$ Welcome to the Matrix Unix/Linux

Computational Biology

Lecture 1

Dr. Chris Bird

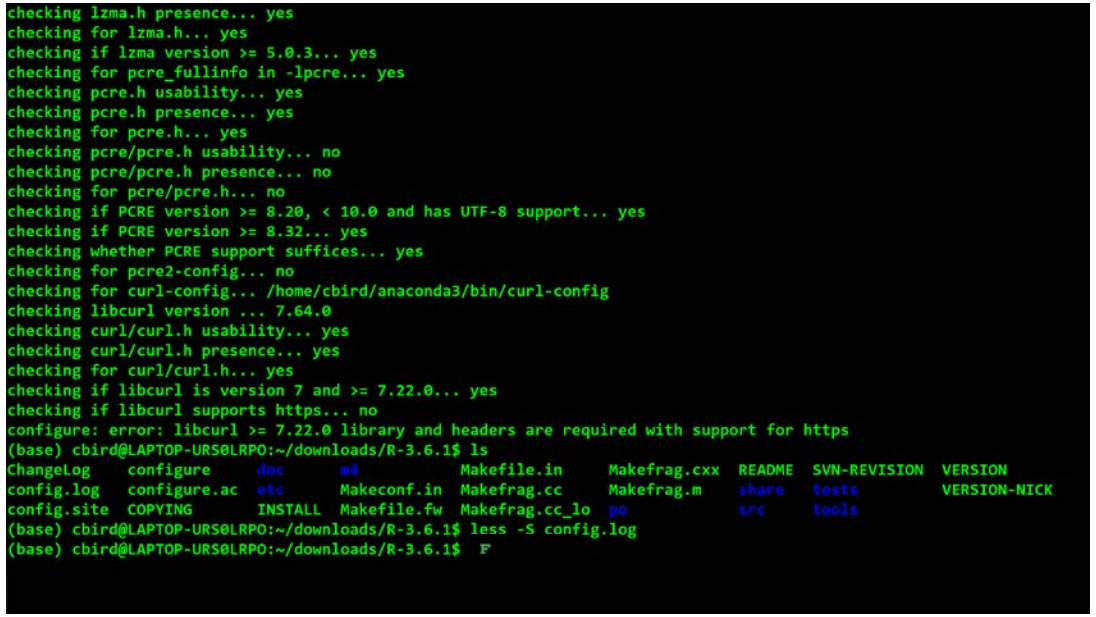
LINUX is a Free & Open Source Version of the UNIX Operating System

- An operating system is the primary interface between you and the computer
- Open source is a decentralized development model where all aspects of a project are viewable and generally free to use
- Linux is free
  - Supercomputers
  - Useful text manipulation tools

2 Primary Methods of Interfacing with Computers

Graphical User Interface (GUI)

Command-line Interface (CLI)

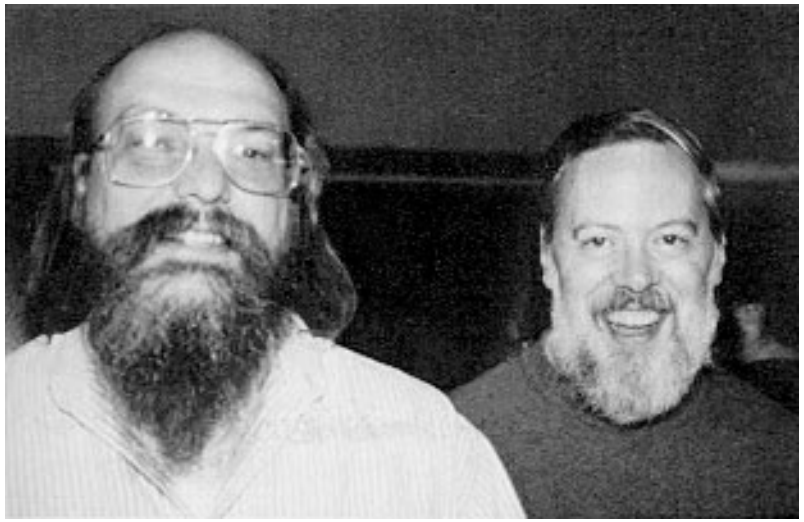


Why use CLI Linux?

- Free
- Automation
- Flexibility
- Powerful
- Designed for developers
- Supercomputers use it
- Many software tools for biologists
- Large body of support online

The UNIX Philosophy

- One program (command) does one thing
- All programs accept input as a text stream and output a modified text stream
- Programs can be linked together into serial pipelines to achieve complex results



The Unix philosophy (excerpt):

-Make each program do one thing well.

-Expect the output of every program to become the input to another, as yet unknown program.

McIlroy, Pinson & Tague, 1978

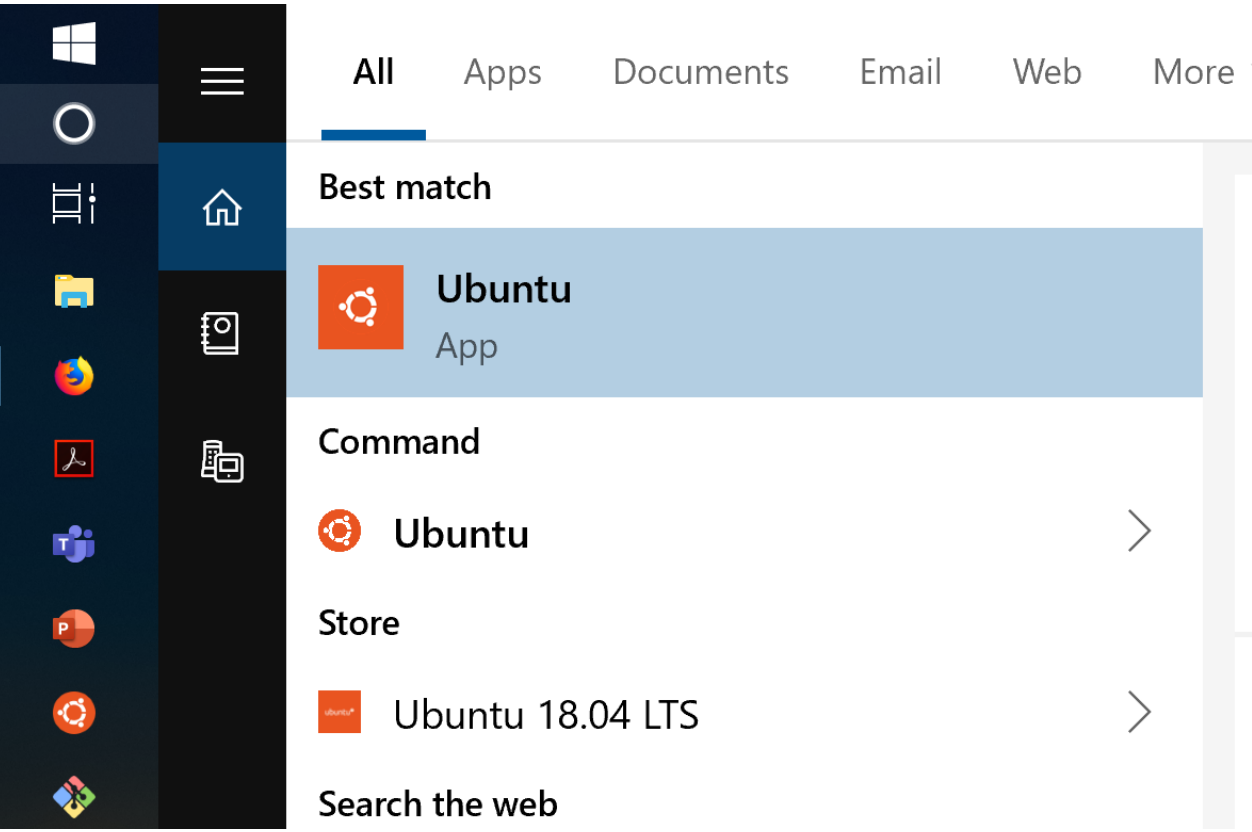
Linux CLI Pipelines Facilitate Scientific Reproducibility and Long-Term Efficiency

Comparison of GUI and CLI for manipulating data

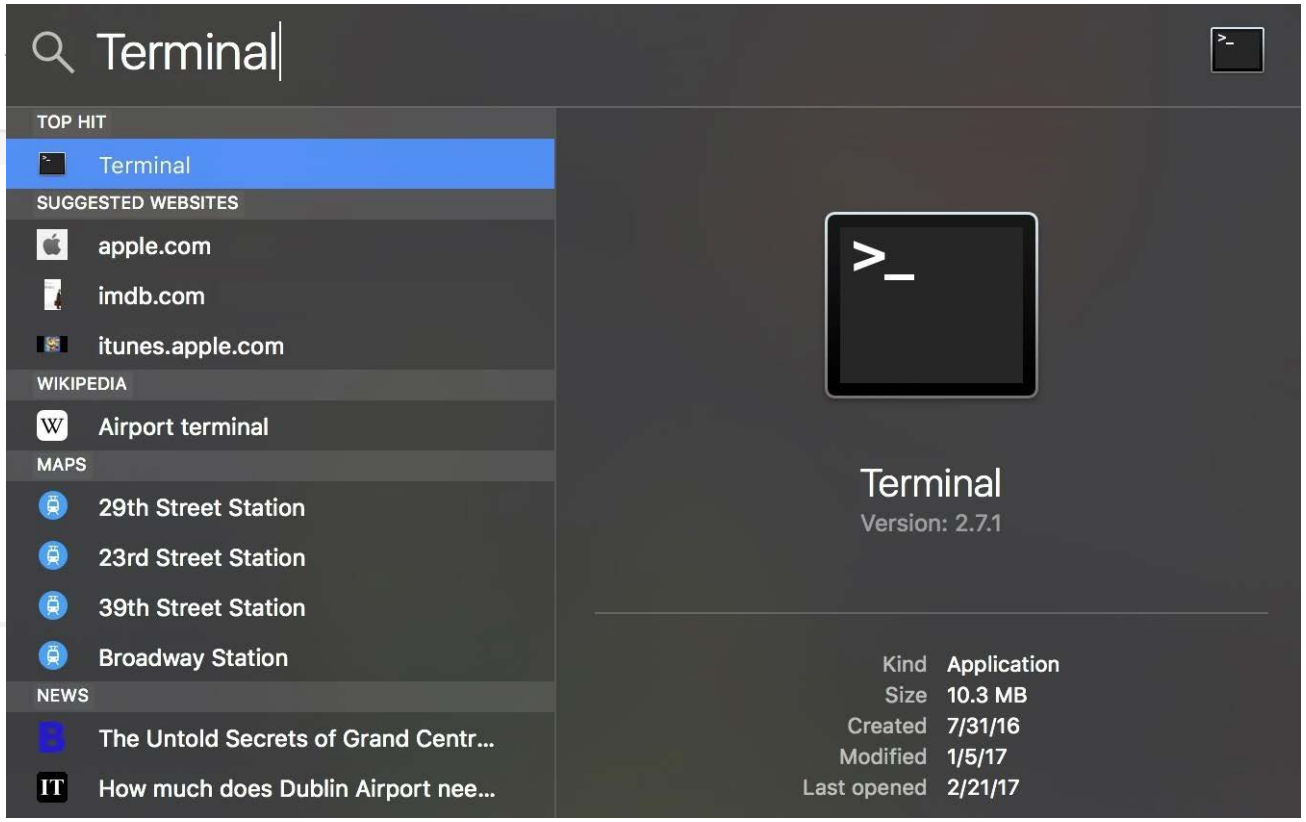
	GUI	CLI
Learning curve	**Short, shallow**	Long, steep
Amount of your time taken to process large amounts of data	Long	**Short**
Process Documented or Recorded	Often not, mouse clicks	**Always**
Ability to identify mistake	Poor	**Excellent**
Time to recover from mistake	Long	**Short**
Ease for another lab to reproduce	Difficult to impossible	**Simple**

Open A Terminal Window

WIN10: Search Ubuntu

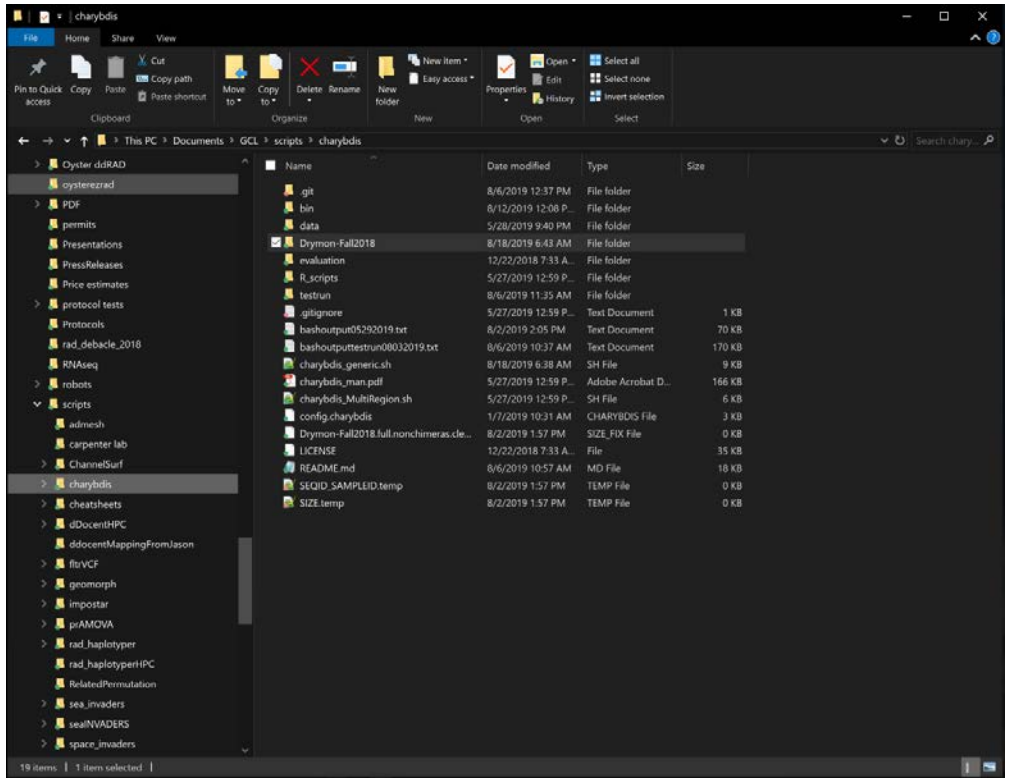


MacOS: Search Terminal

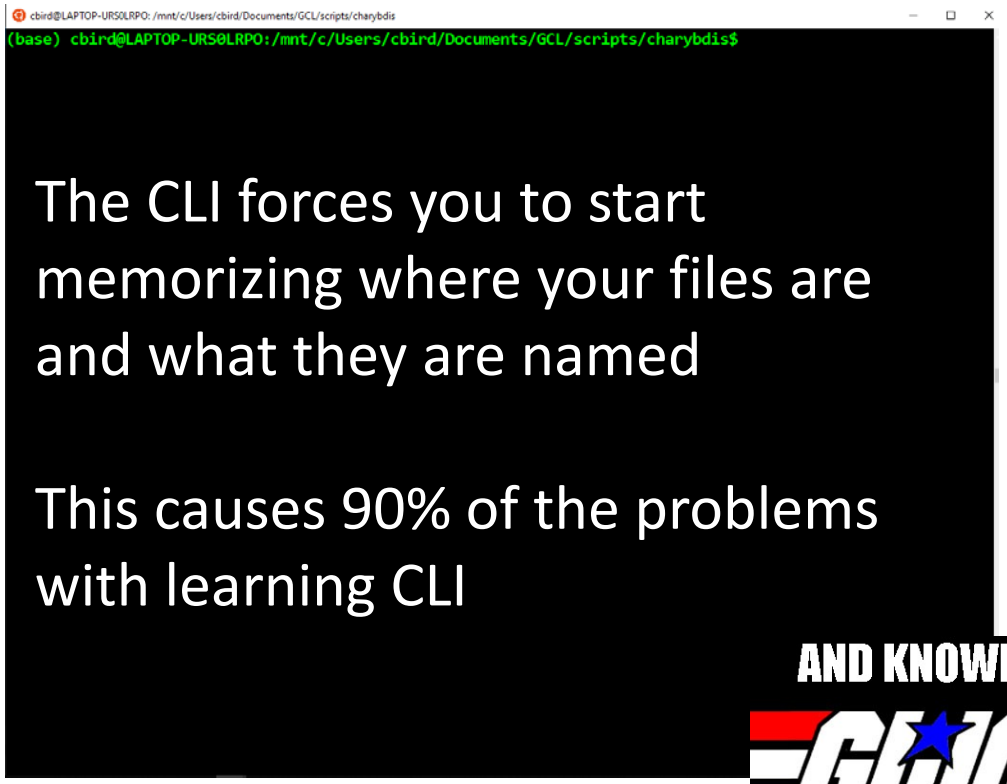


The Directory Structure is the Organization of Files and Folders (aka Directories) In Your Computer

WIN10 File Explorer

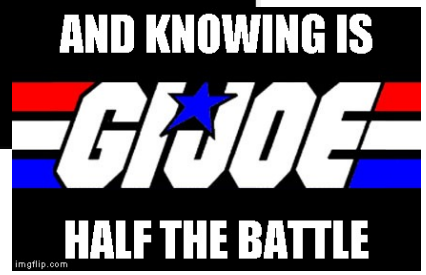


Ubuntu Terminal



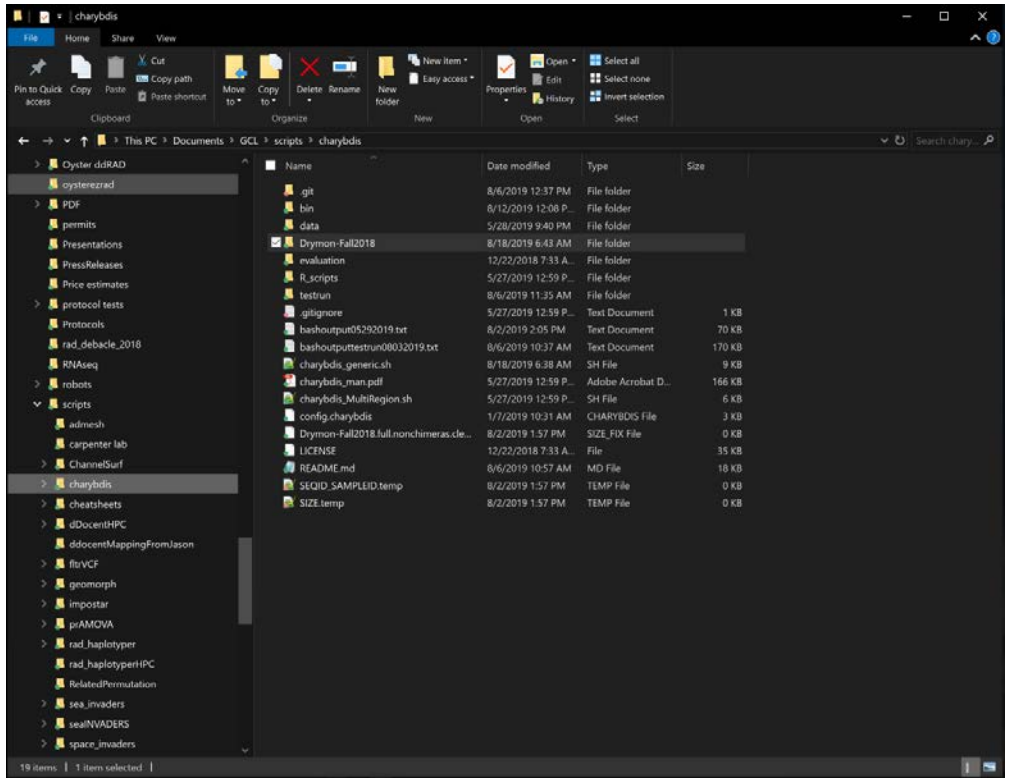
The CLI forces you to start memorizing where your files are and what they are named

This causes 90% of the problems with learning CLI

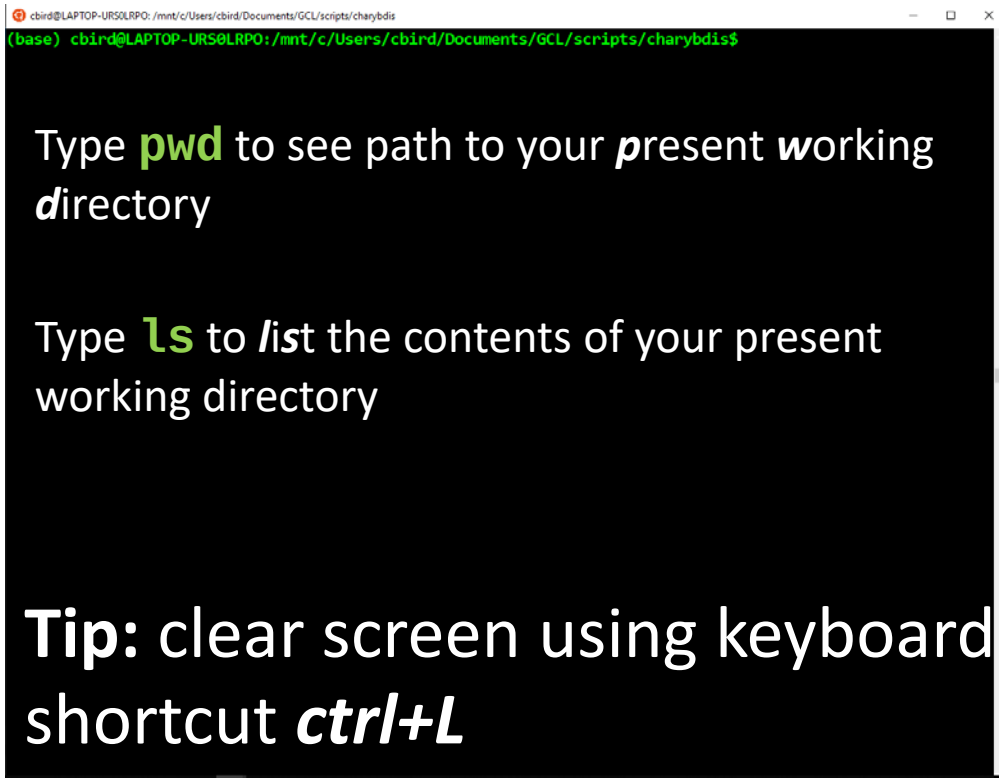


The Directory Structure is the Organization of Files and Folders (aka Directories) In Your Computer

WIN10 File Explorer



Ubuntu Terminal



Unix/Linux Command Line Terminology

The **path** is the address of a file or directory in the directory structure

Description	***Path*** **in Unix, Linux, Ubuntu, MacOS, Android**	**Path in Windows**
Root, or top of the directory tree	/	c:\
A **file** named file.txt in the root dir	/file.txt	c:\file.txt
A **directory** named folder1 in the root dir	/folder1	c:\folder1
A file named dna.txt in folder1	/folder1/dna.txt	c:\folder1\dna.txt

Important Directories

/bin

- Contains several basic programs

/dev

- Contains the files connecting to devices such as the keyboard, mouse, and screen

/etc

- Contains configuration files

/tmp

- Contains temporary files

cbird@LAPTOP-URS0LRPO: /mnt/c/Users/cbird/Documents/GCL/scripts/charybdis

```
(base) cbird@LAPTOP-URS0LRPO:/mnt/c/Users/cbird/Documents/GCL/scripts/charybdis$
```

Try using `ls` to view these directories

```
ls /bin
ls /dev
ls /etc
ls /tmp
```

Your Home Directory

/username/home

- Starting or login directory
- Specific to user
- Place for personal files, dirs, programs, downloads etc

$HOME

- The **path** to your home dir is stored in this **variable**
- A variable stores information
- Always preceded by a **$** after it is created
- **$HOME** is an **environmental variable** created by the operating system and bash

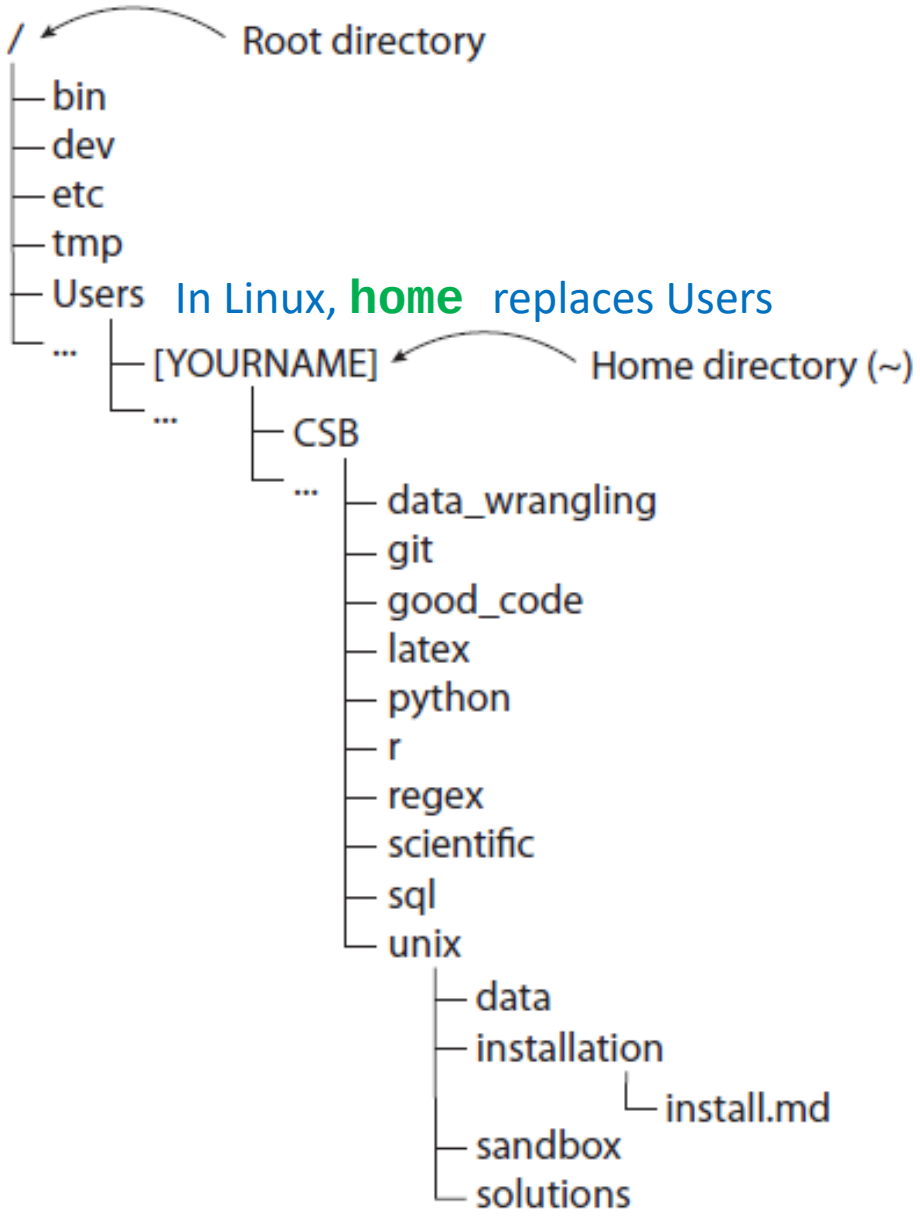
cbird@LAPTOP-URS0LRPO: /mnt/c/Users/cbird/Documents/GCL/scripts/charybdis

```
(base) cbird@LAPTOP-URS0LRPO:/mnt/c/Users/cbird/Documents/GCL/scripts/charybdis$
echo $HOME
pwd
ls
ls $HOME
ls ~
```

If you followed install instructions, you should have a **CSB** dir

Directory Tree

Showing Contents of $HOME/CSB/unix/installation



Full path of the file install.md:
`/Users/[YOURNAME]/CSB/unix/installation/install.md`

cbird@LAPTOP-URS0LRPO: /mnt/c/Users/cbird/Documents/GCL/scripts/charybdis

```
(base) cbird@LAPTOP-URS0LRPO:/mnt/c/Users/cbird/Documents/GCL/scripts/charybdis$
ls $HOME/CSB/unix/installation
```

On your own time, if you install **tree**, you can view the directory tree on screen

```
sudo apt-get install tree
cd $HOME
tree CSB
tree -L 1 CSB
tree -L 2 CSB
man tree
```

On mac:
brew install tree

CSB/unix Repository

CSB/unix/data

- Contains data for examples and exercises

CSB/unix/installation

- Contains instructions for installing software for this chapter

CSB/unix/sandbox

- Dir where we work and experiment

CSB/unix/solutions

- Solutions in code (bash) pseudocode (plain English) for your consultation when you get stuck with an exercise.

cbird@LAPTOP-URS0LRPO: /mnt/c/Users/cbird/Documents/GCL/scripts/charybdis

```
(base) cbird@LAPTOP-URS0LRPO:/mnt/c/Users/cbird/Documents/GCL/scripts/charybdis$
cd $HOME
ls CSB/unix
ls CSB/unix/data
ls CSB/unix/installation
ls CSB/unix/sandbox
ls CSB/unix/solutions
```

Tip: use the ↑ key to recall last command

$ Welcome to the Matrix

1.4 The Shell

Computational Biology

Lecture 1

Dr. Chris Bird

The Shell

- The **shell** is software that controls the operating system **kernel** and is accessed through a **terminal** window
- The shell we are using in Ubuntu and MacOS is **BASH**, or **Born Again Shell**
- The **commands** we've been using are BASH commands which allow us to control the operating system

~

- Indicates where I am, the home dir

$

- Indicates the terminal is ready to accept commands
- From here forward, $<space><command> indicates you should type the command into the terminal

#

- A hash symbol means that everything that follows is a comment, usually in Engish

cbird@LAPTOP-URS0LRPO: ~

```
(base) cbird@LAPTOP-URS0LRPO:~$
```

Below, I've indicated that I want you do do what follows the # by typing the command that follows the $ and you expect your output to be similar to the line(s) not preceded by # or $

```
# display the date and time
$ date
Sat Aug 24 12:18:24 DST 2019
```

Bash Keyboard Shortcuts

↑	Scroll through previous commands
Tab	autocomplete command, dir, or file name
	if you hit tab and nothing happens there's either multiple matches or 0 matches
Tab,Tab	show matches
Ctrl+A	Go to the beginning of the line.
Ctrl+E	Go to the end of the line.
Ctrl+L	Clear the screen.
Ctrl+U	Clear the line before the cursor position.
Ctrl+K	Clear the line after the cursor.
Ctrl+C	Kill the command that is currently running.
Ctrl+D	Exit the current shell.
Alt+F	Move cursor forward one word (in OS X, Esc+F).
Alt+B	Move cursor backward one word (in OS X, Esc+B).

cbird@LAPTOP-URS0LRPO: ~

```
(base) cbird@LAPTOP-URS0LRPO:~$

# try some of the shortcuts
$
```

Bash Commands

`cal 2020 -j`

- **Commands** like `cal` are programs that follow the UNIX philosophy
- **Arguments** like `2020` are essentially options, order usually matters and some commands require particular arguments
  - `cp` or copy requires at least which file to copy and where to copy it, in that order
- `-j` is an **option**, in this case it means Julian calendar
  - --julian is the same as –j, options that are words are always preceded by two dashes

```
# print calendar
$ cal
 August 2019
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

$ cal 2020
$ cal –j
$ cal --julian
$ cal –j 2020
```

If you want to stop a command, ***ctrl+c***

Getting Bash Help

- It's impossible to remember all command and arguments

- If you know what you want to do, but you don't know the command
  - Google search "bash <English description of what you want to do>"

- If you know the command, but you don't know the arguments
  - `man <CommandName>`
  - All manuals have same format

```
# view calendar manual
$ man cal
NAME
  <name and brief descrip>
SYNOPSIS
  <examples of how to run>
DESCRIPTION
  <detailed description>
  <list of arguments/options>
```

Tip: scroll with arrow keys and close manual with ***q*** key

Changing Directories

`cd ..`

- Move up to parent directory

`cd /`

- Move to root directory

`cd ~`

- Move to home directory

`cd -`

- Move to last directory

`pwd`

- Path to present working dir

`ls`

- Show contents of present directory

```
# move around dir system
$ cd ..
$ pwd
$ cd /
$ pwd
$ cd -
$ pwd
$ cd ~
$ pwd
# show dir contents
$ ls
$ ls -l
$ ls -ltrh
```

Note: single letter *options* can typically be combined together, `-l -t -r -h`

Interpreting Output of `ls -l`

Dirs are highlighted below, files are not

```
(base) cbird@LAPTOP-URS0LRPO:~$ ls -ltrh
total 1.0K
-rwxrwxrwx 1 cbird cbird 515 Jul 10  2018 hosts
-rw-rw-rw- 1 cbird cbird 146 Jul 10  2018 initialize.bash
-rw-rw-rw- 1 cbird cbird  39 Aug  2  2018 tamucchpcmlogin.bash
-rw-rw-rw- 1 cbird cbird  42 Jan 11  2019 oduhpcmlogin.bash
-rw-rw-rw- 1 cbird cbird  61 Feb 15  2019 mntUSB.bash
-rw-rw-rw- 1 cbird cbird  93 Jun 21 06:46 onedrive.bash
drwxrwxrwx 1 cbird cbird 512 Aug 24 10:57 downloads
drwxrwxrwx 1 cbird cbird 512 Aug 24 11:25 CSB
(base) cbird@LAPTOP-URS0LRPO:~$
```

Interpreting Output of `ls -l`

Next Slide

		Usr	Grp	Size	Date	Names
-rwxrwxrwx	1	cbird	cbird	515	Jul 10 2018	hosts
-rw-rw-rw-	1	cbird	cbird	146	Jul 10 2018	initialize.bash
-rw-rw-rw-	1	cbird	cbird	39	Aug 2 2018	tamucchpcmlogin.bash
-rw-rw-rw-	1	cbird	cbird	42	Jan 11 2019	oduhpcmlogin.bash
-rw-rw-rw-	1	cbird	cbird	61	Feb 15 2019	mntUSB.bash
-rw-rw-rw-	1	cbird	cbird	93	Jun 21 06:46	onedrive.bash
drwxrwxrwx	1	cbird	cbird	512	Aug 24 10:57	downloads
drwxrwxrwx	1	cbird	cbird	512	Aug 24 11:25	CSB

```
(base) cbird@LAPTOP-URS0LRPO:~$
```

Interpreting Output of `ls -l`

Permissions

	User	Group	Global
File	-rw-	rw-	rw-
Dir	drwx	rwx	rwx

Write

Read

Execute

Paths

- A path is the address of file or directory
- An **absolute path** is complete and starts with root `/` or a variable that starts with root
  - These return the same result regardless of pwd

```
/home/<username>/CSB
~/CSB
$HOME/CSBB
```

- **Relative paths** start from the present location
  - These only work if you are in the right dir
  - . Means present directory
  - .. means parent directory

```
./CSB
CSB
```

- It's best not to used spaces in dir and file names
  - See pg 21 for dealing w/ spaces

```
$ cd ~

# show contents of CSB dir
# absolute paths
$ ls /home/<username>/CSB
$ ls ~/CSB
$ ls $HOME/CSB

#relative paths
$ ls ./CSB
$ ls CSB
```

Note: if a path includes a space, either wrap path in quotes or precede each space with `\`

Mind Expander 1.1

Computational Biology

Lecture 1

Dr. Chris Bird

$ Welcome to the Matrix

1.5 Commands to Remember

Computational Biology

Lecture 1

Dr. Chris Bird

Copy with `cp <from> <to>`

```
# goto sandbox
$ cd ~/CSB/unix/sandbox

# copy the following file to the present directory
$ cp ../data/Buzzard2015_about.txt .

# copy file and rename it in present dir
$ cp ../data/Buzzard2015_about.txt ./Buzzard2015_about2.txt

# copy whole data dir to present dir, then view present dir
$ cp –rf ../data .
$ ls
```

Note: -r means recursive, -f means force

Move or rename with **mv <from> <to>**

```
# make sure you are still in sandbox, if not then cd ~/CSB/unix/sandbox
$ pwd

#move the file to the data directory
$ mv Buzzard2015_about2.txt ../data

# rename a file that isn't in your pwd
$ mv ../data/Buzzard2015_about2.txt ../data/Buzzard2015_about_new.txt

# check your work
$ ls ../data
```

Note: bash gives no positive feedback, only negative if something is wrong

Create file with `touch <filename>`

```
# make sure you are still in sandbox, if not then cd ~/CSB/unix/sandbox
$ pwd

# inspect the current contents of the directory
$ ls -l

# create a new file (you can list multiple files)
$ touch new_file.txt

# inspect the contents of the directory again
$ $ ls -l

# if you touch the file a second time, the time of last access will change
$ touch new_file.txt
$ ls -l
```

Note: bash gives no positive feedback, only negative if something is wrong

Remove file(s) or dir(s) with `rm <name>`
Make dir with `mkdir <name>`

```
# make sure you are still in sandbox, if not then cd ~/CSB/unix/sandbox
$ pwd

# delete new_file.txt in sandbox, the –i requests confirmation
$ rm -i new_file.txt

# make dir d1 in present dir, d2 in d1, and d3 in d2; if you have tree try it
$ mkdir -p d1/d2/d3
$ tree d1
d1
└── d2
    └── d3

# remove the d1,d2,& d3 dirs recursively
$ rm -rf d1
```

be careful with `rm`, you could delete your whole computer and there is no undo

View large files with	`less -S <filename>`
Print and concatenate files	`cat <filename>`
Print and sort files	`sort <filename>`

```
# move to the data dir
$ cd ~/CSB/unix/data

# look at DNA alignment file, try duckduckgo search on “bash less commands”
$ less –S Marra2014_data.fasta

# type /ATCG inside of less to search; u=up, d=down, G=end, g=begin, q=exit

# concatenate files and/or print to screen
$ cat Marra2014_about.txt Gesquiere2011_about.txt Buzzard2015_about.txt

# print the sorted lines of a file
$ sort Gesquiere2011_data.csv

# sort numerically by column 2 in reverse order and view in less
$ sort –n –k2 –r Gesquiere2011_data.csv | less
```

Count words with `wc <filename>`

Determine file type `file <filename>`

```
# count lines, words, and characters
$ wc Gesquiere2011_about.txt

# count lines only
$ wc -l Marra2014_about.txt

# determine file type, ASCII is a type of human-readable text file
$ file Marra2014_about.txt
Marra2014_about.txt: ASCII English text
```

Don’t forget to use *Tab* key to autocomplete names, prevents spelling mistakes

Get beginning of file **head -n # <filename>**

Get end of file **tail -n # <filename>**

```
# display first two lines of a file
$ head -n 2 Gesquiere2011_data.csv

# display last two lines of file
$ tail -n 2 Gesquiere2011_data.csv

# display from line 2 onward
# (i.e., removing the header of the file)
$ tail -n +2 Gesquiere2011_data.csv

# display all but the last line
$ head -n -1 Gesquiere2011_data.csv
```

Don’t forget to use *Tab* key to autocomplete names, prevents spelling mistakes

Mind Expander 1.2

Computational Biology

Lecture 1

Dr. Chris Bird

$ Welcome to the Matrix

1.6 Advanced Commands

Computational Biology

Lecture 1

Dr. Chris Bird

Redirection of output (stdout) to file	`[command] > filename`
Append stdout to file	`[command] >> filename`
Redirect contents of file to stdin	`[command] < filename`

```
# let's start by moving to our sandbox
$ cd ~/CSB/unix/sandbox

# print text to screen, then print to file, then print file to screen
$ echo "My first line"
$ echo "My first line" > test.txt
$ cat test.txt

# append file with additional text, then print file to screen
$ echo "My second line" >> test.txt
$ cat test.txt
```

Don't forget to use *Tab* key to autocomplete names, prevents spelling mistakes

Problem Solving Scenario

- A machine provides you with thousands of data files
- There's so many, it's breaking your file browser
- How many files are there?

- We will use unix/data/Saavedra2013 as an example of a directory with many files

```
# save file names to file in pwd
$ ls ../data/Saavedra2013 > filelist.txt

# look at the file
$ cat filelist.txt

# count lines in a file
$ wc -l filelist.txt

# remove the file
$ rm filelist.txt
```

Problem Solving Scenario – Application of pipe |

- A pipe passes the stdout from one command to the stdin of another
- How many files are there?

```
# list file names
$ ls ../data/Saavedra2013

# list file names and pipe into wc
$ ls ../data/Saavedra2013 | wc –l
59 filelist.txt
```

TSV and CSV Data Files

	Ozone	Solar.R	Wind	Temp	Month	Day
128	32.0	92.0	15.5	84	9	6
78	61.0	285.0	6.3	84	7	18
105	65.0	157.0	9.7	80	8	14
64	NaN	101.0	10.9	84	7	4
98	122.0	255.0	4.0	89	8	7
145	36.0	139.0	10.3	81	9	23
27	23.0	13.0	12.0	67	5	28
28	45.0	252.0	14.9	81	5	29
113	9.0	36.0	14.3	72	8	22
132	24.0	259.0	9.7	73	9	10

- Tab Separated Values (TSV)
  - Tabs denote columns
- Comma Separated Values (CSV)
  - Commas denote columns
- Tidy data
  - Each row is one unit of observation
  - Each column is one dimension or aspect of the units of observation
- File extensions not always accurate

It’s Easy to Convert Among Formats Using **tr**

```
# view contents of csv
$ less -S ../data/Pacifici2013_data.csv

# replace semicolons with commas using tr [find] [replace]
$ cat ../data/Pacifici2013_data.csv | tr “;” “,” | less –S

# view as tsv
# \t is the nearly universal symbol for tab
$ cat ../data/Pacifici2013_data.csv | tr ";" "\t" | less -S
```

tr is short for <u>tr</u>anslate

Using **cut** to grab columns and **head** to grab rows

```
# change directory
$ cd ~/CSB/unix/data

# display first line of file (i.e., header of CSV file)
$ head -n 1 Pacifici2013_data.csv

# display first column of file
$ cut -d ";" –f 1 Pacifici2013_data.csv

# display second through fourth columns
$ cut -d ";" -f 2-4 Pacifici2013_data.csv

# display first “cell” of data
$ head -n 1 Pacifici2013_data.csv | cut -d ";" -f 1
```

Connecting cut head tail sort uniq

```
# select 2nd column, display first 5 elements
$ cut -d ";" -f 2 Pacifici2013_data.csv | head -n 5

# select 2nd and 8th columns, display first 3 elements
$ cut -d ";" -f 2,8 Pacifici2013_data.csv | head -n 3

# select 2nd column without header, show 5 first elements
$ cut -d ";" -f 2 Pacifici2013_data.csv | tail -n +2 | head -n 5

# identify the orders in csv
# select 2nd column without header, unique sorted elements
$ cut -d ";" -f 2 Pacifici2013_data.csv | tail -n +2 | sort |\
> uniq

# count how many records per order in csv
$ cut -d ";" -f 2 Pacifici2013_data.csv | tail -n +2 | sort |\
> uniq -c
```

Mind Expander 1.3

Computational Biology

Lecture 1

Dr. Chris Bird

Substituting Characters Using & Predefined Characters

tr

[:upper:]

```
# change all a to b
$ echo "aaaabbb" | tr "a" "b"
bbbbbbb

$ echo "123456789" | tr 1-5 0
000006789

$ echo "ACtGGcAaTT" | tr actg ACTG
ACTGGCAATT

$ echo "ACtGGcAaTT" | tr [:lower:] [:upper:]
ACTGGCAATT

$ echo "aabbccddee" | tr a-c 1-3
112233ddee
```

Substituting Characters Using & Predefined Characters

tr
[:upper:]

```
# delete all occurrences of a
$ echo "aaaaabbbb" | tr -d a
bbbb

# remove consecutive duplicate occurrences of a
$ echo "aaaaabbbb" | tr -s a
Abbbb

# move to sandbox and list files
cd ../sandbox; ls
```

`;` is equivalent to end of line

`tr` does not accept a file as an argument, always use pipe

Make a new file BodyMass.csv in sandbox dir based on Pacifici2013_data.csv, columns 2-6, remove header, sort lines according to body mass (large to small), change ; to spaces

```
# 1. View header row to refresh your memory
$ head -n1
# 2. Start building pipe, use less to view
$ cut -d ";" -f2-6 ../data/Pacifici2013_data.csv | less -S
# 3. Add to pipe, use less to view
$ cut -d ";" -f2-6 ../data/Pacifici2013_data.csv | \
> tr ";" "\t" | less -S
# 4. Add to pipe, figure out sort options, use less to view
$ cut -d ";" -f2-6 ../data/Pacifici2013_data.csv | \
> tr ";" " " | tail -n+2 | sort -nrk6 | less -S
# 5. Instead of piping to less, redirect output to file
$ cut -d ";" -f2-6 ../data/Pacifici2013_data.csv | \
> tr ";" " " | tail -n+2 | sort -nrk6 > BodyMass.csv
```

\ is actually the “escape character”, what follows the \ is treated differently

Wildcards are Symbols that Represent Multiple Characters

*

- Zero or more characters, except leading dot

?

- Any single character, except leading dot

```
# goto miRNA dir inside data dir
$ cd ~/CSB/unix/data/miRNA

# count the numbers of lines in all the .fasta files
$ wc -l *.fasta

# print the first two lines of each file
# whose name starts with pp
$ head -n 2 pp*

# determine the type of every file that has
# an extension with exactly three letters
$ file *.???
```

Selecting lines with matching pattern using
grep [options] [pattern] filename

- Every line that matches pattern is returned
- Many options to increase functionality
- Regular Expressions are used for pattern matching in text files
  - A language of wildcards
  - 2 syntaxes: POSIX, Perl

```
$ cd ~/CSB/unix/sandbox

# how many wombats (fam Vombatidae)?
$ grep "Vombatidae" BodyMass.csv
$ grep -c "Vombatidae" BodyMass.csv

# which cattle are in file?
$ grep "Bos" BodyMass.csv

# Only match whole words
$ grep –w "Bos" BodyMass.csv

# Make search case insensitive
$ grep –i "Bos" BodyMass.csv
```

Selecting lines with matching pattern using

grep [options] [pattern] filename

```
# which mammals have body weight most similar to the gorilla?
# option –B lines before match, option –A lines after match
$ grep -B 2 -A 2 "Gorilla gorilla" BodyMass.csv
# show line number of gorilla
$ grep -n "Gorilla gorilla" BodyMass.csv

# -v means match anything except pattern
$ grep Gorilla BodyMass.csv | grep -v gorilla

# return all lines with Gorilla or Pan, note use of escape char \
$ grep -w "Gorilla\|Pan" BodyMass.csv

# return all lines with Gorilla for all files in data dir
# and it’s subdirs. -r recursive, searches subdirs
$ grep -r "Gorilla" ../data
```

Searching for files with `find [dir] [options] [pattern]`

```
# current directory is the unix sandbox
$ find ../data
# how many files are in data?
$ find ../data | wc -l
# find file named n30.txt in data
$ find ../data -name "n30.txt"
# use wildcards to find all files in data that contain about
$ find ../data -iname "*about*“
# count all files that end in .txt in data, then
# do same but don’t include subdirs
$ find ../data -name "*.txt" | wc –l
$ find ../data -maxdepth 1 -name "*.txt" | wc –l
# count files in data that don’t include about
$ find ../data -not -name "*about*" | wc –l
# find directories with data in path or name
$ find ../data -type d
```

Mind Expander 1.4

Computational Biology

Lecture 1

Dr. Chris Bird

Permissions

- Three types of permissions
  - Read, Write, Execute
  - Program won't run if x is not set
- Three types of users
  - User, Group, Global
- View with `ls -l`
- Change with `chmod`

```
-rwxrwxrwx 1 cbird cbird 515 Jul 10  2018
-rw-rw-rw- 1 cbird cbird 146 Jul 10  2018
-rw-rw-rw- 1 cbird cbird  39 Aug  2  2018
-rw-rw-rw- 1 cbird cbird  42 Jan 11  2019
-rw-rw-rw- 1 cbird cbird  61 Feb 15  2019
-rw-rw-rw- 1 cbird cbird  93 Jun 21 06:46
drwxrwxrwx 1 cbird cbird 512 Aug 24 10:57
drwxrwxrwx 1 cbird cbird 512 Aug 24 11:25
(base) cbird@LAPTOP-URS0LRPO:~$
```

Interpreting Output of `ls -l`

Permissions

	User	Group	Global
File	-rw-	rw-	rw-
Dir	drwx	rwx	rwx

Write

Read

Execute

Setting File Permissions with
chmod [options] ### filename

- Setting permissions using “octal” numeric system
  - read = 4
  - write = 2
  - execute =1
- Simply add numbers together for different combos of permissions
- Each combo is only represented by one number

```
# create a file in the unix sandbox
$ touch permissions.txt
$ ls -l
# change permissions so that user can r,w,x;
# group can r,x; and global can r
$ chmod 754 permissions.txt
$ ls -l
# give everybody full permissions
$ chmod 777 permissions.txt
$ ls -l
# give yourself full permissions, but only let
# others read your files
$ chmod 744 permissions.txt
$ ls -l
```

Super User Do to Execute Command as Administrator: **sudo**
Change Owners With **chown**

- Use **sudo** when computer tells you no
  - Make sure you are certain that you are right and computer is wrong to not execute your command
- You'll need **sudo** for installing software

```
# create a directory with a subdirectory
$ mkdir -p test_dir/test_subdir
$ ls -l
$ ls -l test_dir

# list valid users
$ cut –d: -f1 /etc/passwd

# change owner of dir, -R includes subdirs
$ chown -R ValidUserName test_dir/
$ sudo chown -R ValidUserName test_dir/
$ ls –l
$ ls –l test_dir
# change owner back to you
$ sudo chown -R YourUserName test_dir/
```

$ Welcome to the Matrix

1.7 Scripting

Computational Biology

Lecture 1

Dr. Chris Bird

Scripting

- A script is a file with a list of commands
- Commands are executed sequentially
- Here we create a simple script

```
# create a script in the unix sandbox
$ touch ExtractBodyM.sh
# open ExtractBodyM.sh in GUI text editor
gedit ExtractBodyM.sh
open –a bbedit ExtractBodyM.sh
open ExtractBodyM.sh
```

ctrl-c will quit a command running in the terminal

```
# open ExtractBodyM.sh in CLI text editor
$ nano ExtractBodyM.sh
```

Either type in or copy and paste the pipeline we made previously to make `BodyM.csv` into `ExtractBodyM.sh`

```
cut –d “;” -f 2-6 ../data/Pacifici2013_data.csv | tr ";" " " | tail -n+2 | sort -nrk6 > BodyM.csv
```

Scripting

- It is important to write comments in English to describe what the script is doing
  - You'll forget
  - Makes it easier for others to figure out what's happening
  - Easier to identify errors

ctrl-x to exit nano, then y, then enter

```
# run ExtractBodyM.sh script
$ bash ExtractBodyM.sh
```

I noticed that there had to be spaces after the options for this to run correctly

```
$ ls –ltrh
$ nano ExtractBodyM.sh
```

Add the following comments to the script before the code using nano

```
# isolate columns 2-6 of csv using cut
# translate the ; to " " using tr
# remove the header row using tail
# sort by sixth column, descending order
# save to file
```

ctrl-o, then enter to save changes made in nano without closing

Don't forget to use the tab key to autocomplete file names

cbird@LAPTOP-URS0LRPO: ~/CSB/unix/sandbox

```
  GNU nano 2.5.3                    File: ExtractBodyM.sh

# isolate columns 2-6 of csv using cut
# translate the ; to “ “ using tr
# remove the header row using tail
# sort by sixth column, descending order
# save to file

cut -d ";" -f 2-6 ../data/Pacifici2013_data.csv | tr ";" " " | tail -n+2 | sort -nrk6 > BodyM.csv

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     ^Y Prev Page
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Linter   ^_ Go To Line  ^V Next Page
```

Scripting

- Script is hardcoded
  - Only works with one input and one output file

- In nano replace:

../data/Pacifici2013_data.csv with $1

BodyM.csv with $2

- $1 and $2 are variables

nano

```
# isolate columns 2-6 of csv using cut
# translate the ; to “ “ using tr
# remove the header row using tail
# sort by sixth column, descending order
# save to file

cut –d “;” -f 2-6 $1| \
tr ";" " " | \
tail -n+2 | \
sort -nrk6 > $2
```

I added escape characters \ because my code wouldn’t fit in 1 line. Here they allow 1 line of code to be written across several lines

Ctrl-x, then y, then enter to exit nano

Scripting

- Now we must include arguments to run the ExtractBodyM.sh script
  - In file
  - Out file

- We can make the script executable by changing permissions with chmod and adding a shebang! To the first line of the script
  - A shebang is tells the computer which language the script is in

```
# run ExtractBodyM.sh script
$ bash ExtractBodyM.sh \
> ../data/Pacifici2013_data.csv \
> BodyM.csv
# change permissions so script is
# executable
$ chmod 777 ExtractBodyM.sh
# add shebang! to beginning of script
# cat glues files together. The $() opens
# an invisible shell and runs
# echo “#!/bin/bash”, producing a line of
# text that is added to ExtractBodyM.sh
$ cat $(echo “#!/bin/bash”) ExtractBodyM.sh
# you could also add the shebang! in nano
$ nano ExtractBodyM.sh
```

- Edit script in GUI to match script to the right

```
gedit ExtractBodyM.sh
open ExtractBodyM.sh
```

  - Add 2nd & 3rd lines
  - Use escape characters to separate pipeline by command
  - Copy comments from lines 4-8 and paste above the appropriate command in the pipeline
  - Save and close

Open ▾ | ExtractBodyM.sh ~/CSB/unix/sandbox | Save

File Edit View Search Tools Documents Help

```
#!/bin/bash

# to run do this:
# ./ExtractBodyM.sh [infile] [outfile]

# isolate columns 2-6 of csv (first argument) using cut
# translate the ; to “ “ using tr
# remove the header row using tail
# sort by sixth column, descending order
# save to file (second argument)

# isolate columns 2-6 of csv (first argument) using cut
cut -d ";" -f 2-6 $1 | \
 # translate the ; to “ “ using tr
 tr ";" " " | \
 # remove the header row using tail
 tail -n+2 | \
 # sort by sixth column, descending order
 # save to file (second argument)
 sort -nrk6 > $2
```

sh ▾ Tab Width: 8 ▾ Ln 20, Col 2 ▾ INS

Scripting

- Make sure the script works

- Now that it's executable, we can use `./` to run it rather than **bash**

```
# run ExtractBodyM.sh script
$ ./ExtractBodyM.sh \
> ../data/Pacifici2013_data.csv \
> BodyMass.csv
$
```

$ Welcome to the Matrix

1.8 For Loops

Computational Biology

Lecture 1

Dr. Chris Bird

For Loops: `for [variableName] in [list]`

- For loops automate repetitive tasks
  - 1 task, 100 files
  - Same task, many different arguments
- In the examples to the right the variable is "`file`", the list is composed of either two or all of the fasta files
- When the for loop starts, `file` takes on the value of the first item in the list, `ggo_miR.fasta`
- In the second line of the loop, the command is run on the value in `$file`
- The `done` means goto first line of `for` loop
- `file` takes on the value of the second item in the list, `hsa_miR.fasta`
- Etc…

```
$ cd ~/CSB/unix/data/miRNA
$ ls

# display first two lines of two fastas
$ for file in ggo_miR.fasta hsa_miR.fasta
> do head -n 2 $file
> done

# display first two lines of all fastas
$ for file in *.fasta
> do head -n 2 $file
> done
```

When setting a variable equal to a value, don't use a $.
When calling the value held in the variable, use a $

*.fasta is a list of all files that end with .fasta in the present dir

For Loops: `for [variableName] in [list]`

- Example 2: isolating DNA sequences from particular types of micro RNA
  - `miR-208a`, `miR-564`, `miR-3170`
- saving them into one file per type of miRNA
- Recall that `grep` returns lines that match a pattern
  - Pattern is `$miR`
  - What is the `-A1` argument doing?
- View resulting files

```
# display first two lines of two fastas
$ for miR in miR-208a miR-564 miR-3170
> do grep $miR -A1 *.fasta > $miR.fasta
> done

# Look at one of the files created
$ less -S miR-564.fasta
hsa_miR.fasta:>hsa-miR-564 MIMAT0003228
hsa_miR.fasta-AGGCACGGUGUCAGCAGGC
--
ppy_miR.fasta:>ppy-miR-564 MIMAT0016009
ppy_miR.fasta-AGGCACGGUGGCAGCAGGC
--
ptr_miR.fasta:>ptr-miR-564 MIMAT0008243
ptr_miR.fasta-AGGCACGGUGUCAGCAGGC
```

File Name

Sequence Name

Separator

Sequence

$ Welcome to the Matrix

1.9 Tips & Tricks

Computational Biology

Lecture 1

Dr. Chris Bird

$PATH

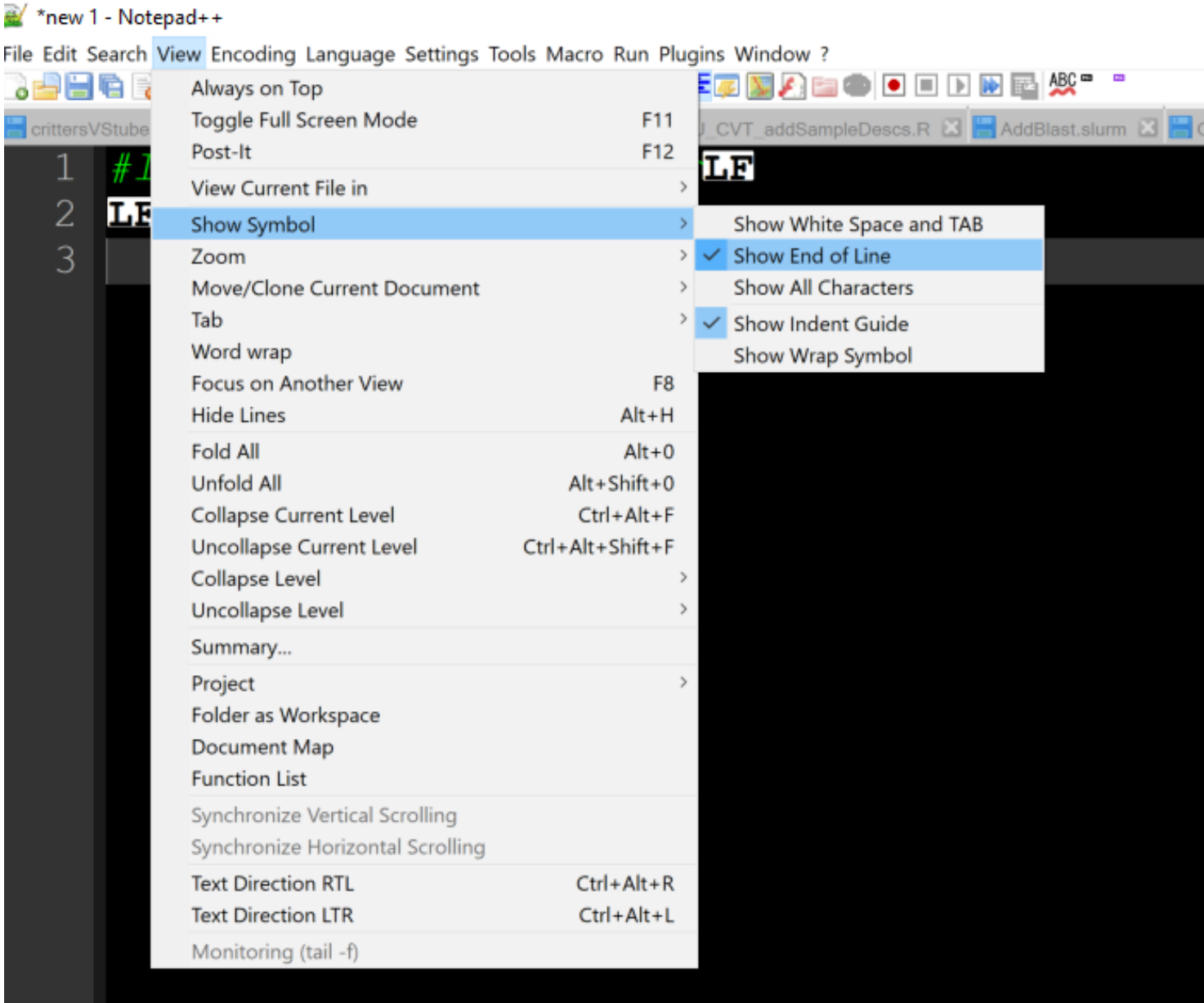
- A variable that holds all paths to directories where executable commands and scripts are located
- When you type `ls`, `bash` looks at `$PATH` to find the `ls` command file
- If you compile and install software manually, you need to move it to a $PATH dir `/usr/local/bin`

```
# show path variable
$ echo $PATH

# identify the path to the ls command
$ which ls
/bin/ls
```

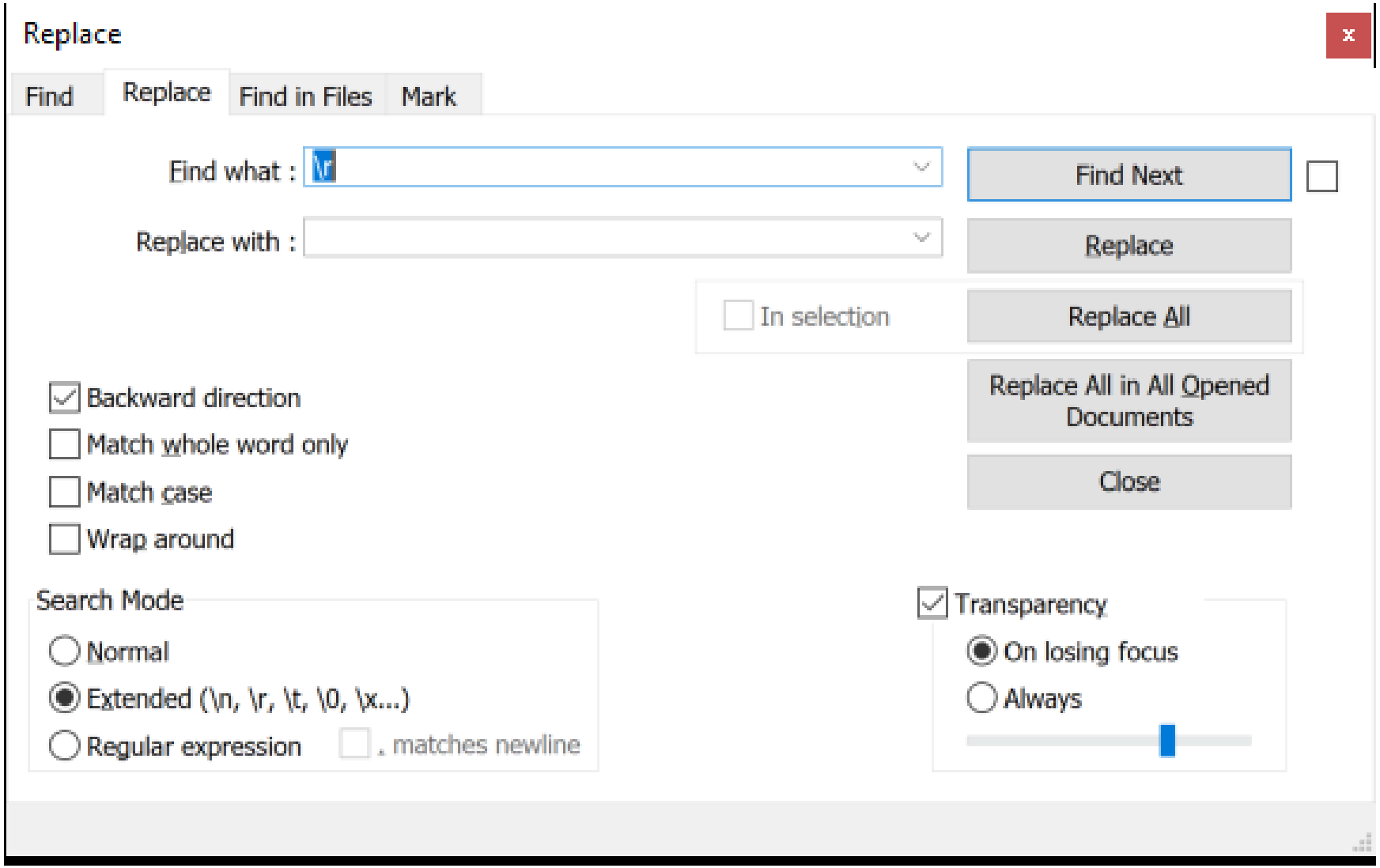
Line Terminators

- There are invisible characters at the end of every line in a text document
  - **Carriage Return (CR or `\r`)**
  - **Line Feed (LF or `\n`)**
- Unix, Linux, Mac systems end lines with LF
- Win systems end lines with CR LF
- Make sure your remove CR from files made in Windows
- This is one of many reasons why we use a Notepad++ or BBedit



Line Terminators

- There are invisible characters at the end of every line in a text document
  - **Carriage Return (CR or \r)**
  - **Line Feed (LF or \n)**
- Unix, Linux, Mac systems end lines with LF
- Win systems end lines with CR LF
- Make sure your remove CR from files made in Windows



Miscellaneous Useful Commands

- Note that some of the commands need to be installed on Macs

`history`	List the last commands you executed.[13]
`time [COMMAND]`	Time the execution of a command.
`wget [URL]`	Download the web page at `[URL]`.[14]
`open`	Open file or directory with default program; use `xdg-open` in Ubuntu or `start` in Windows Git Bash.
`rsync`	Synchronize files locally or remotely.
`tar` and `zip`	(Un)compress and package files and directories.
`awk` and `sed`	Powerful command-line text editors for much more complex text manipulation than `tr`.

13. In Git Bash all commands are listed.

14. Available in Ubuntu; for OS X look at `curl`, or install `wget` (see computingskillsforbiologists.com/wget).

`xargs`	Pass a list of arguments to other commands; for example, create a file for each line in `files.txt`: `cat files.txt | xargs touch`

$ Welcome to the Matrix

Questions?

Computational Biology

Lecture 1

Dr. Chris Bird