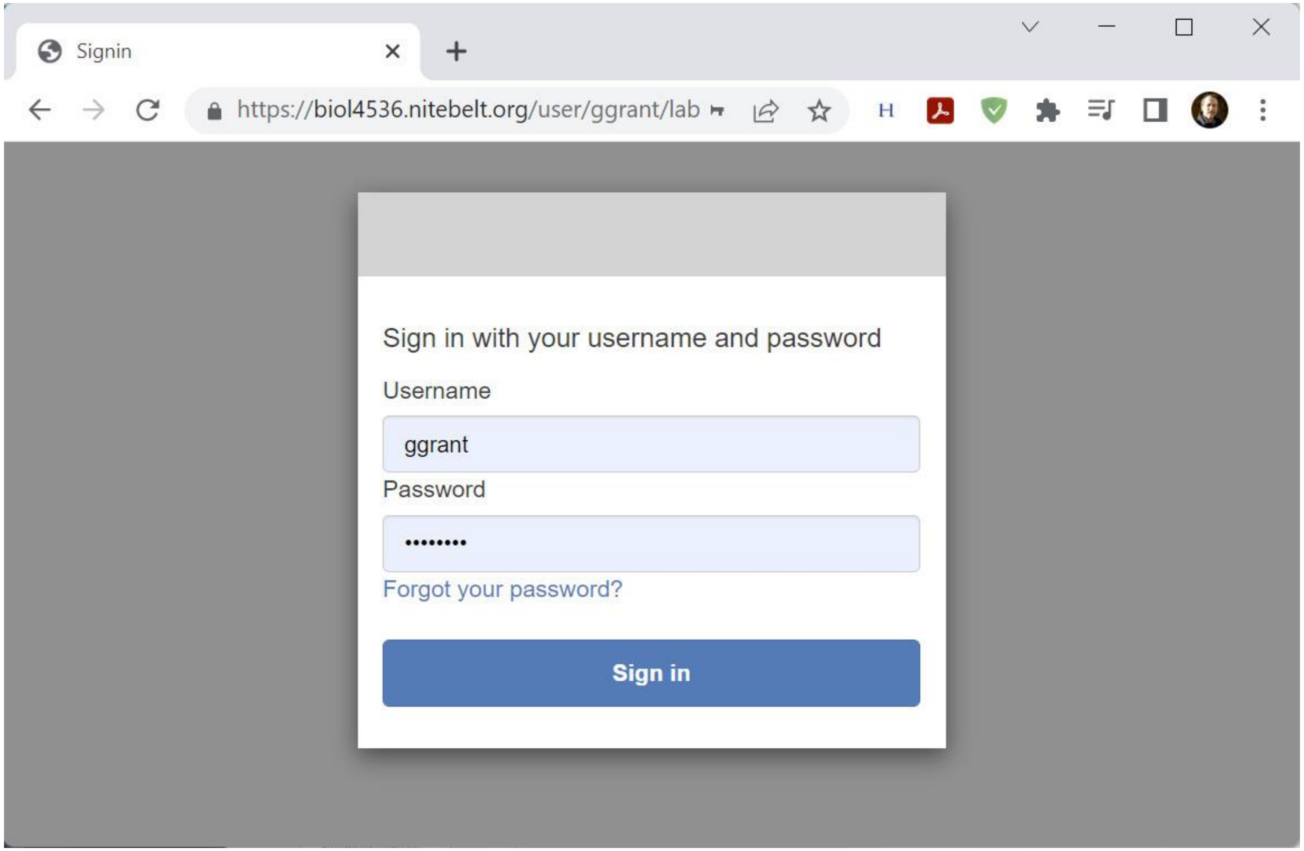
Connecting to the BIOL4536 Server

- The address is: https://biol4536.itmat.org



Log In

- You were (or will be) emailed credentials to log in.



Sign in with your username and password

Username

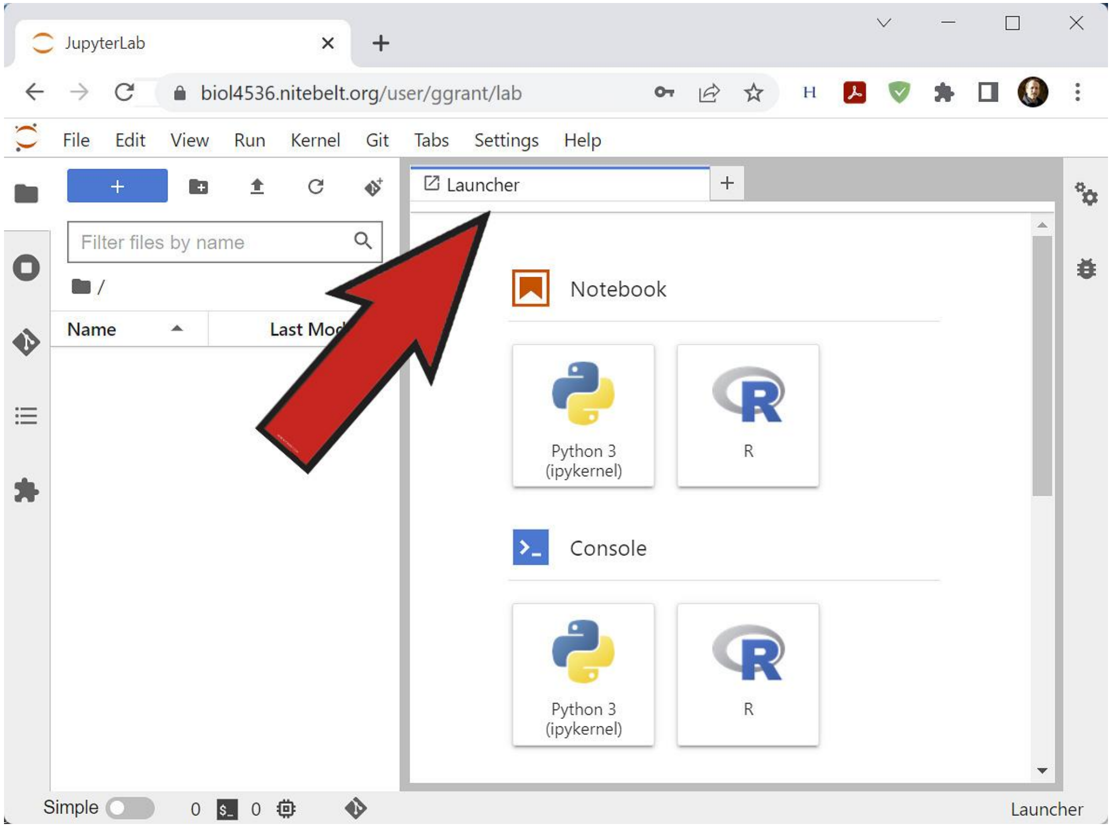
ggrant

Password

••••••••

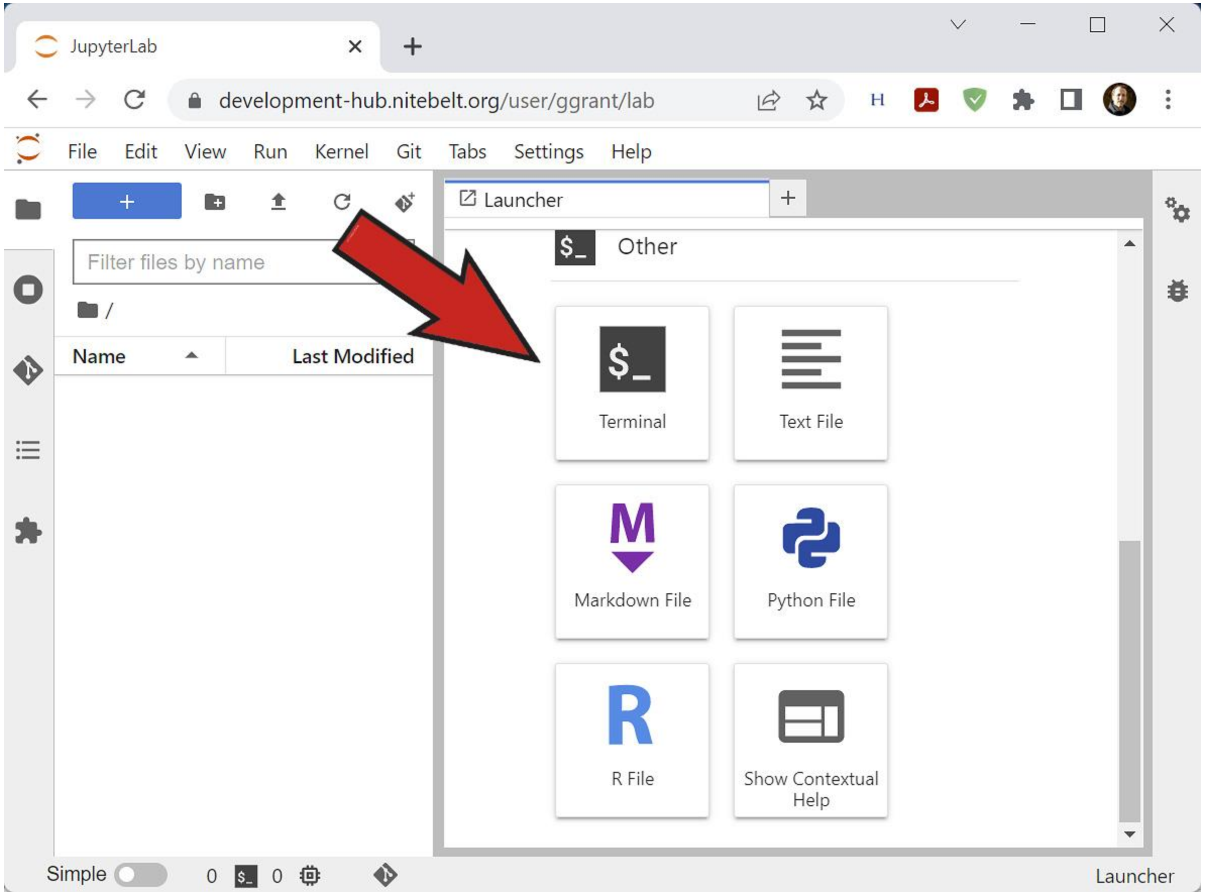
Jupyter

- This is a Jupyter interface.
  - It allows us to embed a Unix interface in a web page.
  - This is convenient for this class but is not typically how you would access Unix. Typically, you'd install an SSH client.
- On the right is a panel from where you can launch several services.



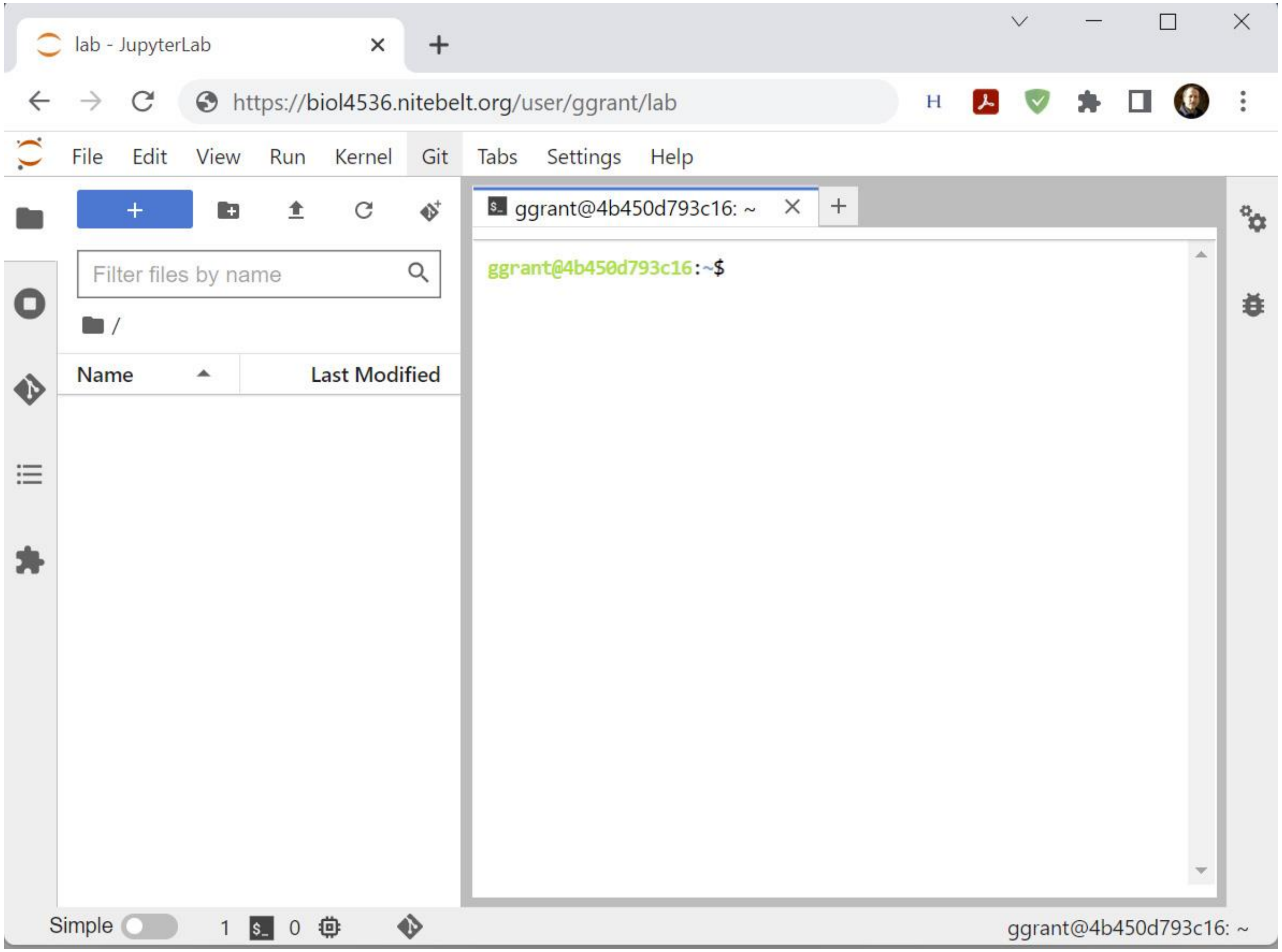
Launch a Terminal

- Scroll down and click on Terminal.



Launch a Terminal

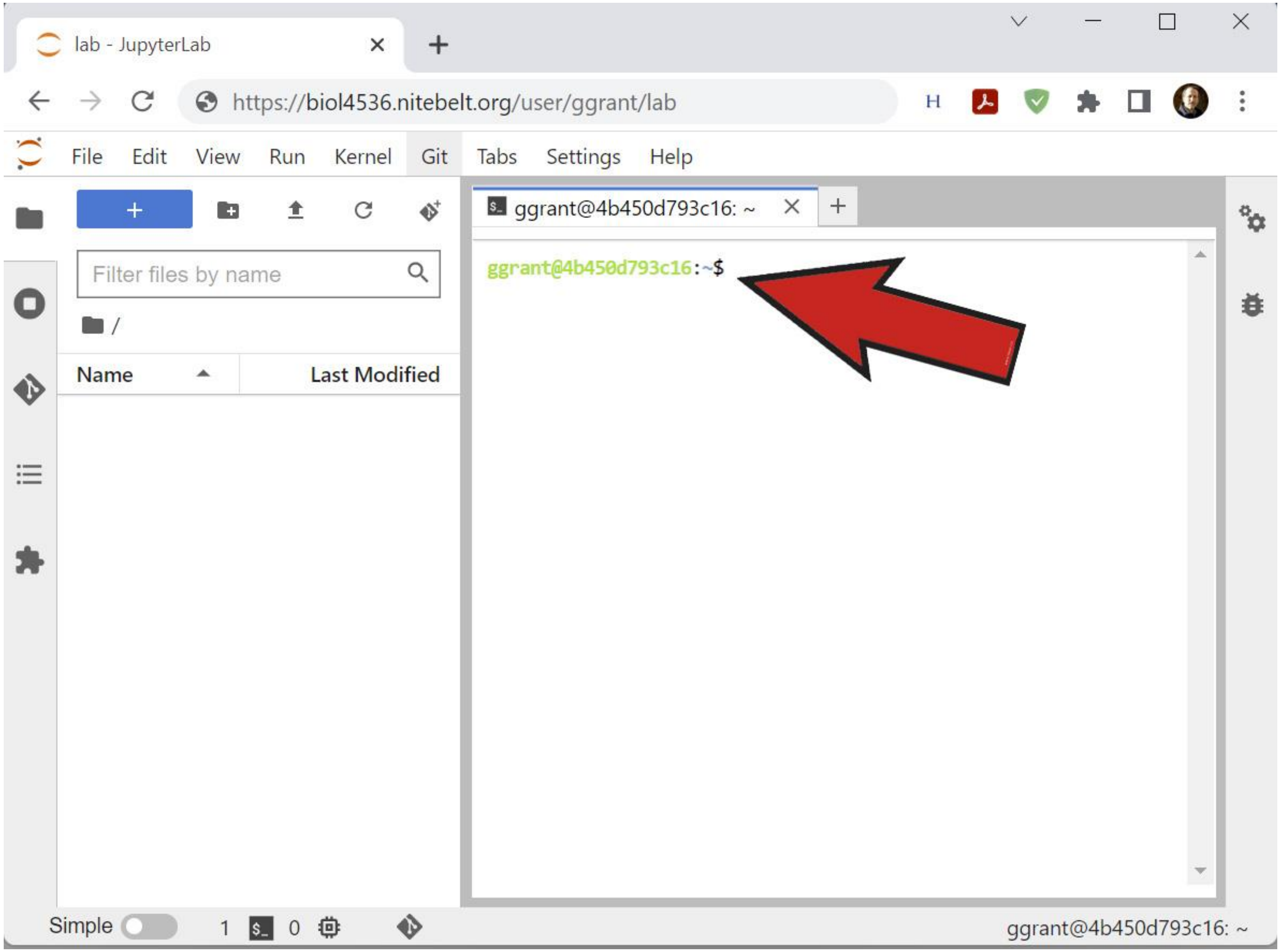
- This is what you should see.



```
ggrant@4b450d793c16:~$
```

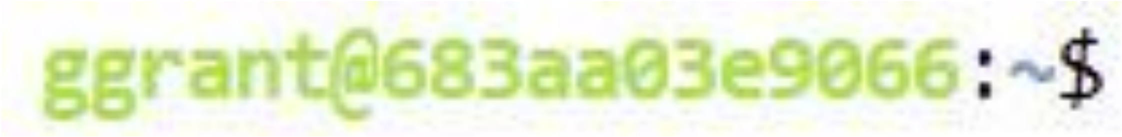
The Prompt

- This is the so-called “prompt” where you enter your commands.



```
ggrant@4b450d793c16:~$
```

The Prompt



```
ggrant@683aa03e9066:~$
```

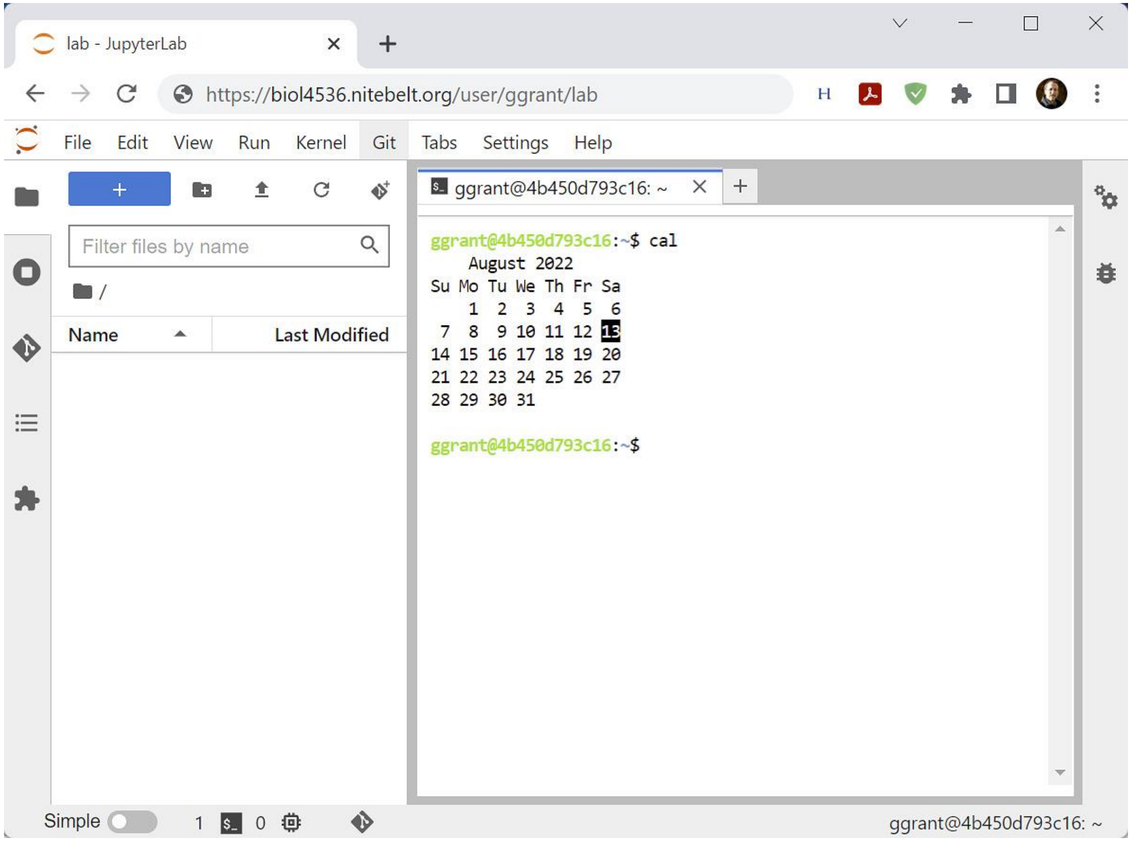
- The prompt shows your username, followed by an @ symbol, followed by an ID that identifies the session.
  - That ID might change from time to time, you don’t need to worry about that.
- We’ll explain later what the :~ is but basically it tells you where you are in the file system.
- The default prompt is a bit ugly with that ID showing.
  - But it is highly configurable, so we can change it and it does not always look like this.
  - We’re not going to get into prompt configuration, because time is limited and it’s just a ‘look and feel’ issue.

Commands

- In UNIX you type a command at the prompt and hit enter.
  - We will also use the term ‘function’ as a synonym for ‘command’
- When you do this, you are essentially just running a program.
- There are directories on your system where these programs live.
- When you enter a command, the system looks in these directories for the command.
  - If it can’t find it, it will return “Command Not Found” or something similar.

cal for example

- Type "cal" at the prompt and hit Enter.
- The "cal" function displays a nifty little calendar of the current month.
  - Followed by the prompt, ready for the next command.



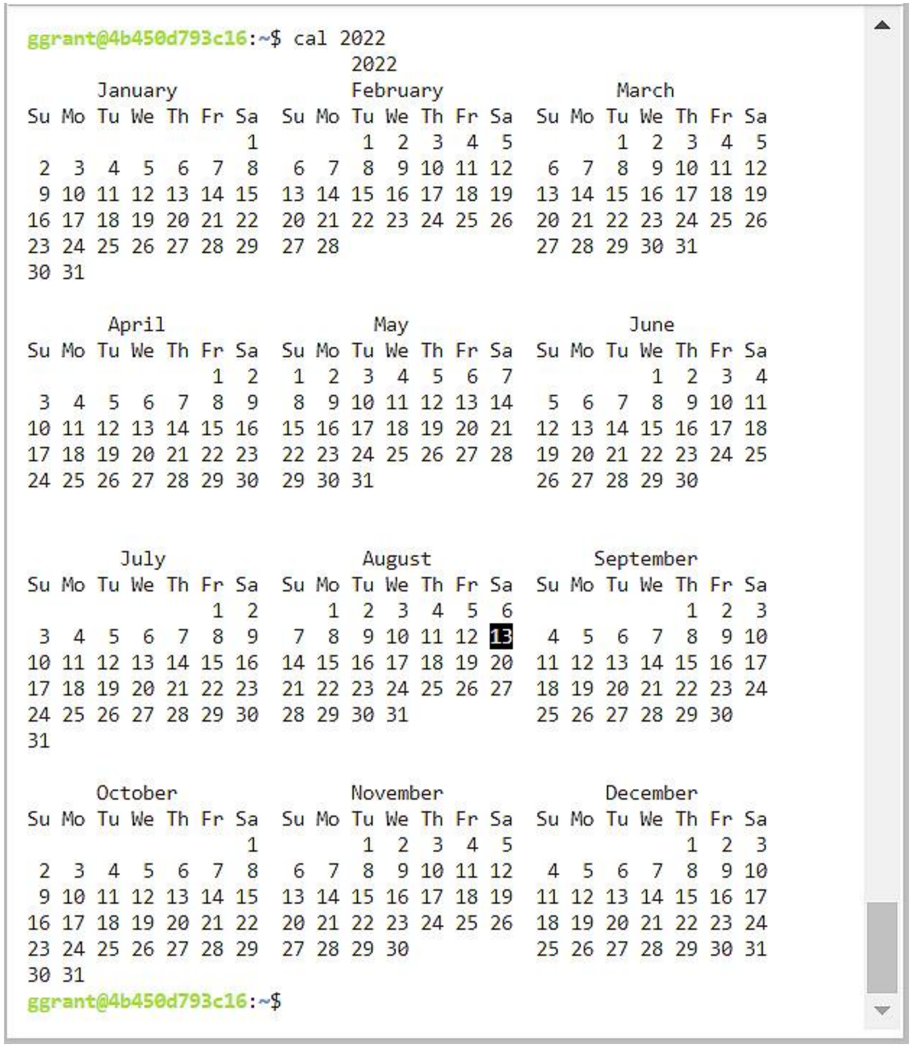
```
ggrant@4b450d793c16:~$ cal
    August 2022
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

ggrant@4b450d793c16:~$
```

Options

- Commands almost always have options.
- Options are separated from the command by spaces.
- For example, the following will show the entire year:

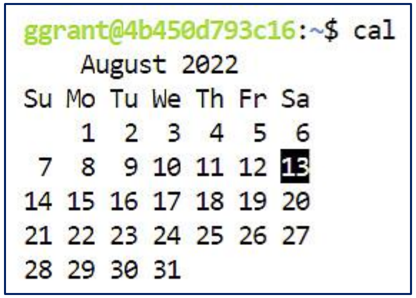
➢ cal 2022
  - cal is the command
  - 2022 is the option to the command

- We'll see shortly how to get a list of all options to any command.



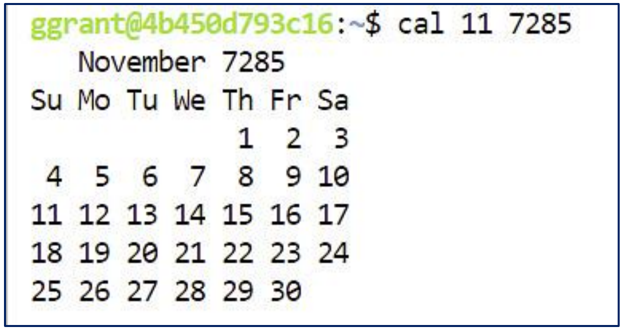
```
ggrant@4b450d793c16:~$ cal 2022
                            2022
      January               February               March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1         1  2  3  4  5         1  2  3  4  5
 2  3  4  5  6  7  8   6  7  8  9 10 11 12   6  7  8  9 10 11 12
 9 10 11 12 13 14 15  13 14 15 16 17 18 19  13 14 15 16 17 18 19
16 17 18 19 20 21 22  20 21 22 23 24 25 26  20 21 22 23 24 25 26
23 24 25 26 27 28 29  27 28                 27 28 29 30 31
30 31

       April                  May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                1  2   1  2  3  4  5  6  7            1  2  3  4
 3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
24 25 26 27 28 29 30  29 30 31              26 27 28 29 30


        July                 August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                1  2      1  2  3  4  5  6               1  2  3
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   4  5  6  7  8  9 10
10 11 12 13 14 15 16  14 15 16 17 18 19 20  11 12 13 14 15 16 17
17 18 19 20 21 22 23  21 22 23 24 25 26 27  18 19 20 21 22 23 24
24 25 26 27 28 29 30  28 29 30 31           25 26 27 28 29 30
31

      October               November              December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1         1  2  3  4  5               1  2  3
 2  3  4  5  6  7  8   6  7  8  9 10 11 12   4  5  6  7  8  9 10
 9 10 11 12 13 14 15  13 14 15 16 17 18 19  11 12 13 14 15 16 17
16 17 18 19 20 21 22  20 21 22 23 24 25 26  18 19 20 21 22 23 24
23 24 25 26 27 28 29  27 28 29 30           25 26 27 28 29 30 31
30 31
ggrant@4b450d793c16:~$
```

cal



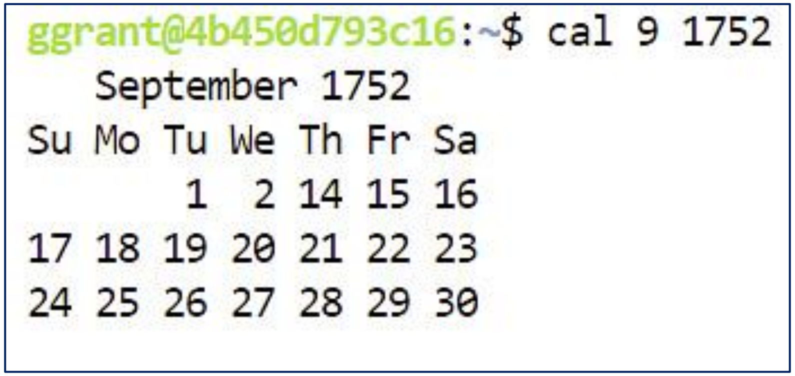
```
ggrant@4b450d793c16:~$ cal
    August 2022
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

- The cal command illustrates how everything in Unix is taken to the geek level.
- You surely have your go-to calendar that you use daily.
- But does it display any year from the year 1 to the year 9999?
- Here is what November will look like in the year 7285.



```
ggrant@4b450d793c16:~$ cal 11 7285
   November 7285
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

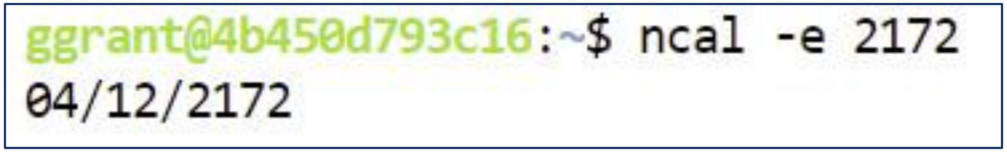
cal



```
ggrant@4b450d793c16:~$ cal 9 1752
   September 1752
Su Mo Tu We Th Fr Sa
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

- This is September of the year 1752 when they had to skip 12 days.
  - When the Julian Calendar was replaced by the Gregorian Calendar.

Options and Parameters



```
ggrant@4b450d793c16:~$ ncal -e 2172
04/12/2172
```

- This command returns the date of Easter in the year 2172.

  `ncal` is the command

  `-e` is an option to `ncal`

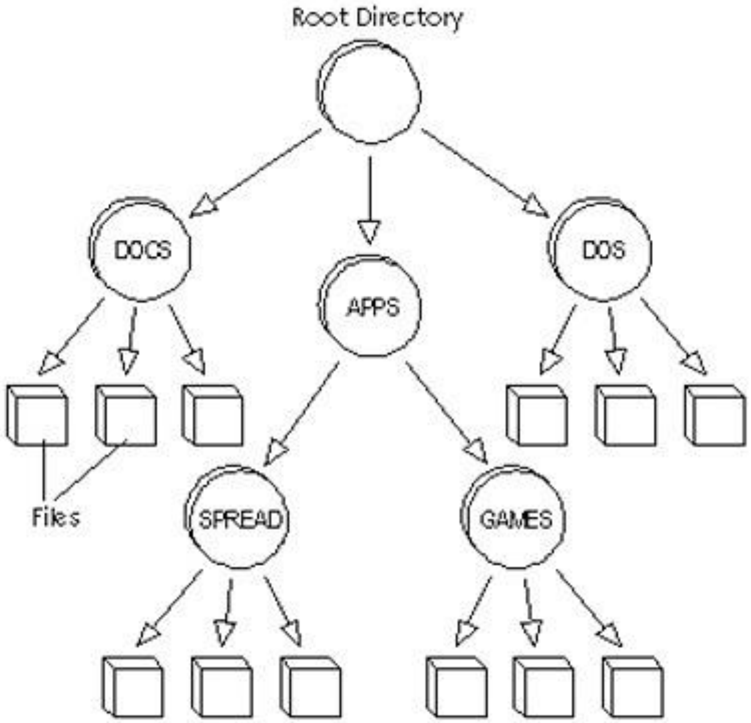
  2172 in this case is a parameter to the `–e` option.

  - `-e` means ‘return the date of easter’ but it needs to know what year, which is specified by the parameter.

- Some options take parameters, some do not.
  - We will see soon how this is specified in Unix documentation.

The File System

- The file system is organized like an upside-down tree.
- Files are stored in ‘directories’
- Every directory has a parent directory.
  - With one exception, the directory at the very top called the Root Directory.

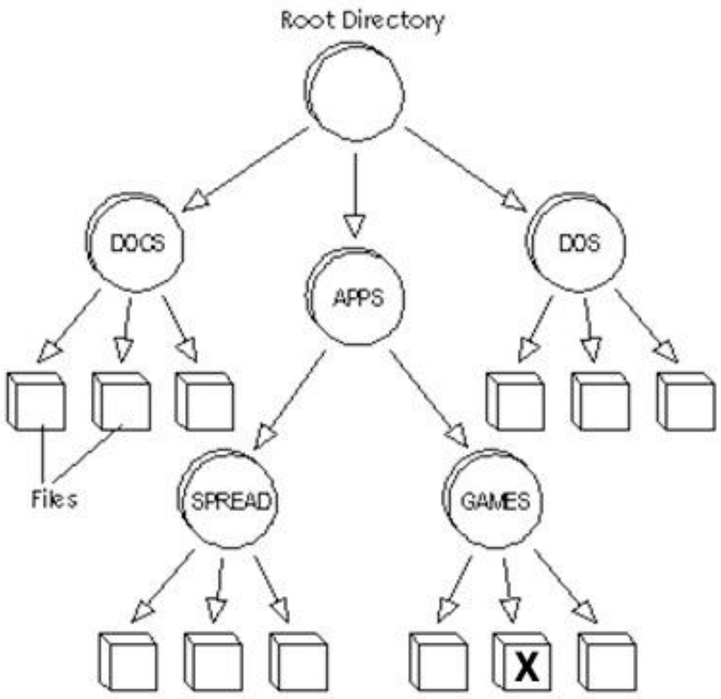


The Current Working Directory

- When you have a Unix terminal open, you are always somewhere.
  - You have a current location in the file system.
  - It’s called your ‘current working directory’
- When you log in, you start off in your home directory.
  - Every user has their own personal “home” directory.
  - And you can move around the file system from there.
    - We’ll learn the commands move around soon.

The File System

- Every directory has a unique path back to the root.
- On Unix this path is represented by directory names, separated by forward slashes.



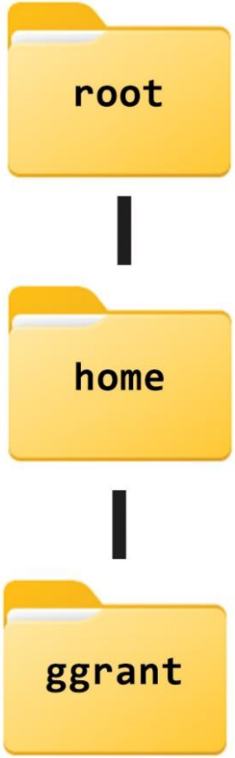
- For example, the file labeled ‘X’ in the picture is specified as

```
/APPS/GAMES/X
```

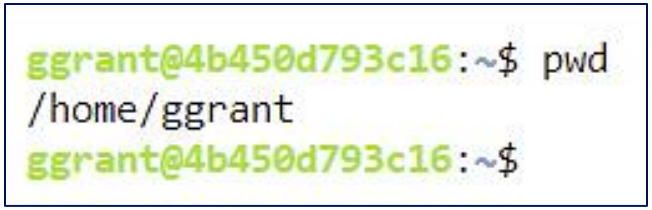
- The root directory itself is specified by just a single forward slash:

```
/
```

pwd



- To find out where you are, enter the command ‘pwd’.
  - Which stands for ‘print working directory’.



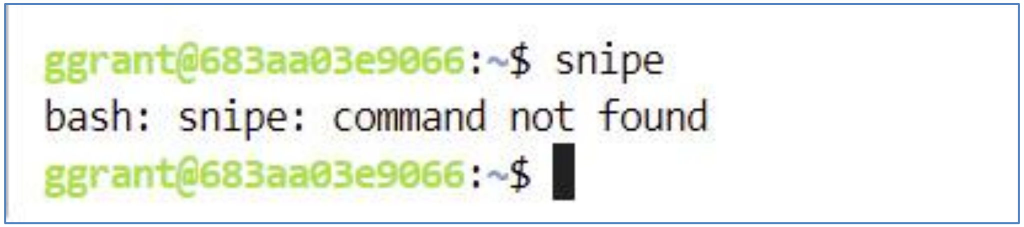
```
ggrant@4b450d793c16:~$ pwd
/home/ggrant
ggrant@4b450d793c16:~$
```

- /home/ggrant

This means we’re currently two levels down from the root directory.

  - In the root directory is a directory called ‘home’ (as well as others)
  - In the ‘home’ directory there’s a directory called ‘ggrant’ (as well as others)
  - And the ggrant directory is where I was when I entered pwd.

command not found



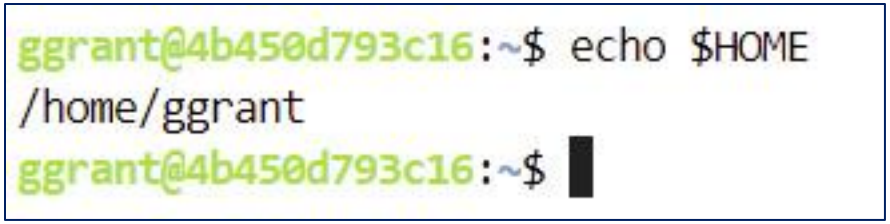
```
ggrant@683aa03e9066:~$ snipe
bash: snipe: command not found
ggrant@683aa03e9066:~$ █
```

- If you see this error, it means the command has not been installed on your system for some reason.
  - Or maybe you typed it in wrong.
- Any given installation of Unix will vary slightly from any other by what commands are installed.
- But you can add missing programs by searching the web for them and plopping them in the right directory.

  /bin on our system
- Or you can use a program called a package manager to find and install missing programs.

Finding Your Way Around

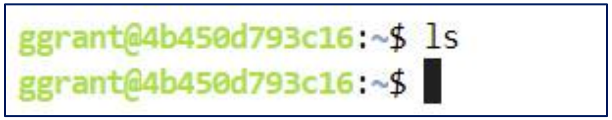
- When you get an account on a UNIX system, you should have a particular directory called your “home directory”.
  - No matter where you are, you can always get back to your home directory by entering "`cd`" at the command-prompt.
- Unix maintains some 'variables’ for you and the location of your home directory is in a variable called `$HOME`.
  - Enter "`echo $HOME`" and see what happens.



```
ggrant@4b450d793c16:~$ echo $HOME
/home/ggrant
ggrant@4b450d793c16:~$
```

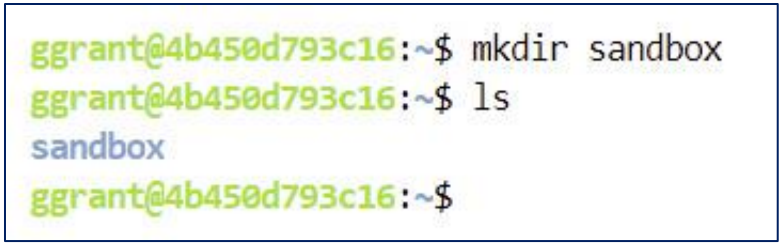
Listing and Making Directories

- To list the contents of the current directory, enter 'ls'
  - At this point there probably are no files or directories in your home so this shouldn't return anything.



```
ggrant@4b450d793c16:~$ ls
ggrant@4b450d793c16:~$ █
```

- To make a subdirectory of the current directory use the 'mkdir' command.
  - Enter 'mkdir sandbox' to create a directory called 'sandbox'.
- Enter 'ls'again and you should now see 'sandbox' on the list.



```
ggrant@4b450d793c16:~$ mkdir sandbox
ggrant@4b450d793c16:~$ ls
sandbox
ggrant@4b450d793c16:~$
```

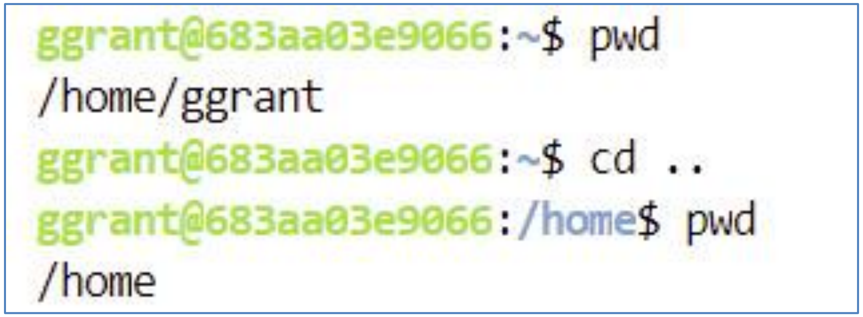
Moving Up the Tree

- To move up one level in the file system tree, simply enter:

➢ cd ..

  – That's cd followed by a space followed by two periods.

- You don't need to specify the directory because there's a unique directory one level up.
  – But you do have to work harder to move down.
- The following illustrates moving one level up from your home directory:

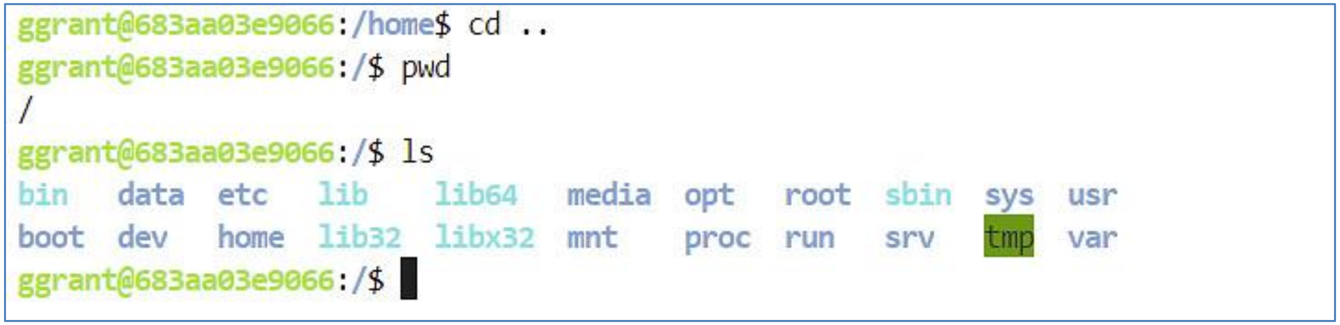


```
ggrant@683aa03e9066:~$ pwd
/home/ggrant
ggrant@683aa03e9066:~$ cd ..
ggrant@683aa03e9066:/home$ pwd
/home
```

  – Notice how the prompt itself is telling you your location.

The Root Directory

- If we do ‘cd ..’ again it will take us to the root directory, the top level of the tree.
- Doing an ‘ls’ there shows us the contents of the root directory.

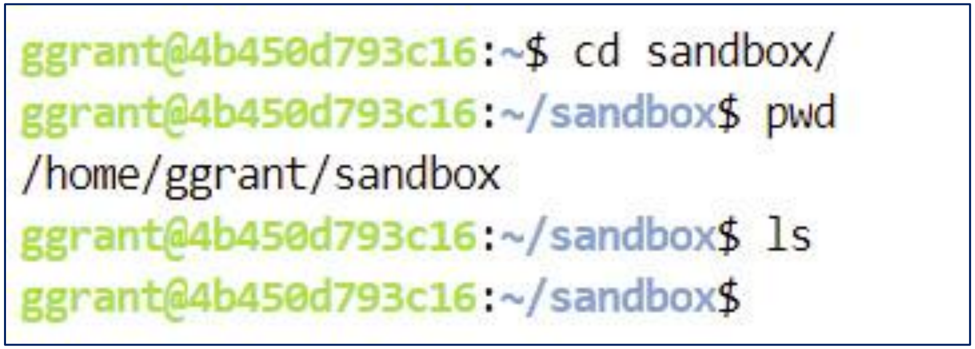


```
ggrant@683aa03e9066:/home$ cd ..
ggrant@683aa03e9066:/$ pwd
/
ggrant@683aa03e9066:/$ ls
bin   data  etc   lib    lib64   media  opt   root  sbin  sys  usr
boot  dev   home  lib32  libx32  mnt    proc  run   srv   tmp  var
ggrant@683aa03e9066:/$
```

- Don’t be confused by the fact that there’s a directory on the list called ‘root’.
  - You can name a directory ‘root’ if you want, but it’s still not The Root directory.
  - The Root directory is simply called /
    - It is not called not /root

Moving Down the Tree

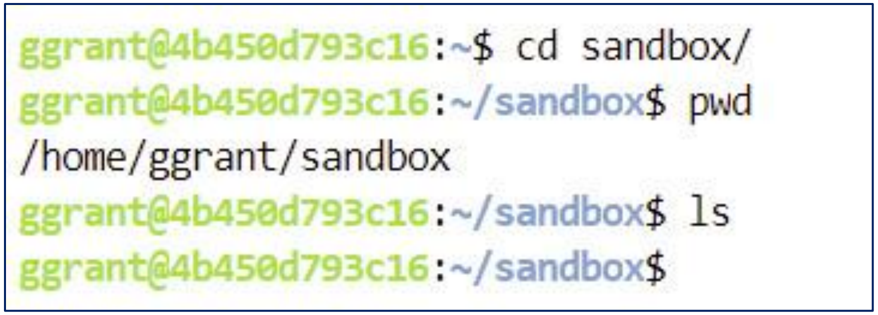
- Enter ‘cd’ to get back to your home directory.
- To move into the sandbox directory, enter
- ➢ `cd sandbox`
  - – cd is short for 'change directory’.
  - – The sandbox directory should now be the current directory
  - – Enter `'pwd'` to make sure you are in the sandbox.
  - – Enter `'ls'` to see the contents of `'sandbox'` (it should be empty).



```
ggrant@4b450d793c16:~$ cd sandbox/
ggrant@4b450d793c16:~/sandbox$ pwd
/home/ggrant/sandbox
ggrant@4b450d793c16:~/sandbox$ ls
ggrant@4b450d793c16:~/sandbox$
```

The Prompt (again)

- Now that we’ve moved into the sandbox, the prompt reflects that.
- The tilde ~ is shorthand for your home directory.
- So ~/sandbox means you are one level below your home in a directory called ‘sandbox’



```
ggrant@4b450d793c16:~$ cd sandbox/
ggrant@4b450d793c16:~/sandbox$ pwd
/home/ggrant/sandbox
ggrant@4b450d793c16:~/sandbox$ ls
ggrant@4b450d793c16:~/sandbox$
```

- We’re not getting into it, but the prompt is highly configurable, so on other systems it will not always look exactly like this.

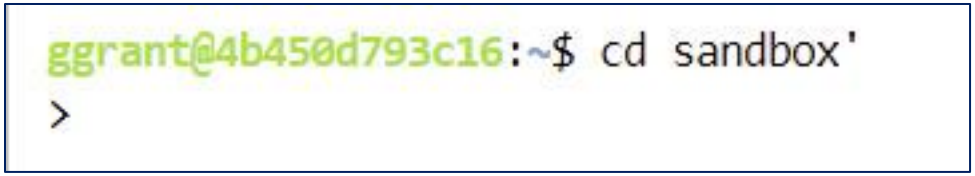
control - c

- The control-c key sequence is used to abort things and it can be your best friend.
  - *On a MAC it may be command-c instead of control-c, for the sake of exposition we will gloss over this distinction.*
  - Hold down the control key down, then hit ‘c’
- It's not uncommon that you try to do something that takes longer than you expected.
  - For example, you might be counting lines in an extremely large file, eventually you decide it's not worth waiting and want to abort the operation.
  - Most of the time ctrl-c will abort the current operation and bring you back to a prompt.

control - c

- If the command-prompt gets into a weird mode, control-c is always a good thing to try. Common typos can cause this, for example enter the following, including the single quote at the end:
  - cd sandbox'

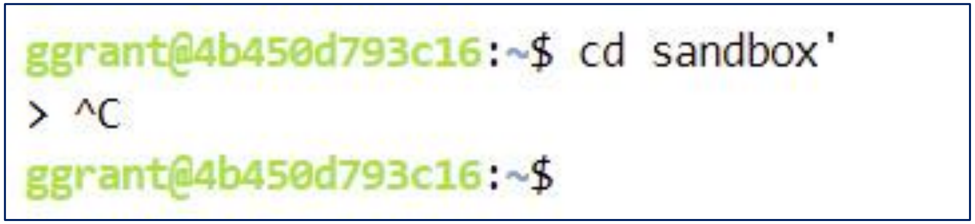
  Notice how it comes back with just a '>’.   It has not executed the command and now there’s a weird prompt.



```
ggrant@4b450d793c16:~$ cd sandbox'
>
```

That's because the single open quote has confused it.

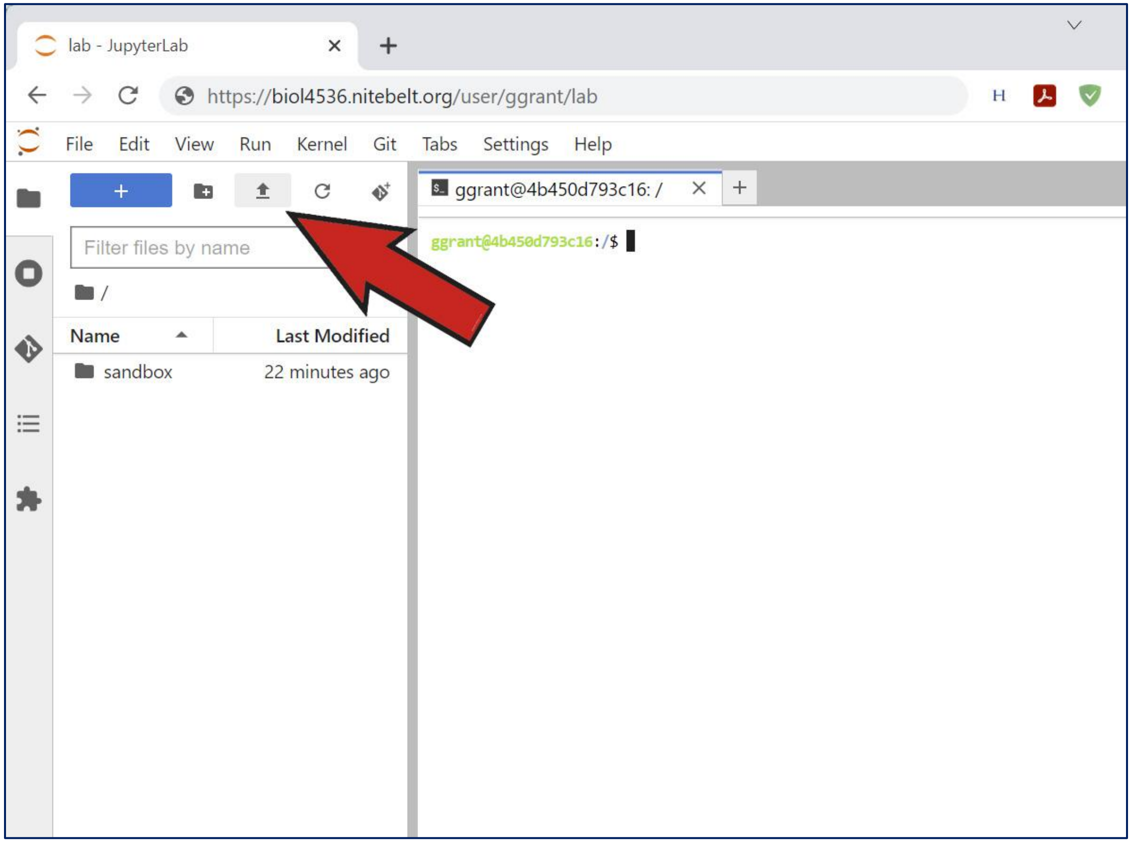
Hit control-c to get out of this and try again.



```
ggrant@4b450d793c16:~$ cd sandbox'
> ^C
ggrant@4b450d793c16:~$
```

Uploading Files

- Jupyter gives you a convenient utility to upload files.
  - It’s not usually that easy if you connect with an ssh client…



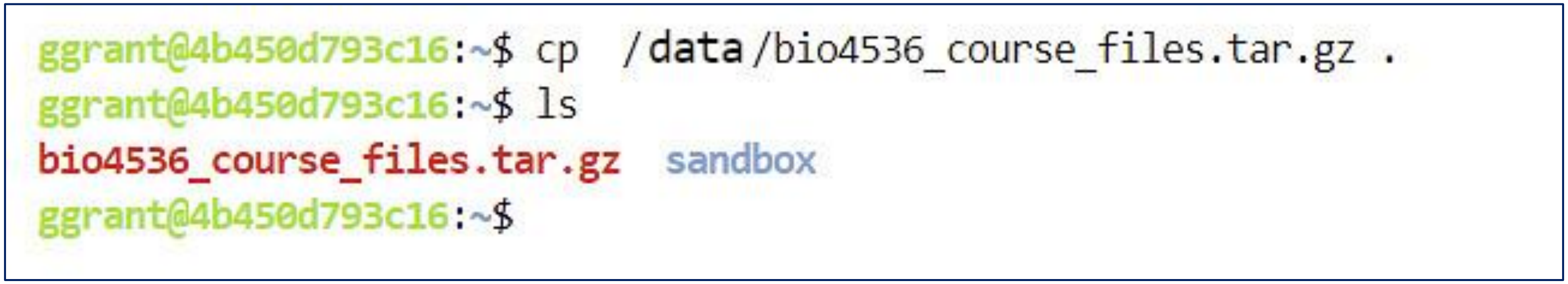
```
ggrant@4b450d793c16:/$
```

Getting Some Files to Play With

- We have already put some data files on the system for you to practice with.
  - They’re in the directory called `/data`
- Enter the following command to copy one of the data files to your home directory.
  - Make sure you are in your home directory when you do this (and don’t miss the period at the end)

```
cp  /data/bio4536_course_files.tar.gz  .
```

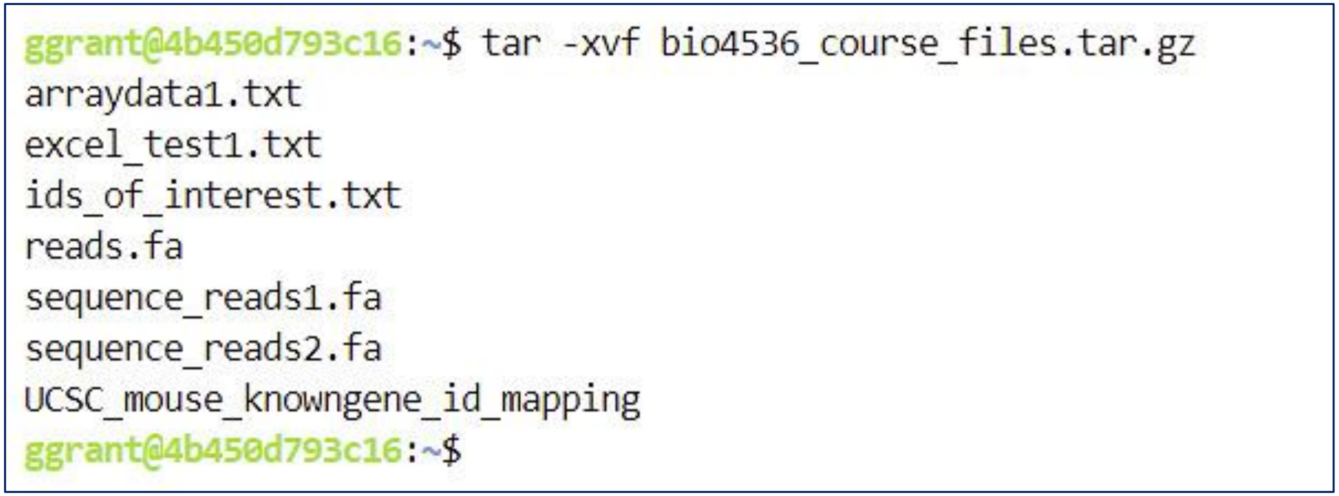
- Once it has finished copying, enter 'ls’ to make sure you see this file listed.



```
ggrant@4b450d793c16:~$ cp  /data/bio4536_course_files.tar.gz .
ggrant@4b450d793c16:~$ ls
bio4536_course_files.tar.gz  sandbox
ggrant@4b450d793c16:~$
```

TAR Archives

- You have saved a so-called ‘tar.gz' file. This is similar to a Windows 'zip' file in that it is a way to pack up several files into one, for easy transport.
- To unpack the file, enter the following:
  - `tar –xzvf python_course_files.tar.gz`



```
ggrant@4b450d793c16:~$ tar -xvf bio4536_course_files.tar.gz
arraydata1.txt
excel_test1.txt
ids_of_interest.txt
reads.fa
sequence_reads1.fa
sequence_reads2.fa
UCSC_mouse_knowngene_id_mapping
ggrant@4b450d793c16:~$
```

Practice Files

- The previous operation unpacked 7 files from the tar.gz archive.
  - It listed the names of the files as it unpacked them.
- We will use these files to practice Unix.
- These are all text files of biological data, or metadata.
  - Metadata is data about data.
- For example, arraydata1.txt is microarray data.
  - Every row corresponds to a spot on the array, and therefore to some gene.
- sequence_reads1.fa is a file of high throughput sequencing data.

Listing Files

- 'ls' is short for 'list' and it simply lists the files in the current directory.
  - Strictly speaking it lists the files *and* directories in the current directory, but we're going to get lazy about being that specific.
  - If you follow 'ls' with the name of a subdirectory of the current directory, it will list the files in that subdirectory.
- By default, 'ls' lists things in alphabetical order, with capitals coming before lower case.
  - But 'ls' is very flexible, we will see shortly how to use its options to return the results in many different ways.

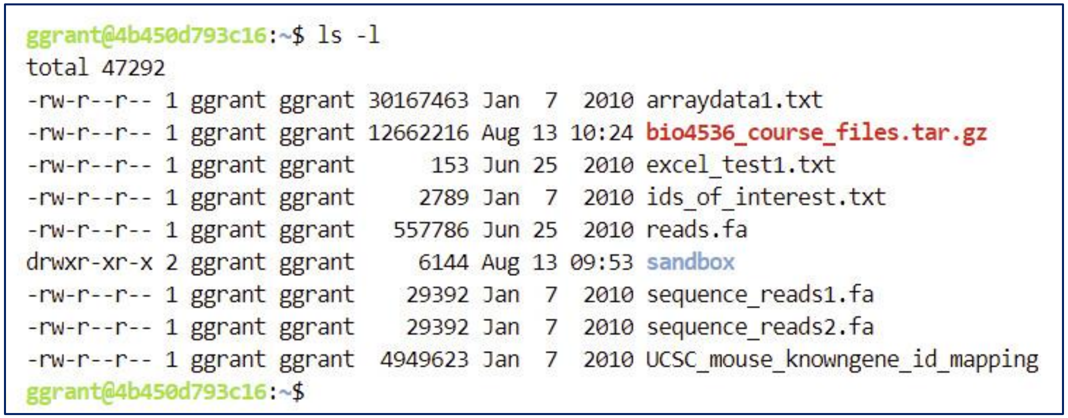
Wildcards and Tab Completion

- The asterisk (*) can be used as a wildcard in 'ls'.
  - From your home directory, try the following:
  - ➢ ls s*
  - ➢ ls *txt
  - ➢ ls a*t

  Etc… you get the picture.
- The **tab** key is your other best friend on the command-line because it completes names for you.
  - Type "ls se" *without hitting enter*, and then hit the tab key.
  - It should finish the string 'se' to be 'sequence_reads'
    - That is where it stops, because there is more than one file which starts 'sequence_reads'.
  - Hit the tab key a couple more times and it will show you all the possibilities.
  - Type '1' (one) and then the tab again and it should complete the string.
- Play around with these two features and make sure you get it, because *bioinformaticians must make a habit of wildcards and tab completion*.

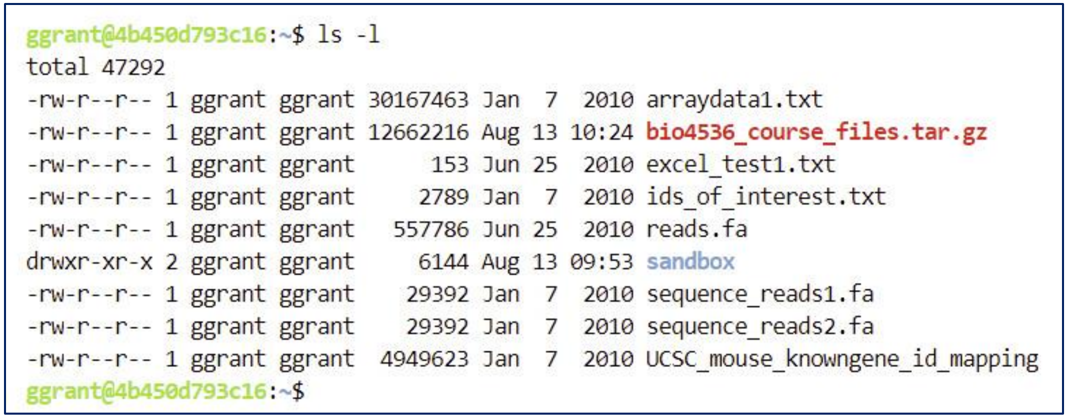
`ls -l`

- The '`ls`' function has many options.
- Enter '`ls -l`' (that's the lowercase letter "L“) and see what happens.
- When you do `ls -l` you get the files listed one per line.
  - You may need to stretch out your window a bit, so that lines don't wrap.
- The 'l' stands for 'long' meaning files are listed in the 'long' format.



```
ggrant@4b450d793c16:~$ ls -l
total 47292
-rw-r--r-- 1 ggrant ggrant 30167463 Jan  7  2010 arraydata1.txt
-rw-r--r-- 1 ggrant ggrant 12662216 Aug 13 10:24 bio4536_course_files.tar.gz
-rw-r--r-- 1 ggrant ggrant      153 Jun 25  2010 excel_test1.txt
-rw-r--r-- 1 ggrant ggrant     2789 Jan  7  2010 ids_of_interest.txt
-rw-r--r-- 1 ggrant ggrant   557786 Jun 25  2010 reads.fa
drwxr-xr-x 2 ggrant ggrant     6144 Aug 13 09:53 sandbox
-rw-r--r-- 1 ggrant ggrant    29392 Jan  7  2010 sequence_reads1.fa
-rw-r--r-- 1 ggrant ggrant    29392 Jan  7  2010 sequence_reads2.fa
-rw-r--r-- 1 ggrant ggrant  4949623 Jan  7  2010 UCSC_mouse_knowngene_id_mapping
ggrant@4b450d793c16:~$
```

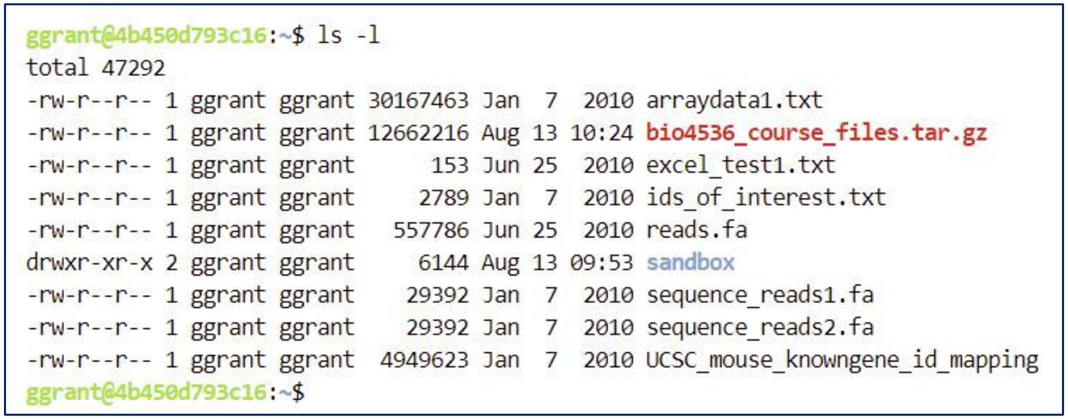
- Long format displays a bunch of information for each file.



```
ggrant@4b450d793c16:~$ ls -l
total 47292
-rw-r--r-- 1 ggrant ggrant 30167463 Jan  7  2010 arraydata1.txt
-rw-r--r-- 1 ggrant ggrant 12662216 Aug 13 10:24 bio4536_course_files.tar.gz
-rw-r--r-- 1 ggrant ggrant      153 Jun 25  2010 excel_test1.txt
-rw-r--r-- 1 ggrant ggrant     2789 Jan  7  2010 ids_of_interest.txt
-rw-r--r-- 1 ggrant ggrant   557786 Jun 25  2010 reads.fa
drwxr-xr-x 2 ggrant ggrant     6144 Aug 13 09:53 sandbox
-rw-r--r-- 1 ggrant ggrant    29392 Jan  7  2010 sequence_reads1.fa
-rw-r--r-- 1 ggrant ggrant    29392 Jan  7  2010 sequence_reads2.fa
-rw-r--r-- 1 ggrant ggrant  4949623 Jan  7  2010 UCSC_mouse_knowngene_id_mapping
ggrant@4b450d793c16:~$
```

- The very first character indicates whether this is a regular file, a directory, or a link.
  - If it's a regular file, then it's a dash "–"
  - If it's a directory, then it's a "d"
  - If it’s a link, then it’s an "l“
    - We won't be concerned with links in this class, but you can imagine what they are.

ls -l



```
ggrant@4b450d793c16:~$ ls -l
total 47292
-rw-r--r-- 1 ggrant ggrant 30167463 Jan  7  2010 arraydata1.txt
-rw-r--r-- 1 ggrant ggrant 12662216 Aug 13 10:24 bio4536_course_files.tar.gz
-rw-r--r-- 1 ggrant ggrant      153 Jun 25  2010 excel_test1.txt
-rw-r--r-- 1 ggrant ggrant     2789 Jan  7  2010 ids_of_interest.txt
-rw-r--r-- 1 ggrant ggrant   557786 Jun 25  2010 reads.fa
drwxr-xr-x 2 ggrant ggrant     6144 Aug 13 09:53 sandbox
-rw-r--r-- 1 ggrant ggrant    29392 Jan  7  2010 sequence_reads1.fa
-rw-r--r-- 1 ggrant ggrant    29392 Jan  7  2010 sequence_reads2.fa
-rw-r--r-- 1 ggrant ggrant  4949623 Jan  7  2010 UCSC_mouse_knowngene_id_mapping
ggrant@4b450d793c16:~$
```

- The rest of the information relates to permissions, ownership, size and date of last modification.
- Permissions get complicated because there are three types of permission:
  - Read, write and execute
- And there are three categories of users:
  - User, group and other.
  - ‘User’ is you. ‘Other’ is everybody. And you can also create ‘groups’.
  - Each of these three categories gets their own permissions to each file.
    - That’s why there are nine of them

Permissions

- We don’t have time to go into permissions in detail.
  - Unix involves a lot of learning on the fly.
  - Nobody remembers everything.
  - Things you do regularly, you remember.
  - Everything else you look up and/or learn on the fly as you need it.
- But just keep in mind that regular Unix users routinely run into bugs that trace back permission settings.

Documentation

- The ls function has many options. One of my favorites is `'ls -ltr'` which lists the files in order of when they were created, so that the most recent is at the end of the list.
  - Notice how the three options 'l', 't', and 'r' are combined into `'-ltr'`.
  - It would be equivalent to type `'ls -l -t -r’`

- We will discuss two ways to find out about the possible options to a command.
- One thing you can always try is to run the command with just the --help option.
  - That’s two dashes.
- For example:

➢ `ls --help`

- Returns documentation. You may need to scroll back if it’s a long page.

- Another way to get documentation is to use the ‘man’ function.
  - See next slide.

man pages

- Pretty much every function has a man page, which is documentation describing its usage and options ("man" is short for "manual")
- For example, enter 'man ls'.
  - You can page through the man page using the space bar, or the up/down arrow keys.
  - Most of this will be unintelligible to you at this point, but some of the options should be clear. For example:
    - -S sort by file size

Navigation

- The basic command for navigation is "cd" which stands for "change directory".
- Entering "cd" all by itself takes you to your home directory.
  - Enter cd now.
- Now enter "cd sandbox" to take you into the sandbox directory.
  - Enter "pwd" to make sure.
- To move **up** one directory in the hierarchy, enter "cd .." (cd followed by a space followed by two dots).
  - *REMEMBER THIS!*
- You can also use cd with a "*full path*".
  - ➢ cd /home/ggrant/sandbox
  - This will always get you to the sandbox no matter where you are.
- Or you can get somewhere *relative* to your home directory by using a tilde.
  - The tilde is shorthand for your home directory.
  - ➢ cd ~/sandbox
  - This takes you to the sandbox no matter where you are

REVIEW: Four Ways to Get from Point A to Point B

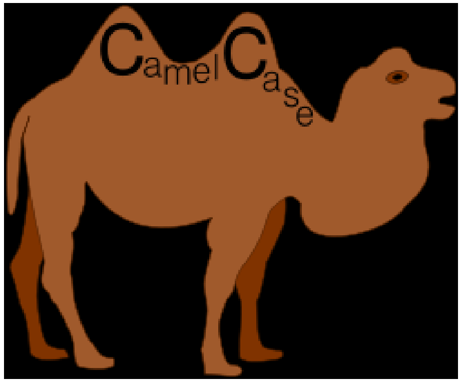
- Move into your home directory (by entering 'cd'). Make a new directory called 'sandbox2'.
- Now 'cd' into original sandbox 'cd sandbox'.
- You can get to the sandbox2 directory in four ways:
  1. Get there in steps:
     - 'cd .. ' followed by 'cd sandbox2'.
  2. Give the *absolute path*:
     - 'cd /home/ggrant/sandbox2'
  3. Give the *relative path* to the current directory:
     - 'cd ../sandbox2'
  4. Give the path relative to your home directory:
     - 'cd ~/sandbox2'

File Naming Conventions

- Windows and Mac are very generous about what they allow in file names.
- But things like spaces play havoc with the command line. You can still use them, but you always work harder to work with files named with spaces.
- The command-line user should avoid using such characters in file names.
  - **Use underscores, periods, and/or dashes instead of spaces.**
- **RULES OF THUMB:**
  - Use only letters, numbers, periods, dashes, underscores in file and directory names.
  - Do not *start* the name of a file with a dash, because otherwise they get confused with function options.
  - If you start a file name with a period, it will become "hidden" and will not show up if you do 'ls'.
    - You must do 'ls –a' ('a' for 'all') to see the hidden files.
  - Typically, you should start all file names with a letter, a number or an underscore.

File Naming Conventions

- There several conventions for avoiding spaces, what you use is a matter of taste:
  - Use underscores, periods or dashes:
    - my_file_of_data.txt
    - my.file.of.data.txt
    - my-file-of-data.txt
  - Use Capitalization:
    - myFileOfData.txt or MyFileOfData.txt
    - This method is called "camel case" because of the way the caps are like a camel's humps
  - Use any combination of the above, there is no right or wrong, just try to make it readable.



If you must…

- If you must put spaces or other weird characters in file and/or directory names, then enclose the entire name in quotes.

  -
    ```
    mkdir "my project (draft 1)"
    ```
- Or alternatively you can put backslashes before weird characters.

  -
    ```
    mkdir my\ project\ \(draft\  1\)
    ```
- Some characters are just downright forbidden in file names however, such as forward and backward slashes "\" and "/“.
  - Forward slashes separate file names in a path, so if file names themselves have forward slashes, that’d be a mess.
- Sometimes you have no choice but to deal with unconventional characters because somebody else gives you a file already named that way.
  - The first thing I always do with such files is change the name.

Copying and Moving Files

- with cp and mv -

- Two more basic file manipulation functions are 'cp' and 'mv'.

➢ cp file1 file2

- – makes a copy of file1 and calls the copy file2
- – To copy a directory use 'cp –r' (‘r' for 'recursive).

➢ mv file1 file2

- – renames file1 to be file2
- – this also works on directories

Deleting Files

- Use the 'rm' function to delete (remove) a file.
  - Be careful, there is no recycle bin, once you 'rm' a file it's gone.
  - Try it out by removing the .tar file.
- You can also use the asterisk wildcard with 'rm' but *be careful, accidentally entering 'rm *' will delete every file in the current directory*.
  - *There is no undo or trash bin.*

Deleting Directories

- You cannot use 'rm’ for a directory. For that there is a separate function 'rmdir'.
  - But if the directory is not empty the system won't let you remove it that easily. And it may look empty but still have hidden files (as discussed earlier, you can do 'ls −a' on the directory to see if there are any hidden files).
  - To force the issue on a non-empty directory, do 'rm −r’.
    - r stands for "recursive" and it means it will remove the directory and all of its contents.
    - But it's dangerous, always be careful removing files.

Aliases

- I find myself doing the following very often:

➢ `ls –ltr`

- It lists files in long format, sorted by time of last modification, most recent last.
- Since I do it so often, I created an alias

➢ `alias lsr=“ls –ltr”`

- Now I just need to type `lsr`

Persistent Aliases

- If you make an alias, it will disappear once you end the session.
- If you want it to persist, you add it to your `.profile` file
  - On some systems it may be called `.bash_profile` or `.bash_login`
- This is a hidden file (it starts with a dot)
- The commands in this file gets executed automatically every time you log in.
- So, if you put the alias command there, it will always be available.

Review

- Here are the commands we've seen so far, make sure you remember them before moving on.
  - We will be revisiting some of these functions and their options in more detail later.

1. **pwd**
   Print the current working directory
2. **cd**
   Change the current working directory
3. **ls**
   List file and directory names
4. **cp**
   Copy a file
5. **mv**
   Rename a file
6. **rm**
   Remove file(s)
7. **mkdir**
   Make a directory
8. **rmdir**
   Remove directorie(s)
9. **tar –xvf**
   Unpack a tar file
10. **man**
    Display the manual page for a function
11. **cal**
    Returns the calendar of any month/year up to the year 9999
12. **date**
    Returns the current date and exact time.
13. **echo**
    Echoes text and variable values to the screen
14. **Aliases**
    Create shorthand for common commands and save them in `.bash_profile` to be persistent.

Binary versus Text

- There are two types of files on a computer, text files and binary files.
  - Most files are binary these days.
    - Images, videos, music, power point slides, excel spreadsheets, etc..
  - Some common text files include:
    - Web HTML source files.
    - Files of source code in some programming language.
    - Files of raw data.
  - Some files that display as text are nonetheless still saved as binary files on your disk, such as Microsoft Word documents
- In biology we deal a lot with text files.
  - Files of sequence or of numerical measurements.
- The command line allows you to view text files gracefully, while to view binary files you need special programs.
  - You can manipulate binary files on the command line (e.g. change their names, delete them, move them, etc.), but you cannot easily view them from there.
- ***The functions on the following slides show how to view text files and should usually be applied to text files only.***

Inspecting Files

- using 'head' and 'tail' -

- The 'head' function shows you the first ten lines of a *text* file. (*Note: See the earlier slide titled "command not found" if this or any other function returns a "command not found" error*.)

```
➢ head ids_of_interest.txt
NM_022023
NM_144958
NM_028889
NM_172938
NM_172405
NM_175025
NM_029777
NM_027154
NM_207141
NM_013830
```

- If you do 'head –n 20 ids_of_interest.txt' it will show the first 20 lines.
  - Or replace 20 with any number.
  - The shorthand "head –20' is equivalent to 'head –n 20
- The 'tail' function works the same way, except that it shows the last ten lines of the file. Tail also takes the numeric option.

Inspecting Files

- using ‘less' -

- The ‘less' function lets you see the entire file, one page at a time.
  - Enter ‘less ids_of_interest.txt' and see what happens.
  - Use the space bar to scroll to each next page.
  - To quit without paging all the way to the end, hit the 'q' key.
- The ' less’ command allows some more sophisticated options.
  - A very useful option is –S which enables both vertical *and* horizontal scrolling.
  - The less function also allows for searching for keywords in the document.
  - See the ' less' man page for a full description of all options.

Redirection

- "Redirection" simply means sending the output of a command to a file instead of to the screen.
- The redirection operator is the ">" symbol.
- Try the following:
  - ➢ head −15 ids_of_interest.txt > first_15_ids.txt
  - – Now do 'ls –l' and you'll see a new file called 'first_15_ids.txt'.
  - – Do a 'more' on this file and see what's in there. You should see the first 15 lines from the file ids_of_interest.txt
- In summary, you have 'redirected' the output of 'head −15 ids_of_interest.txt' to a new file.

Redirection Continued…

- Be careful with redirection because you can easily overwrite files by accident. If you do

```
head -15 ids_of_interest.txt > myfile.txt
```

  and myfile.txt already exists, it will be destroyed and overwritten by the new file.

- Instead, if you use two ">" symbols in a row, it adds to the current file (if there is one) without destroying it

```
head -15 ids_of_interest.txt > myfile.txt
tail -15 ids_of_interest.txt >> myfile.txt
```

- This will put the last 15 lines after the first, so myfile.txt after this should have 30 lines. Do 'more' on myfile.txt to make sure.

The cat function

- The 'cat' command is one of the most used of all commands.
  - "cat" is short for "concatenate"
- You can use 'cat' to view an entire file.
  - It's like 'more' but it doesn't give you one page at a time, it gives you everything at once.
  - Try 'cat ids_of_interest.txt' and see what happens.
- You can use cat together with the redirection operator as a most basic text editor. Enter the following:
  - ➢ cat > test1.txt
  - Now type some text, anything, it can include newlines. When done hit enter so the cursor is on a new line, and then hit control-d.
  - Do 'ls' and you should see a new file, a file called 'test1.txt'.
  - Do 'cat test1.txt' and it should show you the text you entered.
  - To *add* text to the file do "cat >> test1.txt".
    - Without the second '>', "cat > test1.txt " will destroy the original file and writes the new one in its place.
  - This is a way to very quickly jot or paste something into a file, but any serious text editing should be done elsewhere, like in Pico.

Using 'cat' to concatenate things

- You can use concatenate to glue two or more files together. Try the following:

  ```
  cat first_15_ids.txt test1.txt
  ```

- It should return to the screen the contents of both files one after the other. Now do:

  ```
  cat first_15_ids.txt test1.txt > test2.txt
  ```

  - There should now be a file `test2.txt` that contains the concatenation of the two files. Use '`ls`' and '`cat`' to make sure.
- You can't concatenate files in Windows Explorer or Finder, you would have to use a text editor to merge them. But if your files are enormous, as they often are in biology, your text editor might choke.
  - A text editor wants to read an entire file into RAM memory all at once which will fail if the file is larger than the available RAM.
  - cat, on the other hand, doesn't need to read the files in all at once, in order to write their concatenation to a new file.
- This is the only reasonable way to merge files that are gigabytes in size.
  - Next generation sequencing produces gigabye sized files, and growing.

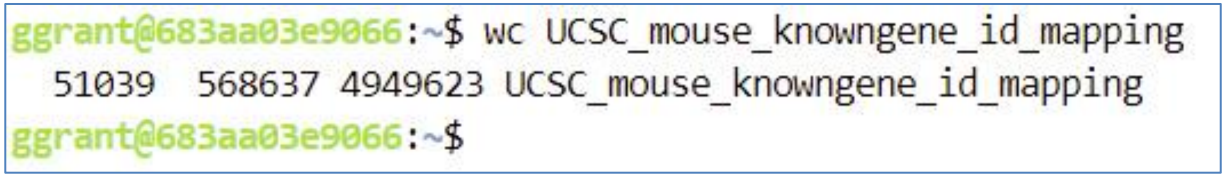
Inspecting Files

- counting lines, words, characters -

- The 'wc' command (for 'word count') gives the number of lines, words, and characters in a file. Try the following:

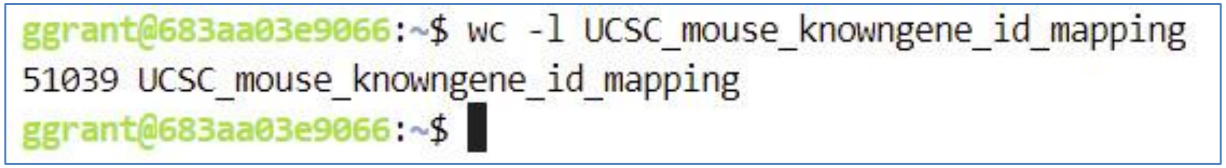
➢ wc UCSC_mouse_knowngene_id_mapping

- This will return three numbers
  - The first is the number of lines, the second is the number of words, the third is the number of characters.



```
ggrant@683aa03e9066:~$ wc UCSC_mouse_knowngene_id_mapping
  51039  568637 4949623 UCSC_mouse_knowngene_id_mapping
ggrant@683aa03e9066:~$
```

  - To just get the number of lines, use 'wc -l'



```
ggrant@683aa03e9066:~$ wc -l UCSC_mouse_knowngene_id_mapping
51039 UCSC_mouse_knowngene_id_mapping
ggrant@683aa03e9066:~$ █
```

Comparing Files

- The diff command is used to compare two text files. Create two files as follows:

```
cat > test1.txt
```

```
1 2 3
4 5 6
a b c
```

```
cat > test2.txt
```

```
1 2 3
x y z
a b c
```

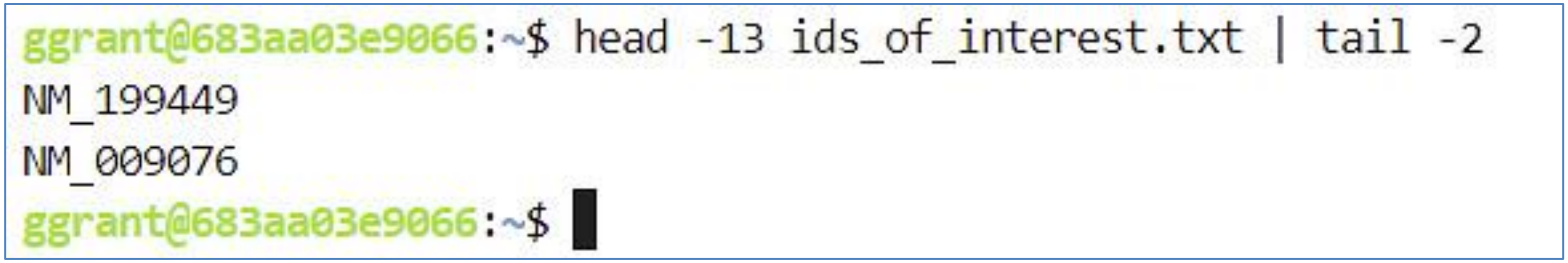
- Now run diff:

```
diff test1.txt test2.txt
```

```
2c2
< 4 5 6
---
> x y z
```

The pipe operator "|"

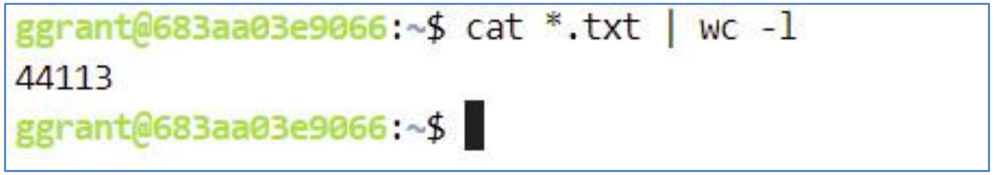
- The vertical bar "|" is called the *pipe operator*.
- You use the pipe to direct the output of one command to be the input of another.
  - In other words, to daisy-chain commands together.
- Suppose, for example, we wanted the 12th and 13th lines only of the file ids_of_interest.txt. We can do that using **head** and **tail** as follows:
  - ➢ head −13 ids_of_interest.txt | tail −2



```
ggrant@683aa03e9066:~$ head -13 ids_of_interest.txt | tail -2
NM_199449
NM_009076
ggrant@683aa03e9066:~$ █
```

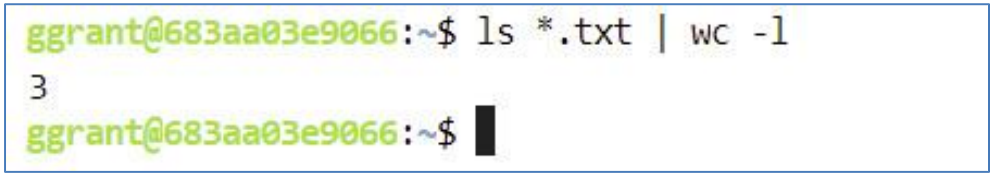
The pipe operator "|"

- As another example, the following counts the number of lines of all files that end in .txt combined.
  - ➢ cat *.txt | wc –l



```
ggrant@683aa03e9066:~$ cat *.txt | wc -l
44113
ggrant@683aa03e9066:~$ ▌
```

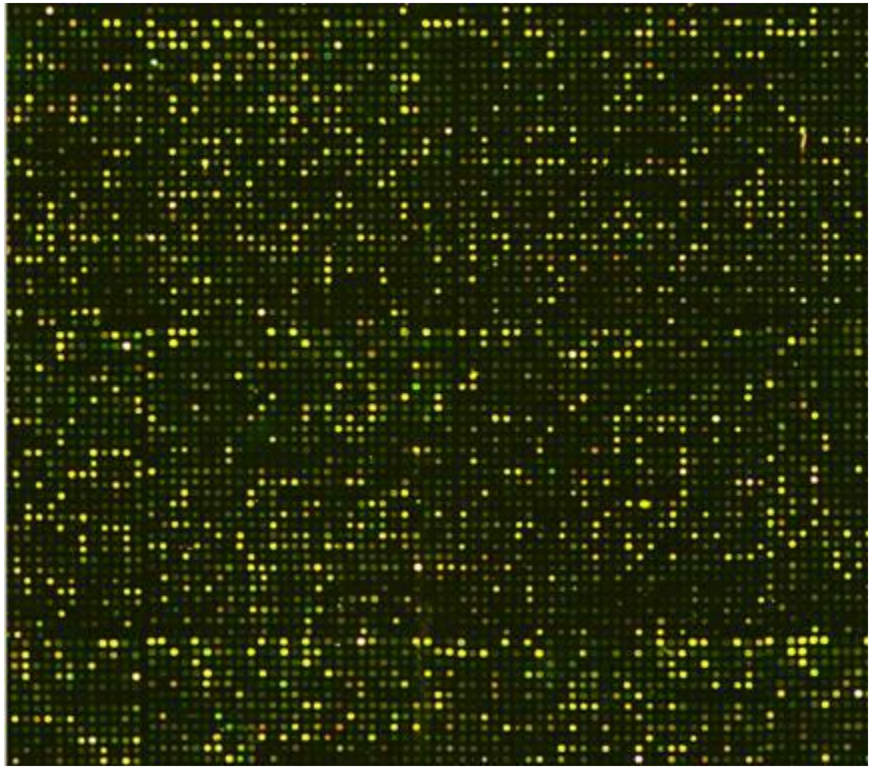
- And the following simply counts how many files are in the directory that end in .txt.
  - ➢ ls *.txt | wc –l
  - – There should be three:
    - arraydata1.txt, first_15_ids.txt, and ids_of_interest.txt



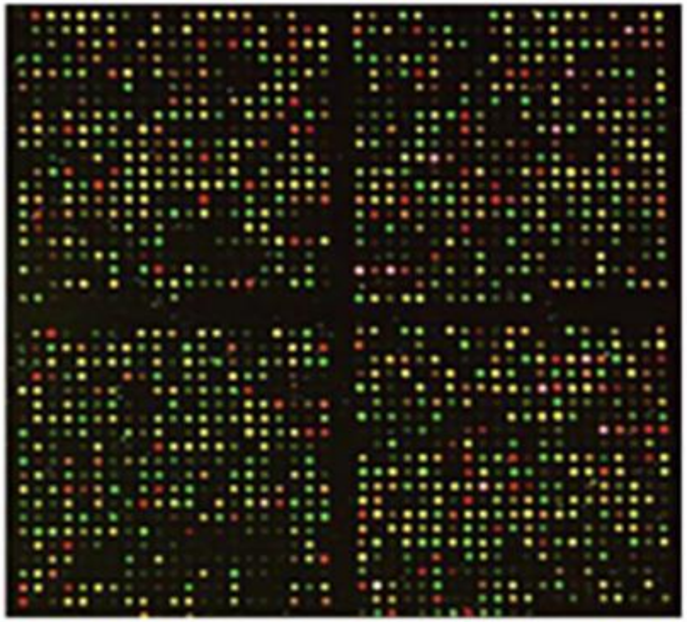
```
ggrant@683aa03e9066:~$ ls *.txt | wc -l
3
ggrant@683aa03e9066:~$ ▌
```

High Throughput
- Gene Expression -

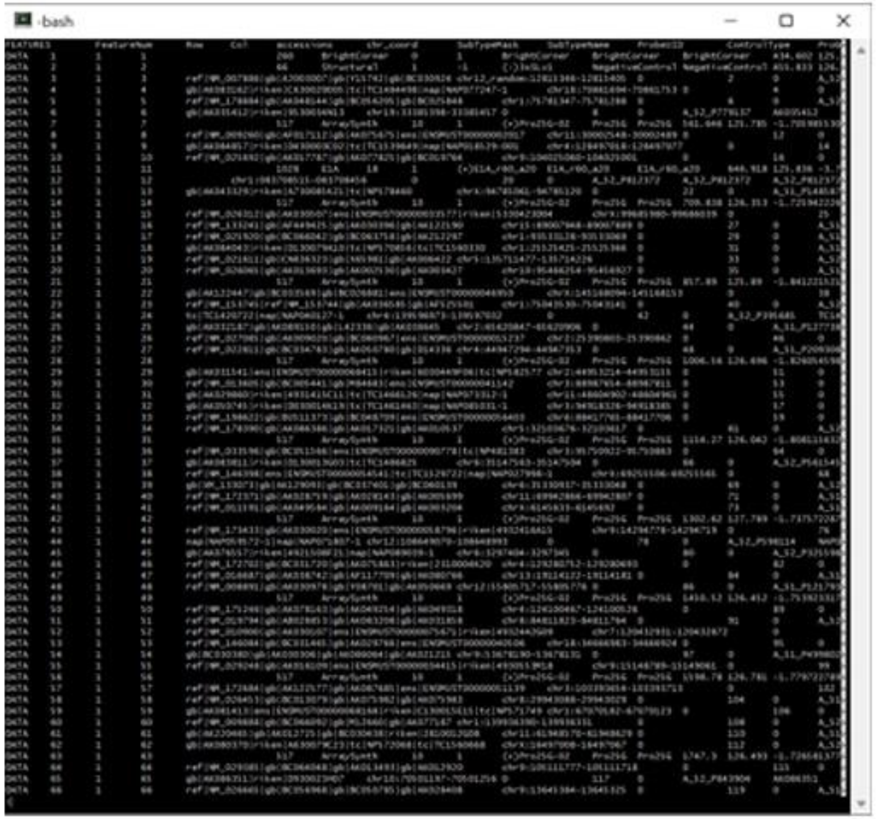
- **MICROARRAYS**
- Measure the expression level of tens of thousands of genes simultaneously in a sample.
- Each spot corresponds to a gene.
- The brighter the spot, the higher the gene is expressed.



Microarray Quantification



Microarray Image

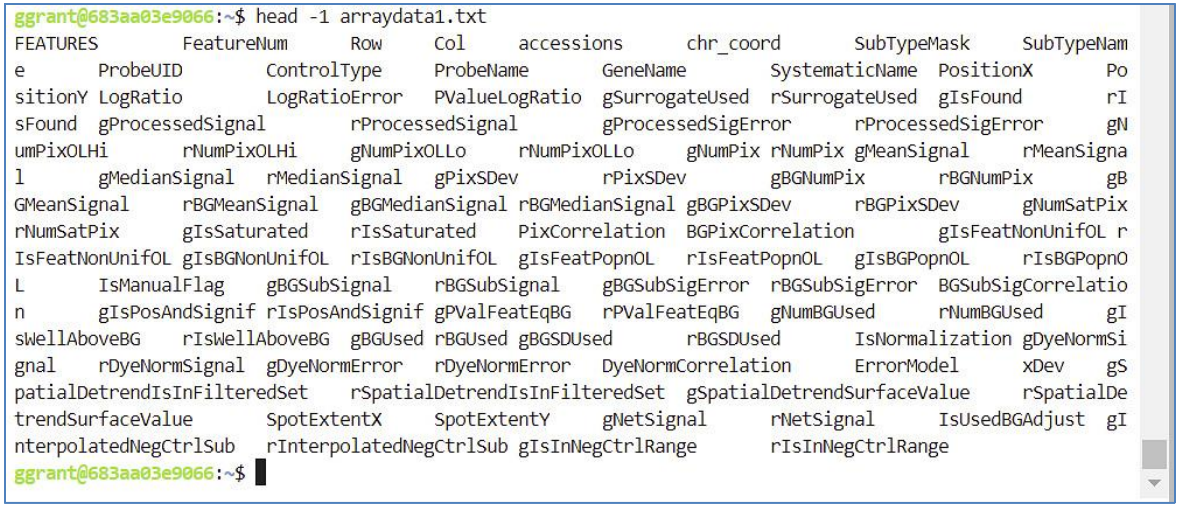


Tab Delimited Text File
(one row per spot)

- For each spot, many pieces of information are recorded in the different columns.
  - In our case over 100 metrics and flags are output.

cut

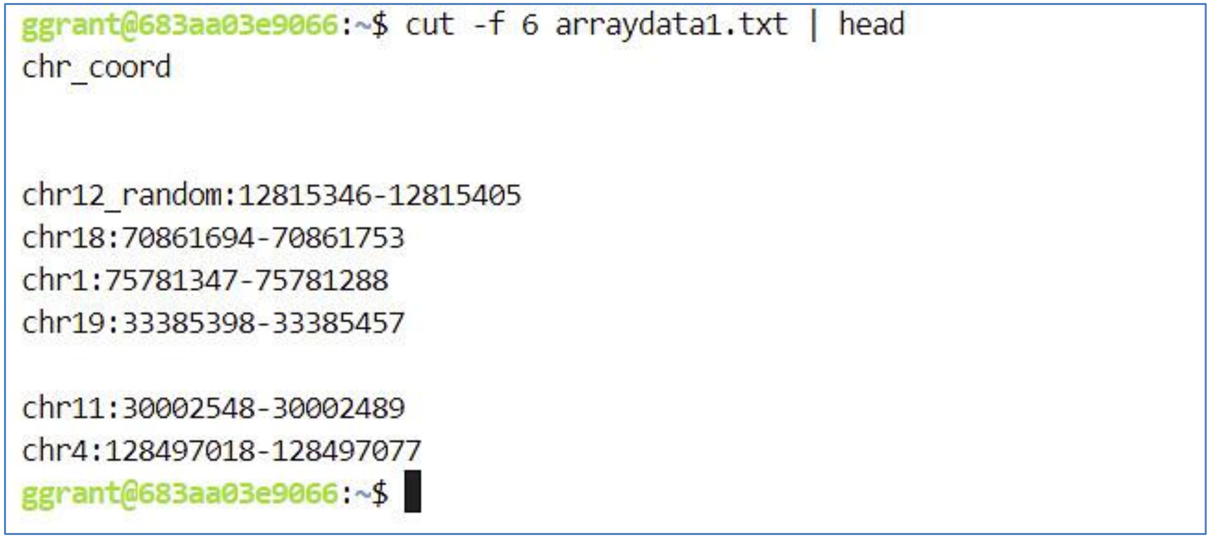
- Very often in biology we deal with tab delimited spreadsheets of data.
- The file arraydata1.txt in your sandbox is such a file.
  - It is a file of information about a microarray experiment.
- We have seen how to grab lines of a file using head and tail. In contrast, 'cut' allows you to grab columns.
- Enter '`head -1 arraydata1.txt`' to see the first (header) line.
  - This line shows the names of all the **columns**, there are 100 columns.
- Every **row** of this file (after the header row) represents a probe for a different gene.



```
ggrant@683aa03e9066:~$ head -1 arraydata1.txt
FEATURES        FeatureNum      Row     Col     accessions      chr_coord       SubTypeMask     SubTypeNam
e       ProbeUID        ControlType     ProbeName       GeneName        SystematicName  PositionX       Po
sitionY LogRatio        LogRatioError   PValueLogRatio  gSurrogateUsed  rSurrogateUsed  gIsFound        rI
sFound  gProcessedSignal        rProcessedSignal        gProcessedSigError      rProcessedSigError      gN
umPixOLHi       rNumPixOLHi     gNumPixOLLo     rNumPixOLLo     gNumPix rNumPix gMeanSignal     rMeanSigna
l       gMedianSignal   rMedianSignal   gPixSDev        rPixSDev        gBGNumPix       rBGNumPix       gB
GMeanSignal     rBGMeanSignal   gBGMedianSignal rBGMedianSignal gBGPixSDev      rBGPixSDev      gNumSatPix
rNumSatPix      gIsSaturated    rIsSaturated    PixCorrelation  BGPixCorrelation        gIsFeatNonUnifOL r
IsFeatNonUnifOL gIsBGNonUnifOL  rIsBGNonUnifOL  gIsFeatPopnOL   rIsFeatPopnOL   gIsBGPopnOL     rIsBGPopnO
L       IsManualFlag    gBGSubSignal    rBGSubSignal    gBGSubSigError  rBGSubSigError  BGSubSigCorrelatio
n       gIsPosAndSignif rIsPosAndSignif gPValFeatEqBG   rPValFeatEqBG   gNumBGUsed      rNumBGUsed      gI
sWellAboveBG    rIsWellAboveBG  gBGUsed rBGUsed gBGSDUsed       rBGSDUsed       IsNormalization gDyeNormSi
gnal    rDyeNormSignal  gDyeNormError   rDyeNormError   DyeNormCorrelation      ErrorModel      xDev    gS
patialDetrendIsInFilteredSet    rSpatialDetrendIsInFilteredSet  gSpatialDetrendSurfaceValue     rSpatialDe
trendSurfaceValue       SpotExtentX     SpotExtentY     gNetSignal      rNetSignal      IsUsedBGAdjust  gI
nterpolatedNegCtrlSub   rInterpolatedNegCtrlSub gIsInNegCtrlRange       rIsInNegCtrlRange
ggrant@683aa03e9066:~$
```

cut

- The sixth column gives the chromosomal location of the probe. To get just this column enter the following:
  - ➢ cut –f 6 arraydata1.txt | head
- We piped this to 'head' because otherwise it would dump thousands of lines out to the window.
  - – Three of the entries are blank, since location is unknown or missing those probes
- Make a habit of using 'head' like this.



```
ggrant@683aa03e9066:~$ cut -f 6 arraydata1.txt | head
chr_coord


chr12_random:12815346-12815405
chr18:70861694-70861753
chr1:75781347-75781288
chr19:33385398-33385457

chr11:30002548-30002489
chr4:128497018-128497077
ggrant@683aa03e9066:~$ 
```

cut

- continued -

- Column 12 gives the name of the gene and column 33 gives an expression intensity level for the gene.
- To cut out just those two columns:

```
➢cut –f 12,33 arraydata1.txt > id2intensity.txt
```

- This redirects the output to a new file called id2intensity.txt.
- Take a look at this file to make sure it looks proper.
  – Use 'head' or ‘less' to look at the file, whatever you prefer.
- You can also specify a range of columns as follows:

```
➢ cut –f 12-20 arraydata1.txt
```

- This will grab columns 12 through 20.

paste

- 'Paste' is basically the opposite of 'cut'. It allows you to glue files together horizontally.
- Don't confuse the 'cut' and 'paste' commands with cutting and pasting using the mouse, they are not the same thing.
- Enter the following:

```
cut –f 10,11 arraydata1.txt | head > temp1
cut –f 33,12 arraydata1.txt | head > temp2
```

- Cat both files temp1 and temp2 to see what's in them, they should both have ten lines, each with two columns.
- Now enter the following:

```
paste temp1 temp2
```

- It should print out ten lines each with four columns. The two files have been pasted together side-by-side.
  - You can do this in Excel, but not easily if the file has millions of rows.

sort

- Sorting is basic and the Unix sort function is powerful, fast and easy.
- Try the following:

  ➢ `cut -f 12 arraydata1.txt | head -20`

- Now try:

  ➢ `cut -f 12 arraydata1.txt | head -20 | sort`

- The output is now sorted, in lexical order. Try the following instead:

  ➢ `cut -f 12 arraydata1.txt | sort | head -20`

  – This gave a different answer, can you figure out why?

- Sort writes its output to the screen. If you want to save it, redirect it to a file.
- Two useful options to sort are:
  1. `-n` which makes it sort numerically, if the data are numerical.
  2. `-u` means any duplicate lines in the file will only be written once ('u' for 'unique'). Try the following:

  ➢ `cut -f 12 arraydata1.txt | sort | wc -l`

  ➢ `cut -f 12 arraydata1.txt | sort -u | wc -l`

  – The second command returns fewer lines, because entries in column 12 are not unique.

Review

1. **cat**

   Prints files to the screen, and concatenates files vertically.

2. **more** or **less**

   Displays a *text* file one page at a time, use spacebar to scroll.

3. **head**

   Prints the first ten lines of a file

   Use –n option to print *n* lines instead of ten

4. **tail**

   Same as head, but replace 'first' with 'last'

5. **cut**

   Gets columns from tab-delimited spreadsheets

6. **paste**

   Concatenates files together horizontally

7. **wc**

   Word count, displays number of lines, words and characters in a file

8. **sort**

   Sorts files

9. **The redirection operator ">" and ">>"**

   Sends the output of a command to a file

10. **The pipe operator "|"**

    Sends the output of one command to be the input of another

11. **And don't forget about *wildcards* and *tab completion*.**

Practice Exercises 1

1. Construct a way to return the *n*-th line of a file using head, tail and the pipe operator, for any *n*.
2. Count the total number of lines of all (text) files in the sandbox directory combined, using one command with the appropriate wildcard.
   - They should all be text files if you removed the .tar file
3. Put lines 1000 through 2000 of the file UCSC_mouse_knowngene_id_mapping in a file called temp.txt
4. Make a file temp2.txt the first ten lines of which are lines 1 through 10 of temp.txt and the next ten are the last ten lines of temp.txt.
   - Use wc to make sure temp.txt and temp2.txt have 1001 and 20 lines respectively.
5. Use 'head' and redirection (twice) to make a file called temp3.txt that has in it the first ten lines of both files in the sandbox that ends in .fa
   - So temp3.txt should have 20 lines, use wc to make sure.
6. The file UCSC_mouse_knowngene_id_mapping is tab delimited. Make a new file called "genesymbol2uniprot.txt" that just has the two columns 'Gene_Symbol' and 'UniProt'.
   - To figure out which are the right columns, look at the header row of the file.

Control Key Shortcuts

- If you hit the up-arrow key a few times, it will scroll back through your recent commands.
  - Even if you close the window and start over later, the old commands should will still be there.
- Scroll back to any previous command. Notice that the cursor is at the end of the line.
- Hit control-a.
  - The cursor should move to the beginning of the line.
- Now hit control-e.
  - That should take you back to the end of the line.
- Control keys allow you to type less and to navigate without moving your hands from the comfortable typing position.
- Unfortunately, the browser hijacks some of these so they’re not all available to our web-based interface.
- The next page gives the expanded list of control keys.
  - *You don’t need to memorize them*, just read through it to know what you can do, then use it as a reference.

Summary of working control key shortcuts

Control key	Action
f	move cursor forward one position ('f' for 'forwards')
a	move cursor to the beginning of the line
e	move cursor to the end of the line ('e' for 'end')
c	abandon this line, start a new one
k	clear from the cursor to the end of the line – text is copied to the clipboard
u	clear from the beginning of the line to the cursor – text is copied to the clipboard
y	paste the text on the clipboard (if any)
d	delete forward ('d' for 'delete')
h	delete backwards
l	clear the screen
r	search backwards through previous commands, after hitting control-r type the string you are searching for

Practice Exercises 2

Move into your home directory

1. Put lines 101 through 200 of the file sequence_reads1.fa in a file called temp.txt. What is the size, in bytes, of the file temp.txt.
2. In the file UCSC_knowngene_id_mapping, what is the value of the mRNA field in the 12,345th row of *data*.
3. The file arraydata1.txt is a tab delimited spreadsheet. The header line gives the meaning of each column. Each such meaning is described by one word. Use head and wc to determine the total number of columns.
4. Consider the UCSC_knowngene_id_mapping file:
   a) How many rows are in the file?
   b) How many *unique* (different) id's are there in the second column (the Known_Gene_ID_UCSC field)?
5. What does the –n option to the mv function do?

History and control-r

- Enter 'history' at the prompt and see what happens. It should list the entire history of commands you have entered.
  - Before doing this, you might want to enter 'clear' to clear the screen.
- To reissue a command, enter '!' followed by the number of the command on the history list.
  - '!75' will re-execute the 75th command on the history list.
  - To edit command 75 enter '!75:p' and then hit the up arrow (or control-p).
- To *search* through the history of commands, hit control-r and then start typing a search string.
  - The commands matching the search string should display.
  - After entering a search string, hitting control-r again scrolls back through the previous commands that match the search string.

History Config

- Add the two commands to your `.bash_profile` file.

```
export HISTSIZE=5000
export HISTCONTROL=ignoreboth
```

  - If you forgot what `.bash_profile` is, review the slides on “aliases”.
- The first will make it save the last 5,000 commands.
- If you execute a command twice *in a row*, the second thing will make sure it’s only put once in your history file.
- Just FYI, your history is saved in `.bash_history`

tar

- We have seen 'tar' already, when we unpacked that tar archive of data files.
- The tar function can also be used to create archives. Let's make an archive called txtfiles.tar of all files that end in .txt. We do this with a wildcard:

  ```
  tar  -cvf  txtfiles.tar  *.txt
  ```

- You can use wildcards like we did here, or you can just list the names of the files to archive, separated by spaces. If any directories are included in the list, it will tar up the entire directory with all of its contents.
- Do `ls -l` and see how big the file is.

gzip

- The 'gzip' utility is used to compress a file. Enter the following:
    - ➢ gzip txtfiles.tar
- That will replace the file txtfiles.tar with a smaller file txtfiles.tar.gz.
    - – Do `ls -l` again to see how much it compressed.
- For transporting large ***text*** files you want to compress them.
    - – Don't bother using gzip on binary files like images, music or videos, those are already compressed about as much as they can be. Use gzip on large ***text*** files.
- To uncompress simply use 'gunzip'.
- It is possible to untar and unzip simultaneously using 'tar -xvzf'.

Text Editors

- nano -

- Working bioinformaticians need to learn to use a high-powered text editor like Emacs.
- But, to get started, perhaps most intuitive alternative is Nano. From your home directory, enter:

➢ `nano ids_of_interest.txt`

- The file should open in the window in a rudimentary text editor.
- The commands are at the bottom and the '^' symbol means hold the control key down. You can find more commands by hitting `control-g`.
  - Some are not available because the browser hijacks them,for example you can’t use ctrl-w, ctrl-t or ctrl-n
  - ctrl-w will close the browser tab, so be careful!
- You should be able to feel your way around this program and you can use it for basic text editing.

grep

- A "*string*" is a term used to refer to any list of characters.
  - For example, "abc123".
- The 'grep' function is used to search for strings in text files.
  - ***grep is perhaps the single most useful command you can learn.***
- Enter the following:

```
➢ grep ACGTA sequence_reads1.fa
```

- It should return ten lines (if they wrap, stretch out your window to be wide enough, and do it again).
  - To highlight the match itself, add use the --color option (note this option requires *two* dashes).

```
➢ grep --color ACGTA sequence_reads1.fa
```

- These are the 10 lines in the file that have the string 'ACGTA' somewhere in the line.
- The following returns the number of lines that have four A's in a row:

```
➢ grep AAAA sequence_reads1.fa | wc -l
```

There should be 77 such lines, try it.

grep continued…

wildcard .

- grep understands wildcards.
  - The two simplest wildcards are dot '.' and star '*'.
- Dot will match any (single) character. The following will return any line that has five G's followed by *any two characters* followed by five more G's. There should be five such lines.
  - ➢ grep GGGGG..GGGGG sequence_reads1.fa
  - To actually match the period itself, precede it with a backslash and put use quotes.

grep continued…

wildcard *

- Star '*' must follow another character and it matches any number of that character in a row. The following returns all lines that have five G's, followed by any number (including possibly zero) of A's, followed by five more G's.
  - ➢ grep GGGGA*GGGGG sequence_reads1.fa
  - – How would you modify this if you wanted to assure there was at least one A between the two strings of five G's? There is exactly one such line, see if you can use the * to return just that line.
- grep understands a very general wildcard syntax called 'regular expressions' that we will learn more of as we go.
  - – Just as a matter of trivia, grep stands for ‘global regular expression print’ but you don’t have to remember that.

grep

- anchors ^ and $ -

- Sometimes you want to be sure the text you are searching for matches at the beginning or the end of a line only.
- For this use the ^ and $ symbols.
- For example, to get all lines that *start* with three C's in a row, do the following:

```
grep ^CCC sequence_reads1.fa
```

- To get the lines that *end* in three C's, do the following:

```
grep CCC$ sequence_reads1.fa
```

- To find the lines that both start in a C and end in a C, pipe two grep's together:

```
grep ^C sequence_reads1.fa | grep C$
```

- The "^" and "$" characters are called 'anchors' because they anchor the term to a specific location (the start or the end) of the string.

grep

- daisy chaining -

- It's possible to chain more than one grep together using the pipe operator to achieve more complex searches.

- There is exactly one line in sequence_reads1.fa that has four A's in a row, four C's in a row, four G's in a row *and* four T's in a row. The following will find it.

```
grep AAAA sequence1.fa | grep CCCC | grep GGGG | grep TTTT
```

grep

- excluding matches -

- The option –v tells grep to return everything that *does not* match.
- For example, the file sequence_reads1.txt has alternating lines of sequence names and sequence. Head the file to see the first few lines.
  - Sequence name lines start with ">"
    - This is fastA format.
  - Let's return all lines that do not contain the ">".

➢ `grep –v ‘>’ sequence_reads1.fa | more`

- Notice we put quotes around the search string '>' in the last example.
  - If we don't put the quotes, the system will confuse it with the redirection operator.
  - If you use characters that can confuse the system, the string must be enclosed in quotes.

grep continued…

-A and -B

- –A (for 'after') and –B （for 'before') are two very useful options to grep.
  - Use '–A n' to show for each line that matches the search string, the *n subsequent* lines as well.
  - Similarly, '–B n' will show the *n* lines *before* each match.
- The following returns two lines:

  ```
  grep ^CCCC sequence_reads1.fa
  ```

- Now try the following:

  ```
  grep -B 1 ^CCCC sequence_reads1.fa
  ```

  - Each line should be returned with the one line preceding it.
  - Different matches are separated by a line with two dashes '−−'.
  - Think about how you might get rid of those '--' lines by piping through another grep. Here's how to do it:

    ```
    grep -B 1 ^CCCC sequence_reads1.fa | grep  -v  -
    ```

grep continued…

- more useful options -

- The `-n` option returns the line number of each match.
- The `—i` option *ignores* case
- The `-c` option returns just the count of the number of matches, and not the matches themselves.
  - Using `-c` is equivalent to piping the output of grep to `wc -l`.
- The -w option forces grep to match only complete words. Compare the following:

➢ `grep 10 sequence_reads1.fa`

This returns 13 lines, any line that has '10' in it.

➢ `grep -w 10 sequence_reads1.fa`

This returns just one line, the line with '10' not appearing as part of a bigger number.

grep continued…

- groups -

- Suppose you want to match a character that's either an A or a G?
- The following will achieve that.
  - `grep [AG] file.txt`
- You do not need a comma, the following will match an A or a G or literally a comma.
  - `grep [A,G] file.txt`
- You can also use ranges, the following will match any capital letter:
  - `grep [A-Z] file.txt`

– The following will match any number from 0 to 5
  - `grep [0-5] file.txt`

– The following will match any read ID that starts with a 1, ends with a 2 and only has numbers from 1 to 5 in the middle:
  - `grep 'seq\.1[1-5]+2[ab]' reads.fa`

GREP: Non-Matching Characters

- Suppose you want to match a character that's anything but a T?
  - `[^T]` will achieve that.
- For example, if you want to find all rows in the file `reads.fa` that have five A's upstream of five G's with no T in between?
  - The following will not achieve that:

```
➢ grep AAAAA.*GGGGG reads.fa | grep –v AAAAA.*T.*GGGGG
```

- This will make sure there aren't any occurrences of five A's upstream of five G's with a T in between, which is too strong.
  - For example, it will not find this line, even though there is an occurrence of five A's upstream of five G's with no T in between:

AAAAACGGGGGTGGGGG

GREP: Non-Matching Characters

- The grep syntax [^T] will match any (single) character except T.
- So, [^T]* will match any string of characters as long as none of them are T.
- Similarly, [^XYZ] will match any (single) character as long as it contains no X’s, Y’s or Z’s.
- We can now solve our problem on the previous slide as follows:

```
grep "AAAAA[^T]*GGGGG" reads.fa
```

- This will match AAAAACGGGGTGGGGG
  - Because it has “AAAAACGGGGG” in it.

grep and Python

(you will need this for HW5)

- The “re” in the word “grep” stands for “regular expression”.
  - Regular Expressions are basically wild-cards on steroids.
- You can call Unix commands from within python
  - We’ll see how later in these slides.
- But python also has its own built-in functions to do regular expressions.
  - No surprise, the function is called “re”.
- You must import the ‘re’ package.
- The following two functions perform matching and replacement.

```
import re
bool(re.search(pattern, string))
```

This returns True or False depending on whether the pattern is found in the string.

```
new_string = re.sub(pattern, replacement, original)
```

This takes the string “original” and replaces “pattern” with “replacement”

For example: `re.sub(‘A+’, ‘TTT’, “AAGGNNNAAAAAACCCCC”)`

This returns TTTGGNNNTTTCCCCC (all strings of A’s replaced with three T’s).

That’s enough info to solve the HW, but for those interested, this page has full info on the python re function:

https://docs.python.org/3/howto/regex.html

There’s a lot to it, but bioinformaticians must use regular expressions with great regularity, so eventually you must master them.

Review

- **grep**
  - Search for text in text files
- **clear**
  - Clear the screen
- **tar**
  - Pack several files into one for ease of transport and storage
- **gzip/gunzip**
  - Compress/decompress files for ease of transport and storage
- **history**
  - See a listing of previous commands
- **control key shortcuts**
  - Shortcuts for editing efficiency, using the control key
- **variables**
  - Name/value pairs that the system uses for various bookkeeping.

Practice Exercises 3

1. Put all lines in the file:
   - UCSC_mouse_knowngene_id_mapping

   that have the word 'globin' in them in another file called globins.txt
   - How many lines have the word 'globin'? Did you pay attention to case?
2. Now count the number of lines where the word 'globin' is not part of a bigger word like 'immunoglobin'.
3. Make a file called zinc_rows.txt that has *just the description field* of all lines where the word "zinc" appears, regardless of the capitalization.
   - How many lines are there?
4. Make a file called zinc_rows_sorted.txt which has the same rows as zinc_rows.txt but in sorted (lexical) order.
5. Are there any lines in your file where the word 'finger' appears *before* the word 'zinc'?
6. How many rows have the word 'zinc' appearing at least twice?
   - Find rows that have ‘zinc’ appearing exactly twice.

Practice Exercises 3

- continued -

7. How many times does a command appear on your history list with the string 'cd' in it?
   - Use pipe, grep, wc to find out without actually counting them by hand.
8. You have a file called reads.fa. This file has sequences with names that alternately end in 'a' and 'b'.
   1. Get all the 'a' entries into a file called forward.fa (get both the name line and the following sequence line, for each one). Do it without there being any extra lines in forward.fa, just the name lines and the sequence lines.
   2. Similarly, get the 'b' sequences into a file called reverse.fa

Practice Exercises 3

- continued -

9. Consider the file sequence_reads1.fa
   a) How many sequences have at least six C's in a row?
   b) How many have six C's but no more than six?
10. Consider the anchors ^ and $ used in a grep search
    a) How would you use the anchors to grep for a blank line in a file?
    b) Use this to count the number of blank lines in the file UCSC_mouse_knowngene_id_mapping:

Using Excel

- It is very common to receive data in the form of an Excel spreadsheet.
- An Excel spreadsheet is a binary file, we cannot just treat it as text.
- But Excel allows you to export a spreadsheet to text format.
  - Click on "Save As" and then choose "Text – Tab delimited" from the list of file formats. It will ask you if you are sure, because things like graphs will be left out of the text file.
- This should save a text file on your system that you can now treat like any other text file.

Excel

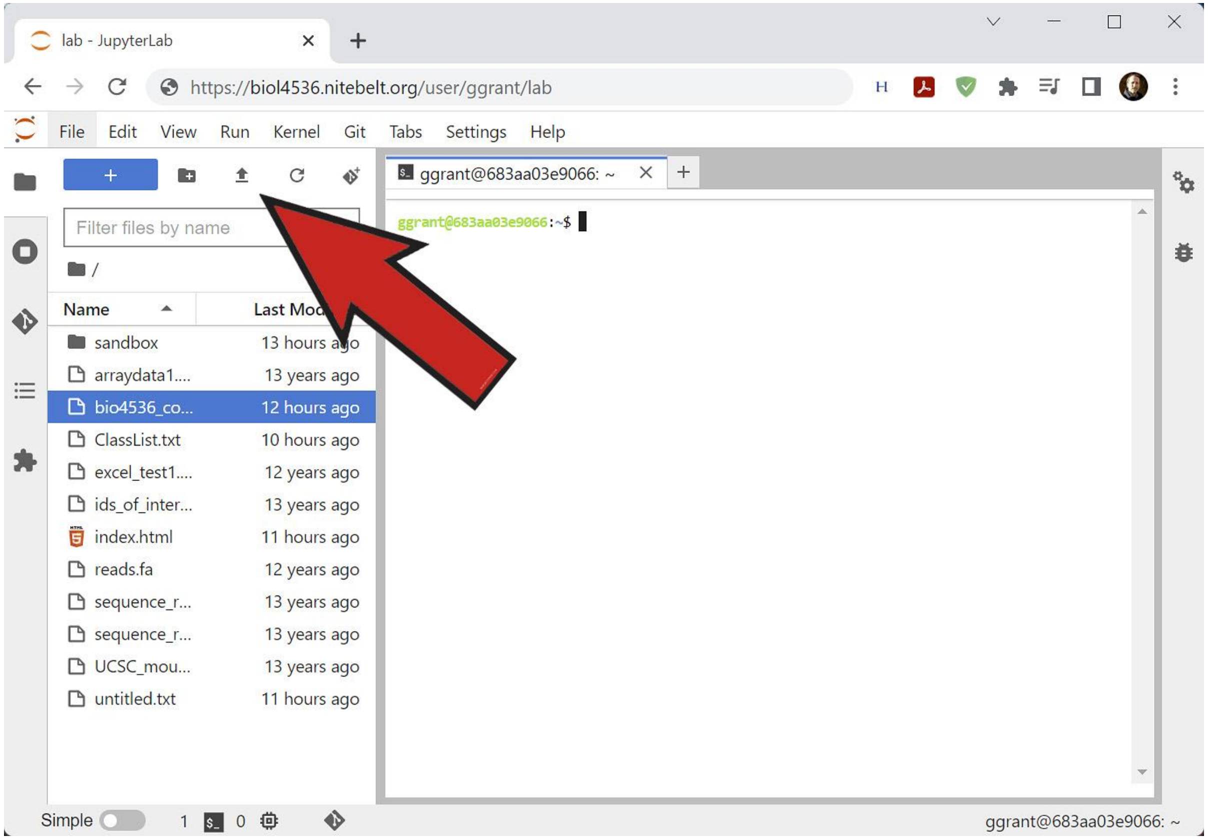
- Open Excel and start a new blank document. In the document, make a table with five rows and four columns as follows:

Id	Exp1	Exp2	Exp3
ID1	2	2	4
ID2	3	4.5	2
ID3	2.5	3	5
ID4	1	0	3

- Now save this as tab delimited text, with the name excel_test2.txt

Upload

- Upload the file to your home directory



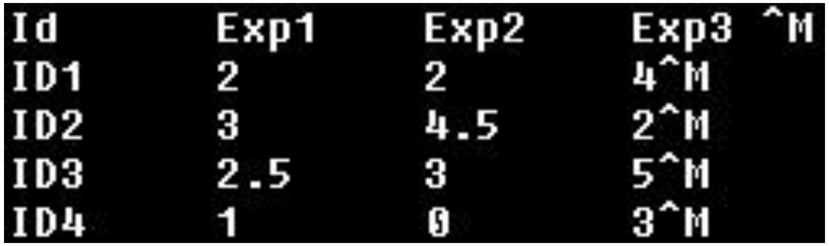
Name	Last Mod
sandbox	13 hours ago
arraydata1....	13 years ago
bio4536_co...	12 hours ago
ClassList.txt	10 hours ago
excel_test1....	12 years ago
ids_of_inter...	13 years ago
index.html	11 hours ago
reads.fa	12 years ago
sequence_r...	13 years ago
sequence_r...	13 years ago
UCSC_mou...	13 years ago
untitled.txt	11 hours ago

```
ggrant@683aa03e9066:~$
```

- Cat the file to see what's in it. It should display a nice clean text version of the excel spreadsheet.

Trojan New Line Characters

- Enter the following:
  - ➢ cat -v excel_test2.txt
- You should see this:



```
Id        Exp1      Exp2      Exp3 ^M
ID1       2         2         4^M
ID2       3         4.5       2^M
ID3       2.5       3         5^M
ID4       1         0         3^M
```

- The –v option tells cat to display weird characters.
- Those '^M' things are non-Unix-friendly newlines put there by Excel.
  - – And many other programs do the same.
- Some Unix programs are smart enough to ignore them (like cat), others will choke on them.
- If somebody gives you a text file and something is not working with it, this is one of the first things you should check.
  - – Use cat –v and look for funky characters.

Running Programs

- Often tools for processing biological data are only available as command line programs.
- Types of programs we might encounter:
  - Perl, Python, Ruby programs
    - More commonly known as 'scripts', these run anywhere you have Unix
  - Java programs
    - Convenient because the same code will run anywhere, once java is installed on your system, and Java is available for all systems.
  - C and C++ programs
    - These are the fastest languages, but the code must be tailored for each operating system and so it might work on some but not on others.
    - And you may have to "compile" the code yourself, more on this later.

Source Code vs. Binaries

- Programs in Java, C and C++ are created as text files (called '*source code*') which are then transformed into the binary files.
  - It’s those binary files (called ‘executables’) that you actually run.
  - The transformation from source code to binary files is called 'compilation'.
  - Java and C are called 'compiled' languages.
- In contrast for Perl, Python, Ruby programs there is no difference between the file of code and the file you run.
  - These are called 'interpreted' languages.

Perl, Python, Ruby

- interpreted languages -

- Python is a workhorse of bioinformatics, with more Python code in the field right now than any other language.
- Part of the appeal of interpreted languages is that they are fast easy languages to write short programs with.
  - *Terminology*: small down-and-dirty programs are often called '*scripts*', however there is no real sharp distinction between what we call a 'program' and what we call a 'script'.
  - Which scripting language people use is largely a matter of taste.
- Perl, Python and Ruby run anywhere you have Unix.

Perl Example

- We are going to run a couple of perl scripts.
  - Nothing deep, just a simple example to get the idea of how to run a script.
- Make sure you’re in your home directory, then copy this file there:

```
cp /data/perlscripts.tar.gz .
```

- And unpack it

```
tar –xvzf perlscripts.tar.gz
```

- You should now have the following three scripts:
  1. table2columnwisepercents.pl
  2. modify_fa_to_have_seq_on_one_line.pl
  3. primes.pl

Perl Example

Run the first one by entering the following:

```
perl table2columnwisepercents.pl
```

This runs the script with no options or parameters.

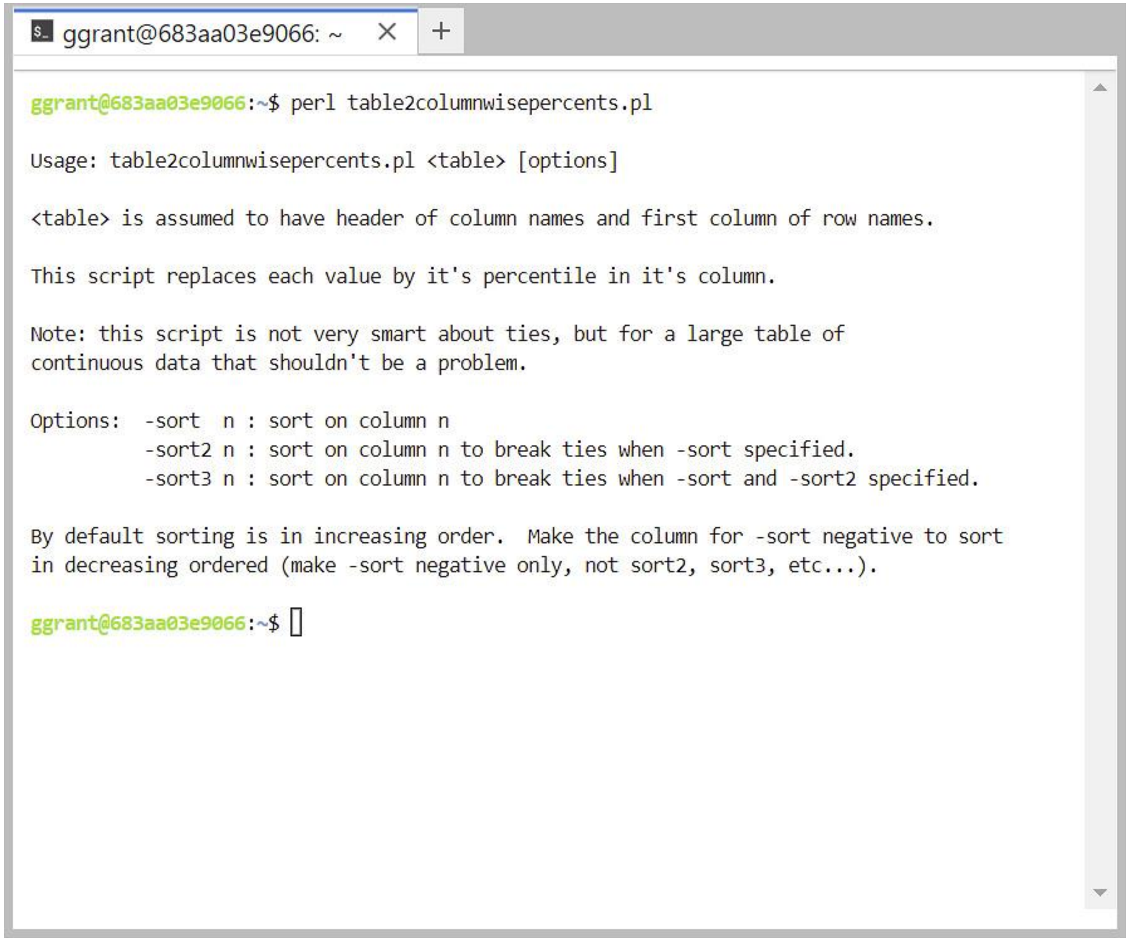
If you do that, it will return what we call ‘usage’

Some scripts are documented better than others, it depends on the author and how much effort they made to document it, so this might not always work, but it's always worth a try.

Usage Syntax

- The first line shows the basic usage:
  - Usage: table2columnwisepercents.pl <table> [options]

- There is one required argument <table> and some optional arguments as well.
  - It's convention that required arguments go betwee pointy brackets and optional arguments between square brackets.

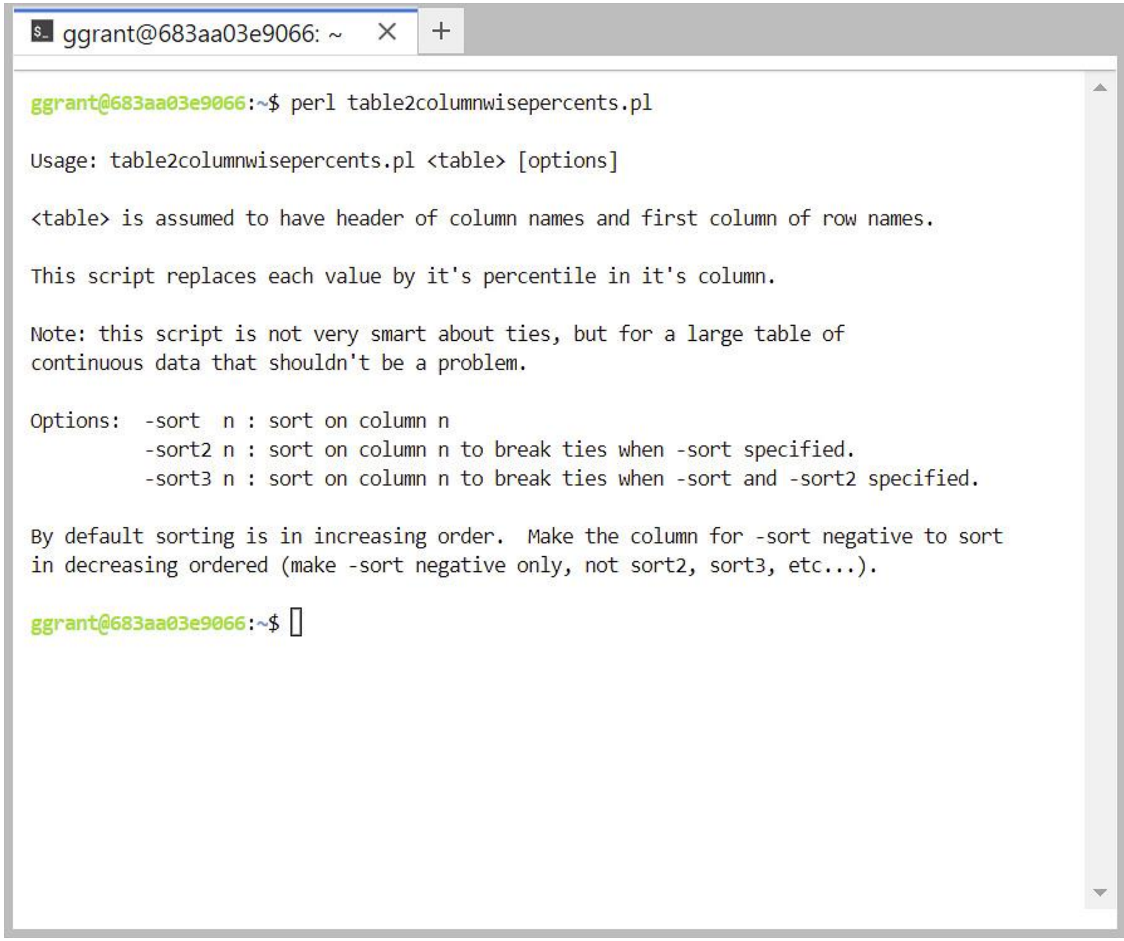


```
ggrant@683aa03e9066:~$ perl table2columnwisepercents.pl

Usage: table2columnwisepercents.pl <table> [options]

<table> is assumed to have header of column names and first column of row names.

This script replaces each value by it's percentile in it's column.

Note: this script is not very smart about ties, but for a large table of
continuous data that shouldn't be a problem.

Options:  -sort  n : sort on column n
          -sort2 n : sort on column n to break ties when -sort specified.
          -sort3 n : sort on column n to break ties when -sort and -sort2 specified.

By default sorting is in increasing order.  Make the column for -sort negative to sort
in decreasing ordered (make -sort negative only, not sort2, sort3, etc...).

ggrant@683aa03e9066:~$ 
```

Perl Example

- continued -

- This script is also kind enough to tell us what it does:
  - This script replaces each value by its percentile in its column.
- As well as how it expects the file <table> to be formatted:
  - <table> is assumed to have header of column names and first column of row names.

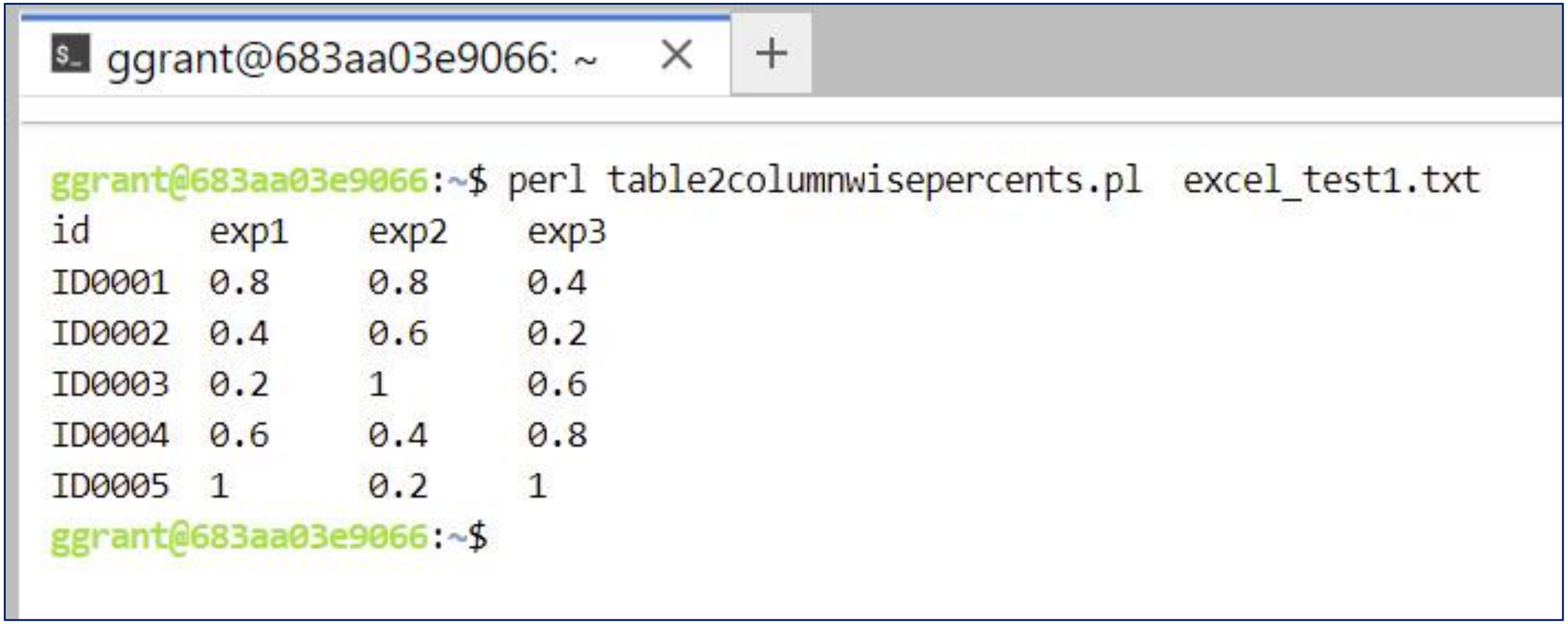


```
ggrant@683aa03e9066:~$ perl table2columnwisepercents.pl

Usage: table2columnwisepercents.pl <table> [options]

<table> is assumed to have header of column names and first column of row names.

This script replaces each value by it's percentile in it's column.

Note: this script is not very smart about ties, but for a large table of
continuous data that shouldn't be a problem.

Options:  -sort  n : sort on column n
          -sort2 n : sort on column n to break ties when -sort specified.
          -sort3 n : sort on column n to break ties when -sort and -sort2 specified.

By default sorting is in increasing order.  Make the column for -sort negative to sort
in decreasing ordered (make -sort negative only, not sort2, sort3, etc...).

ggrant@683aa03e9066:~$ 
```

Perl Example

- continued -

- Run the script on that file we created with excel:
  - perl table2columnwisepercents.pl excel_test1.txt
- It should return the table of percentiles. See if you can figure out how to get it to sort on the second column.



```
ggrant@683aa03e9066:~$ perl table2columnwisepercents.pl  excel_test1.txt
id      exp1    exp2    exp3
ID0001  0.8     0.8     0.4
ID0002  0.4     0.6     0.2
ID0003  0.2     1       0.6
ID0004  0.6     0.4     0.8
ID0005  1       0.2     1
ggrant@683aa03e9066:~$
```

Python and Ruby

- From a user's perspective there really is no difference between a perl script, a python script or a ruby script.
  - Except to run a perl script you precede the script name with 'perl', to run a python script you precede it with 'python', and to run a ruby script you use 'ruby'.
- The difference in these languages is primarily relevant for the programmers writing the scripts.
  - Perl is generally considered more down-and-dirty but is by far the most popular. Python and Ruby are considered more elegant, but it really depends on who you ask.

Compiled Languages
C/C++ vs. Java

- In compiled languages, you may be given binaries, or you may be given source code that you have to compile yourself into binaries.
- C and C++ programs must be written and complied differently for each system.
  - That is why you often must choose which versions of a program to download, depending on your operating system.
    - It's not unusual that the author compiles binaries for one OS and not for another. You've probably seen Windows only or Mac only programs.
  - For corporate software if something is not available for your OS you might be out of luck.
  - In academics people tend to make the source code available, so you can try to compile it for your system if binaries are not available. Compilation for your system may or may not be easy to do.
- Java is more universal. You can compile it once and it will run anywhere. That's one reason people like Java. Typically for Java programs you are given the binaries and they work everywhere.
- We will be running some compiled programs later in the course.

Running Command-Line Programs
- generalities -

- In modern biology you often operate on very large files and programs can take a very long time to run.
- As such, you may want to:
  - Run a program in the background while you do other things
  - Suspend a program so you can hibernate your computer.
  - Kill a program because you get tired of waiting or because it seems to have gotten stuck somewhere.
- Unix offers you several ways to run, monitor and terminate programs, we look at those next.

Generalities of Running Programs

- continued -

- Run the perl script findprimes.pl. It prints out one prime every second. It will never finish, to stop it use control-c.
- Now redirect the output to a file:
  - perl findprimes.pl > primes.txt
- Let it run for ten seconds or so, and kill it using control-c. Cat the file primes.txt to see how many primes were written to it before you killed it.
- Now run it again and wait ten seconds again, but instead of hitting control-c, this time hit control-z.

Generalities of Running Programs

- continued -

- cat the file and see what is in there.
- By doing control-z, you haven't killed the program, you have just suspended it.
  - Enter 'jobs' and you should see the perl job listed as "Stopped".
  - Enter "fg" (for 'foreground') to start it back up. If there is more than one job running, you can enter 'fg n' where n is the job number.
- Wait ten more seconds and then kill it using control-c.
- Cat the file again, it should have grown by a few more primes compared to before.

Generalities of Running Programs

- running jobs in the background -

- Run primes.pl again:
  - ➢perl primes.pl > primes.txt
- Do control-z to suspend it. Now enter 'bg' (for 'background').
- Now enter 'jobs'.
- Now you should see the program listed as "Running".
  - – It is running in the background.
- Cat the file primes.txt, wait a few seconds and do it again, the file should be growing.

Generalities of Running Programs

- running jobs in the background -

- Enter "tail –f primes.txt", that will display the file primes.txt in real time as it grows. Watch it for five or ten seconds and then hit control-c to quit out of "tail –f".
- To kill the perl script, enter 'fg' to bring it back to the foreground, and then control-c.
- Another way to run things in the background, and to do it all in one step, is to put an ampersand "&" after the command:
  - ➢perl findprimes.pl > primes.txt &

Generalities of Running Programs

- killing jobs -

- Do 'jobs' and make sure nothing is running, if anything is, terminate it.
- Run primes.pl in the background again:
  - ➢perl primes.pl > primes.txt &
- Do 'jobs' to make sure it's running.
- Now open a new session (don't close the current one, just open a second one).
- Do 'jobs' in the second window, notice you do not see the job running, because a session does not show what's running in other sessions.

Generalities of Running Programs

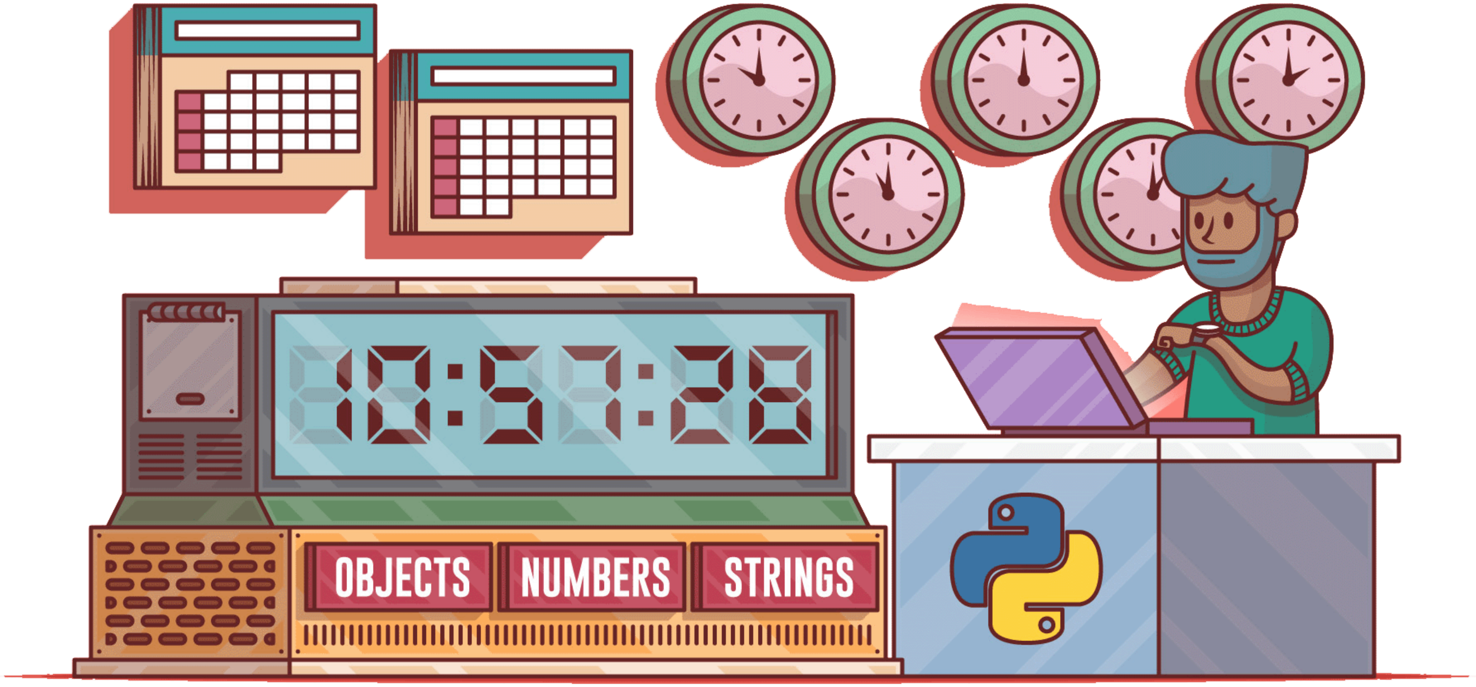
- killing jobs -

- You can see the other processes running by using the 'ps' function. Enter 'ps x'. You should see one entry on the list that indicates a perl script running.
- Note the PID of this process (PID stands for "Process ID").
- If the PID is N, enter "$\mathrm{kill\ N}$". Now go to the first window and do a 'jobs' and the process should have been terminated.
  - Sometimes jobs don't like to die, to insist use '$\mathrm{kill\ -9\ N}$'.
- Now you know how to kill a job from another window.

Wrapping UNIX Commands in Python

- You can call a UNIX command from your python script.

- It may perform an action, like deleting a file.
- Or it may return some output, like calling “cat” on a file.
  - In the latter case you can capture the output in a variable.

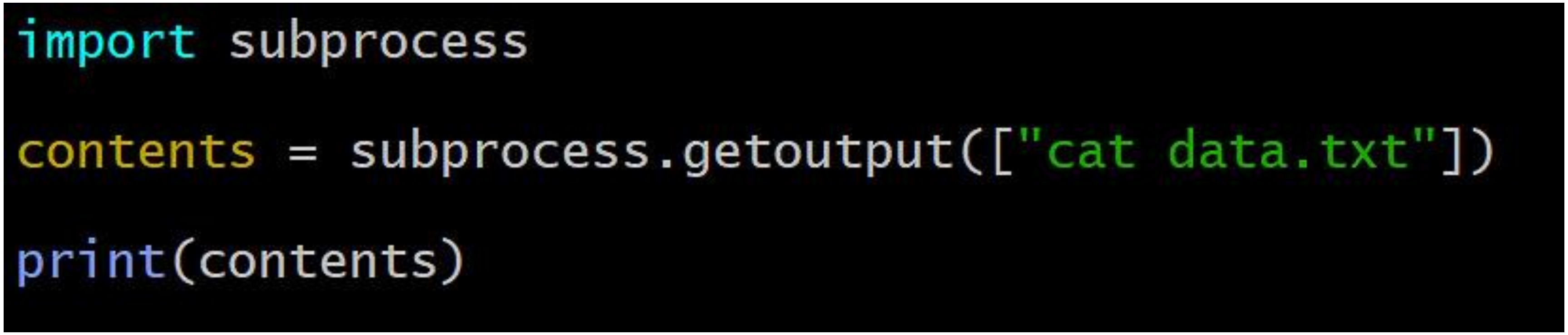


Example: Unix from Python

- Suppose the file data.txt has the four lines of text:

```
>seq1
AGATCAG
>seq2
GATCAGG
```

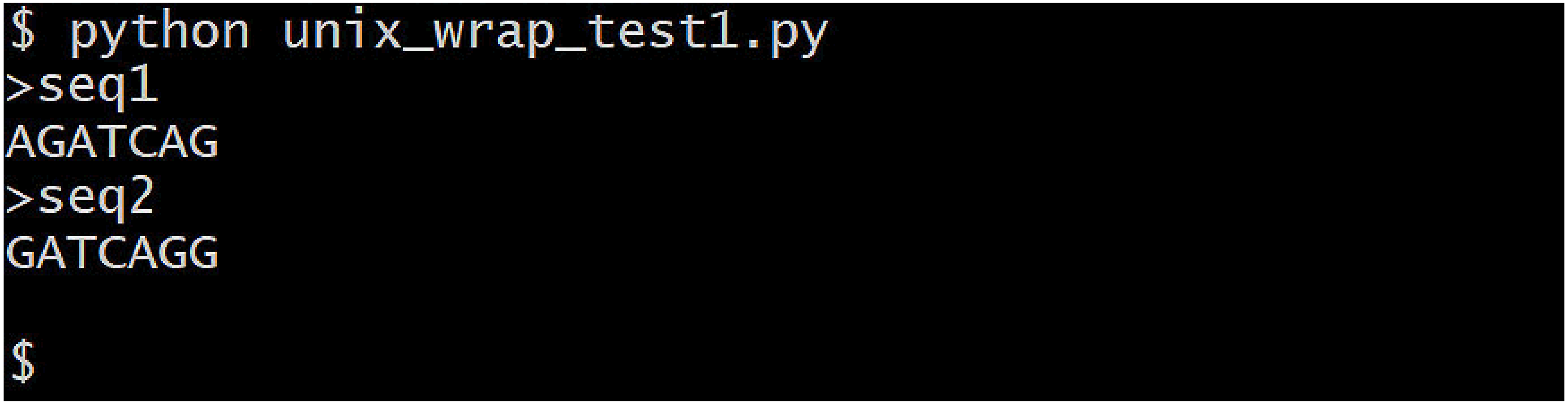
- We want to capture the contents of the file in a variable.
  - The following achieves that.
  - We’ve named this program: unix_wrap_test1.py



```
import subprocess

contents = subprocess.getoutput(["cat data.txt"])

print(contents)
```

Example: Unix from Python

- Running the program

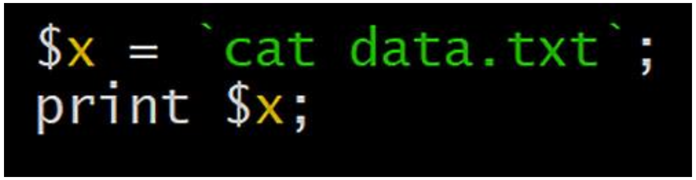


```
$ python unix_wrap_test1.py
>seq1
AGATCAG
>seq2
GATCAGG

$
```

- Notice how it printed two newlines at the end.
  - The file had a newline at the end.
  - The python print command also puts a newline at the end.
  - So, you end up with two.

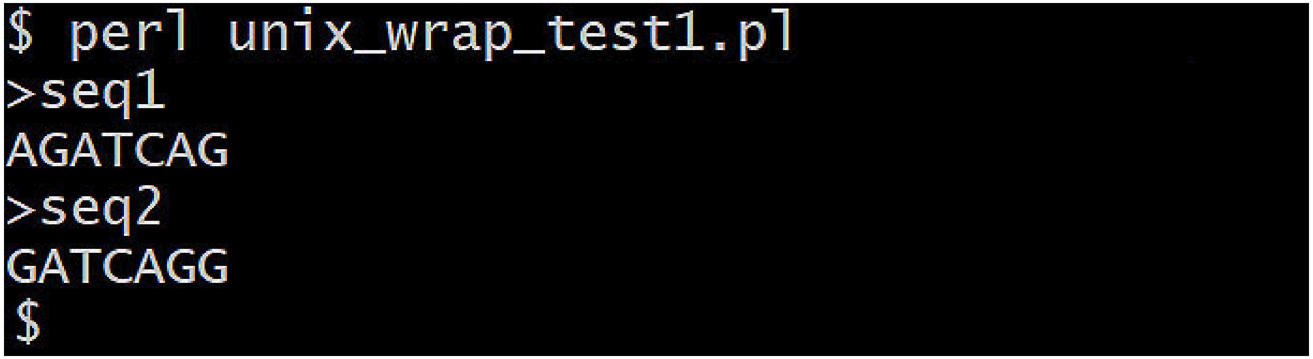
Now … PERL

- Let’s do the exact same thing in perl.



```
$x = `cat data.txt`;
print $x;
```

- Two short lines.
  – Just put the command between back ticks.



```
$ perl unix_wrap_test1.pl
>seq1
AGATCAG
>seq2
GATCAGG
$
```

- Output is the same, except the print command does not add another newline.

Shell Scripts

- Suppose you need to execute a sequence of commands repeatedly.
- You can put them in a file and "source" the file.
- For example, make a file named

`commands1.sh`

with the following three lines

```
grep NM_010922 arraydata1.txt | head -1 | cut -f 6
grep NM_028889 arraydata1.txt | head -1 | cut -f 6
grep NM_207141 arraydata1.txt | head -1 | cut -f 6
```

  - Each command grabs the sixth column of tab delimited text for the first row with a specific ID.

- Now execute:

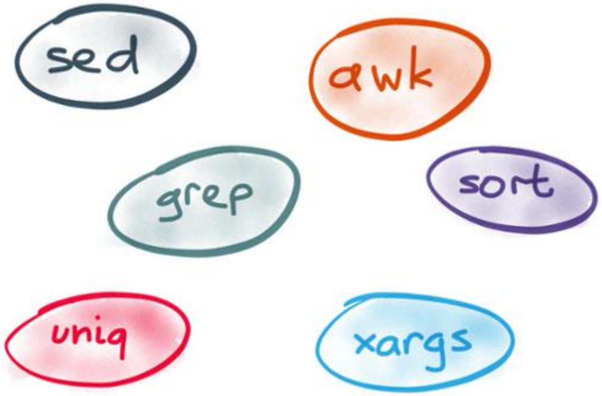
```
> source commands1.sh
```

- Assuming the file arraydata1.txt exists, it will return three rows of output.

Shell Scripts

- Just as python is a programming language from which you can execute UNIX commands in, ``bash'' is also a programming language that you can execute UNIX commands from.
  - It's in fact a programming language that was made specifically for that job of executing commands.
- So, executing commands in a bash script is easy.
  - You don't need to call them with a function like in python or use backticks like in perl, you just put the command directly and verbatim into the script and it knows what to do.
- On the other hand, loops and other mechanisms of program flow that you are used to and exist in all procedural programming languages are clunkier and more particular in a bash script than in almost any other language.
  - That’s the tradeoff, if your program is heavy on UNIX commands and not much else, then use a shell script, if it’s a complex program heavy on program flow (lots of loops and other stuff), then use python.

More Commands



- The following web page has a nice overview of about 60 Unix commands, plus other useful info.
  - You will want to read this if you decide to become serious about UNIX.
- https://www.freecodecamp.org/news/the-linux-commands-handbook

- This handbook does not try to cover everything under the sun related to Linux and its commands.

- It focuses on the small core commands that you will use the 80% or 90% of the time, and tries to simplify the usage of the more complex ones.

UNIX is a lifestyle, a community, a universe

- It's the most inclusive community imaginable, all you need to join is a little bit of knowledge.