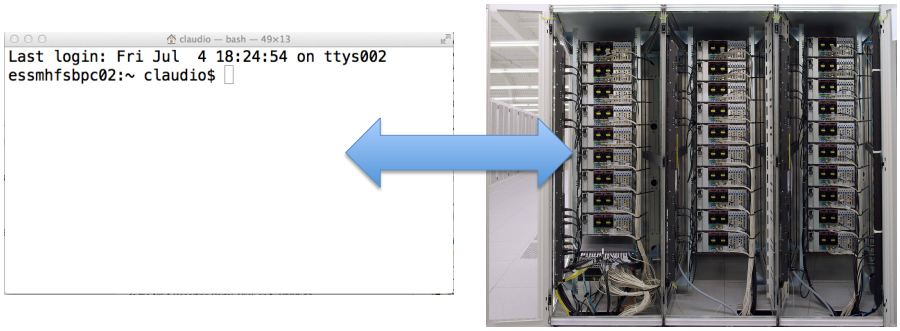
Unix, Linux and why you should care about them

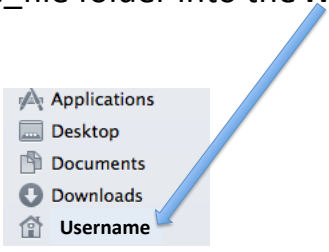


Claudio Casola

ECOSYSTEM SCIENCE AND MANAGEMENT
TEXAS A&M UNIVERSITY

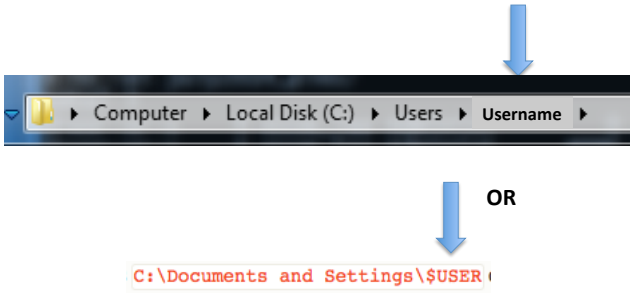
First things first

1. MAC: download the zipped folder pine_tree_files_MAC.zip, unzip it and move the pine_tree_file folder into the **Home directory**



First things first

1. PC: download the zipped folder pine_tree_files_PC.zip, unzip it and move the pine_tree_file folder into the **Home directory**



***nix** or **Unix-like**

1. Unix: Operating System (OS) developed in the AT&T's Bell Laboratories in the early '70s
2. Unix philosophy: OS characterized by a modular design, with a set of simple tools that each perform a limited, well-defined function
3. Unix-like systems are OSs that incorporate the Unix philosophy

What is an OS anyway?

- Coordinator and Traffic Cop:
  - Manages all resources
  - Settles conflicting requests for resources
  - Prevent errors and improper use of the computer
- Facilitator:
  - Provides facilities/services that everyone needs
  - Standard Libraries, Windowing systems
  - Make application programming easier, faster, less error-prone
- Some features reflect both tasks:
  - File system is needed by everyone (Facilitator) …
  - … but File system must be protected (Traffic Cop)

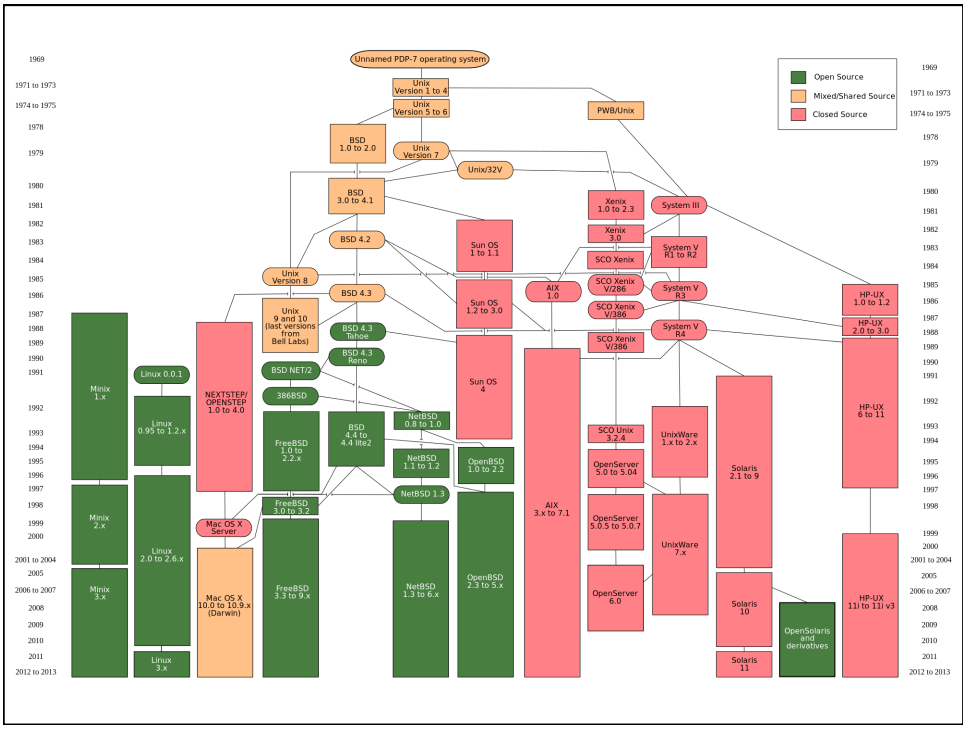
Adapted from A. D. Joseph and I. Stoica, CS162 – Copyright © UCB 2012

Unix and Linux

Linux was started in 1991 by Linus Torvalds, then a student at the University of Helsinki in Finland, because of his **dissatisfaction with MS-DOS** and his desire to obtain a **free version** of UNIX for his new computer

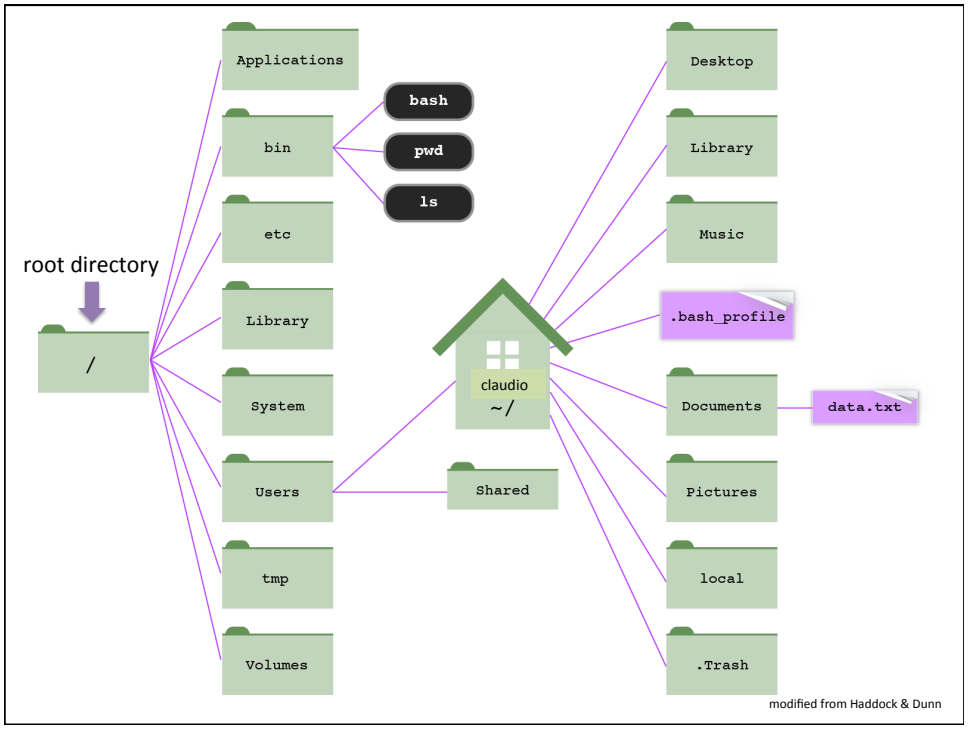
Linux quickly became a global project with programmers from around the world participating in its development via the Internet. As of June 2013, more than 95% of the world's 500 fastest supercomputers run some variant of Linux, including all the 44 fastest.

http://www.linfo.org/newbies.html and http://en.wikipedia.org/wiki/Linux



***nix** or **Unix-like**

1. Unix: Operating System (OS) developed in the AT&T's Bell Laboratories in the early '70s
2. Unix philosophy: **OS characterized by a modular design,** with a set of simple tools that each perform a limited, well-defined function
3. Unix-like systems are OSs that incorporate the Unix philosophy


root directory
/
Applications
bin
etc
Library
System
Users
tmp
Volumes
bash
pwd
ls
claudio
~/
Shared
Desktop
Library
Music
.bash_profile
Documents
data.txt
Pictures
local
.Trash
modified from Haddock & Dunn

PATHS

- A **path** is a written description of a location in the filesystem
- Both directories and files have paths
- Consists of directory names separated by forward slashes (/)
- **Absolute path** – complete and unambiguous location relative to the root; always starts with a forward slash (/)

```
/Users/Username/pine_tree_files/penelope.fa
```

- **Relative path** – describes where a file or folder is in relation to another folder, usually the working directory
- **Working directory** – where you're at right now; if you're in the home directory, the relative path to `penelope.fa` is:

```
pine_tree_files/penelope.fa
```

The Command Line

- Why not just use a Graphical User Interface?
  - Not good for long sequences of operations that need to be repeated on multiple datasets
  - No log of what you did and what commands you executed
  - Not conducive to executing jobs remotely on a cluster
- Use the command line!
  - Extremely, extremely powerful – NOT a primitive interface
  - Up-front investment with ENORMOUS long-term payoff
  - Absolute necessity for big datasets, which characterize modern biology
  - Automate tasks
  - Do and re-do and re-do and re-do analyses with minimal added effort
  - Easy to record what you did, minimizing mistakes while enhancing repeatability and troubleshooting

Files and Directories (Folders)

- Editors
  - Graphical text editors:
    - Gedit, nedit, xemacs, TextWrangler
  - Command-based
    - vi, emacs (powerful but not user friendly)
- File and Directory Names
  - Avoid spaces, rather underscore '_' and dash '-'
  - Avoid meta, special or reserved characters: ? ; ' " ! $ etc.
  - A file cannot have the same name as the directory where it resides

Choosing an Operating System

1. Mac OS X

Pros

- Unix is fully and naturally integrated into the OS
- Perl and Python pre-installed (you still need installing Bioperl and Biopyhton, which you should!)
- Can still use Microsoft Office, Endnote, etc.
- Easiest transition to make
- Lots of Mac-specific biological software (but not everything you might need!)
- Can install a virtual machine to run Windows-based software
- Can run all Linux-based software

Cons

- The hardware is expensive, and the OS is proprietary

Choosing an Operating System

2. Linux

Pros

- You can install it on any PC – a Linux box is a fraction of the price of a comparably outfitted Mac
- Compatible with all the important bioinformatics software
- People will think you're smart; you can snub Mac users
- Open source

Cons

- You've probably never used it; steeper learning curve
- Maintaining a Linux box requires some time investment

Choosing an Operating System

3. Microsoft Windows

Pros

- None

Cons

- Windows is not designed for programming and bioinformatics applications
- But there are some workarounds . . .

Running Linux alongside Windows

1. Run Linux within a virtual machine
   - https://www.virtualbox.org/wiki/Downloads
   - Install VirtualBox, launch it, click 'New' button, name the virtual machine, select the **Ubuntu** version of Linux, and allocate ≥512 Mb of memory
   - Install Ubuntu: http://www.ubuntu.com/
   - No need to restart the computer every time you run Linux
2. Boot into Linux from a flash drive
   - http://unetbootin.sourceforge.net/
   - Need to start and restart your computer to switch the OS
3. Install Cygwin (http://www.cygwin.com/)

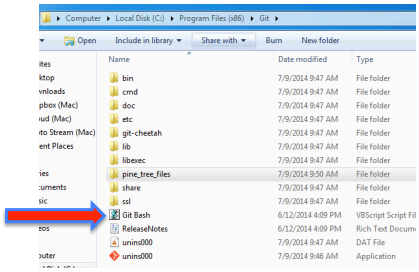
Open the Terminal

On a Mac

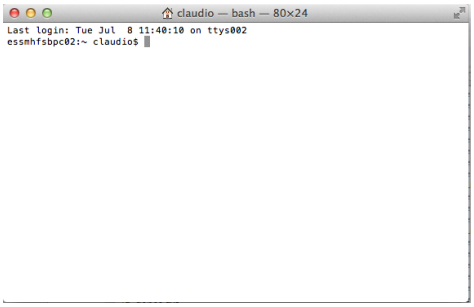
Finder > Go > Utilities > Terminal

On a PC

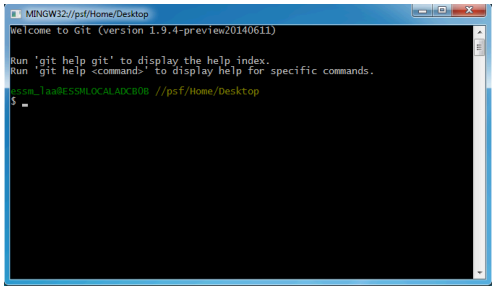
Double click on Git Bash



On a Mac



On a PC



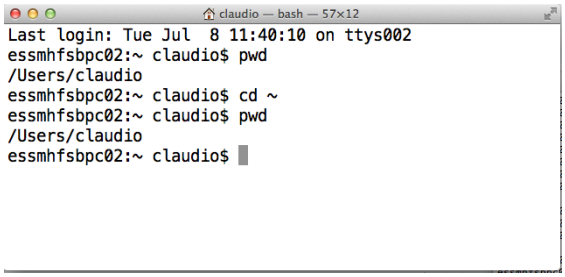
Navigating in Unix

$ `pwd` – prints path of working directory [absolute or relative?]

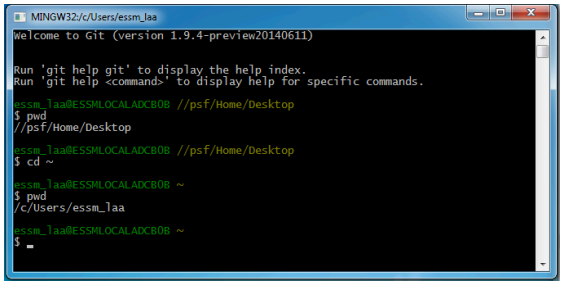
$ `cd` – change directory

- $ `cd ..` [move up one directory]
- $ `cd ../../` [move up two directories]
- $ `cd` [go home]
- $ `cd ~` [go home]
- $ `cd ~/pine_tree_files`
- $ `cd /Users/homedirectory/pine_tree_files`

On a Mac



On a PC



AWESOME SHORTCUTS

1. Auto-complete your commands with tab
2. Re-run a previous command with Up [and down] arrow
3. Find an old command with Ctrl+r
4. Find an old command with `history`

MORE SHORTCUTS

1. Ctrl+a – go to beginning of command
2. Ctrl+e – go to end of command
3. Ctrl+u – erase command from cursor to start
4. esc+b – jump backwards on command line
5. esc+f – jump forwards on command line

Viewing directory contents with `ls`

`-a` : list all directory contents, including hidden files
`-l` : list in long format
`-h` : make file sizes human readable (only with `-l`)
`-t` : sort by time modified (most recently modified first)
`-S` : sort files by size

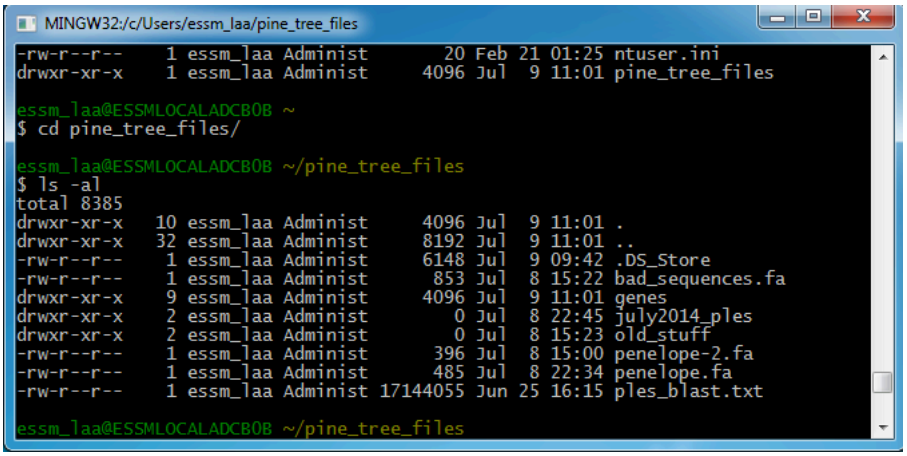
```
$ cd ~/pine_tree_files
$ ls -al
$ ls -alS
$ ls -alSh
```

ls output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```

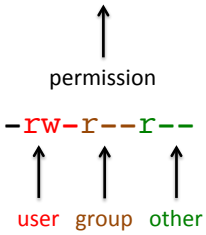
file (-) or directory (d)

ls output



ls output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```



r: read
w: write
x: execute

ls output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```

how many files under that directory

`ls` output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```

who's the owner

`ls` output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```


name of user group

`ls` output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```


file size

`ls` output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```


date/time last modified

`ls` output

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```


file name

Setting file permissions with `chmod`

`$ chmod -` ***ch****ange* ***mod****e*

`$ chmod ugo +/- rwx filename`

who give/take gets to do this

`$ chmod ugo+rwx *` ← be careful – don't assign execute permissions willy nilly

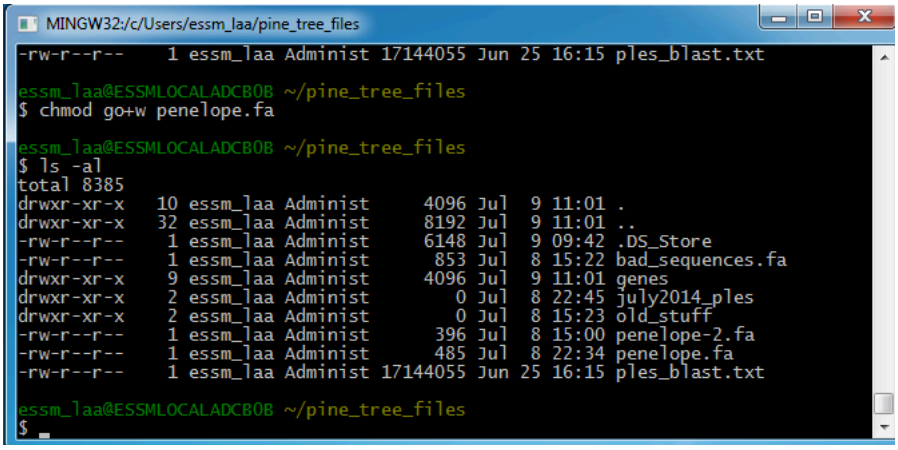
Setting file permissions with `chmod`

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-r--r--   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```

```
$ chmod go+w penelope.fa
```

```
total 624
drwxr-xr-x   8 claudio  staff     272 Jul  9 11:15 .
drwxr-xr-x+ 41 claudio  staff    1394 Jul  9 11:12 ..
-rw-r--r--@  1 claudio  staff    6148 Jul  9 11:13 .DS_Store
-rw-r--r--   1 claudio  staff     853 Jul  8 15:22 bad_sequences.fa
drwxr-xr-x   9 claudio  staff     306 Jul  9 11:01 genes
-rw-r--r--   1 claudio  staff     396 Jul  8 15:00 penelope-2.fa
-rw-rw-rw-   1 claudio  staff     485 Jul  8 22:34 penelope.fa
-rw-r--r--@  1 claudio  staff  296241 Jul  9 11:15 ples_blast.txt
```

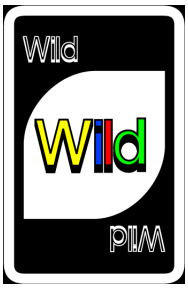
It doesn't work in Git!



* wildcards *

`*` the wildest of all wildcards, representing any number of any character (except a slash)

```
$ cd ~/pine_tree_files
$ ls *.fa
$ ls -al *.fa
```



Viewing files with `less`

```
$ less ples_blast.txt
```

- `/` to search
  - `n` for next occurrence
  - `N` for previous occurrence
- `space bar`, ↑, ↓, ←, → to navigate
- `g` to top of file
- `G` to bottom of file
- `q` to exit
- Lots of options
  - `-i` : ignore case when searching
  - `-S` : don't fold long lines
  - `-N` : show line numbers

Manual pages

Two main ways to find help:

1. Man pages contain comprehensive program usage and options
   `$ man program` [*IT DOESN'T WORK ON GIT!*]
   - navigate and search as you would with `less`
   - `q` to exit
   - generally straightforward, but not always
2. Google

Common Unix commands

`$ clear` – clear the Terminal screen

`$ exit` – exit the session
- always exit your session before closing the Terminal window and quitting Terminal
- not doing so is like unplugging your computer

Creating directories with `mkdir`

- $ `cd ~/pine_tree_files`

- Making directories

$ `mkdir directory-name ...` – make directories

```
$ mkdir old_stuff
$ ls –al
$ mkdir july2014
```

Copying and moving files

Moving and Copying Files

$ `cp source file target file` – copy files

```
$ cp penelope.fa july2014
$ ls -al july2014
$ cp penelope* july2014
$ ls -al july2014
$ ls -al
$ cp penelope.fa ~/Documents/
$ ls –al ~/Documents/
```

$ `mv source file target file` – move/rename files

```
$ mv bad_sequences.fa old_stuff
$ ls old_stuff
```

Deleting stuff

- Removing files

```
$ rm filename ... – remove files
  $ rm ~/Documents/penelope.fa
  $ ls -al ~/Documents/
```

- Removing files and directories recursively

```
  $ rm -r old_stuff
```

  - removes all the files in `old_stuff`, then removes `old_stuff` itself
  - dangerous command (that you'll use all the time)

```
  $ ls -al
```

Really important warnings

- Unix commands are permanent.
- There is no Recycle bin.
- There are usually no warnings.
- If you **overwrite** a file, it's gone forever.
- If you **delete** a file, it's gone forever.

Concatenate and Print files with `cat`

`$ cat` – concatenate and print files

```
$ cat july2014/penelope.fa july2014/penelope-2.fa
```

```
$ cat july2014/penelope*
```

Redirecting output

Use `>` to redirect output to a file instead of the screen.

Write to a file

```
$ cat july2014/penelope* > july2014/all_penelope.fa
```

Append to an existing file

```
$ cat july2014/penelope-2.fa >> july2014/all_penelope.fa
```

Redirecting output

If you redirect (single '>') to an existing file, it will completely **overwrite** it **without** warning.

Searching files with `grep`

`$ grep` – **g**lobally search a **r**egular **e**xpression and **p**rint (print lines in a file that match a pattern)

- EXTREMELY powerful – one of the more useful tools in the Unix bioinformatics toolbox

`$ grep pattern file`

```
$ grep '>' july2014/all_penelope.fa

$ grep '>' july2014/all_penelope.fa >july2014/
all_penelope_headers.txt
```

grep [options]

$ grep

- –c Show only a count of the results in the file
- –v Show only the lines that do not match
- –i Case insensitive matching
- –l List only the file names containing matches
- –n Show the line numbers of the match
- –h Hide the filenames in the output

Comparing data files with sort and join [Not in GIT-Windows!]

$ **join** - Joins the lines of two files which share a common field of data. The files need to be sorted first by the common field of data.

```
$ sort genes/coord_genes.txt >genes/
  coord_genes_sort.txt
$ sort genes/id_genes.txt >genes/id_genes_sort.txt
```

```
BFAR      1    chr16     14726667  14763093
CFC1B     1    chr2      131278666 131285565
POP7      1    chr7      100303675 100305123
DEDD2     1    chr19     42702744  42724304
ACTL6B    1    chr7      100240725 100254084
STARD7    1    chr2      96850602  96874573
```

```
RAD21L1   0    na   uc010gab.1
TFR2      0    na   uc003uvv.1
MOSPD3    0    na   uc003uvq.3
TUSC2     0    na   uc003czy.1
SLFNL1    0    na   uc001cgm.2
PIDD      0    na   uc001lrk.2
AK096982  0    na   uc021tfy.1
LOC613038 0    na   uc002dtc.1
…
…
```

Comparing data files with `sort` and `join`

$ **join** - Joins the lines of two files which share a common field of data. The files need to be sorted first by the common field of data.

```
$ sort genes/coord_genes.txt >genes/
  coord_genes_sort.txt
$ sort genes/id_genes.txt >genes/id_genes_sort.txt
```

BFAR	1	chr16	14726667	14763093
CFC1B	1	chr2	131278666	131285565
POP7	1	chr7	100303675	100305123
DEDD2	1	chr19	42702744	42724304
ACTL6B	1	chr7	100240725	100254084
STARD7	1	chr2	96850602	96874573

RAD21L1	0	na	uc010gab.1
TFR2	0	na	uc003uvv.1
MOSPD3	0	na	uc003uvq.3
TUSC2	0	na	uc003czy.1
SLFNL1	0	na	uc001cgm.2
PIDD	0	na	uc001lrk.2
AK096982	0	na	uc021tfy.1
LOC613038	0	na	uc002dtc.1
…			
…			

Comparing data files with `sort` and `join`

$ **join** - Joins the lines of two files which share a common field of data. The files need to be sorted first by the common field of data.

```
$ sort genes/coord_genes.txt >genes/
  coord_genes_sort.txt
$ sort genes/id_genes.txt >genes/id_genes_sort.txt
```

BFAR	1	chr16	14726667	14763093
CFC1B	1	chr2	131278666	131285565
POP7	1	chr7	100303675	100305123
DEDD2	1	chr19	42702744	42724304
ACTL6B	1	chr7	100240725	100254084
STARD7	1	chr2	96850602	96874573

RAD21L1	0	na	uc010gab.1
TFR2	0	na	uc003uvv.1
MOSPD3	0	na	uc003uvq.3
TUSC2	0	na	uc003czy.1
SLFNL1	0	na	uc001cgm.2
PIDD	0	na	uc001lrk.2
AK096982	0	na	uc021tfy.1
LOC613038	0	na	uc002dtc.1
…			
…			

Comparing data files with `sort` and `join`

```
NAME
     join -- relational database operator

SYNOPSIS
     join [-a file_number | -v file_number] [-e string]
          [-o list] [-t char] [-1 field] [-2 field] file1 file2

…
 -a file_number
           In addition to the default output, produce a line for
           each unpairable line in file file_number.

 -o list
       The -o option specifies the fields that will be output
       from each file for each line with matching join fields.
       Each element of list has the either the form
       `file_number.field', where file_number is a file number
       and field is a field number, or the form `0' (zero),
       representing the join field. The elements of list must be
       either comma (`,') or whitespace separated. (The latter
       requires quoting to protect it from the shell, or, a
       simpler approach is to use multiple -o options.)
```

Comparing data files with `sort` and `join`

```
$ cd genes/
$ join -o 1.1,1.3,1.4,1.5,2.4 coord_genes_sort.txt
id_genes_sort.txt >coord_id_genes.txt

ACTL6B    chr7 100240725 100254084 uc003uvy.3
BFAR      chr16 14726667 14763093 uc002dco.3
CFC1B     chr2 131278666 131285565 uc002trl.2
DEDD2     chr19 42702744 42724304 uc031rkz.1
POP7      chr7 100303675 100305123 uc003uwh.4
STARD7    chr2 96850602 96874573 uc002svm.4
```

Comparing data files with `sort` and `join`

```
$ join -a1 -o 1.1,1.4,2.3,2.4,2.5 id_genes_sort.txt
coord_genes_sort.txt >id_coord_genes.txt
```

```
ACTL6B    uc003uvy.3 chr7 100240725 100254084
AK096982  uc021tfy.1
APC2      uc002lsr.1
BFAR      uc002dco.3 chr16 14726667 14763093
CFC1B     uc002trl.2 chr2 131278666 131285565
DEDD2     uc031rkz.1 chr19 42702744 42724304
LOC613038 uc002dtc.1
LOC646743 uc002trm.4
MOSPD3    uc003uvq.3
NMB       uc002bkz.3
PIDD      uc001lrk.2
POP7      uc003uwh.4 chr7 100303675 100305123
…
…
```

Editing files with `sed`

`$ sed` – **s**tream **ed**itor; sed parses and transforms text

Replace characters in a file

```
$ sed 's/chr/CHR/g' coord_genes.txt >coord_genes-1.txt
```

Append characters on a specific line

```
$ cd ../july2014_ples
```

```
$ sed '/^>/ s/$/_1/g' penelope.fa >penelope-b.fa
```

Editing files with sed

$ sed – stream editor; sed parses and transforms text

Remove blank (empty) lines

```
$ sed '/^\s*$/d' penelope.fa >penelope-c.fa
```

PIPES

```
$ command [options] filename | command [options]
```

- Channel (or "pipe") STDOUT into another Unix command
- The first command has in input file, the next command takes the output of the previous command as its input
- No limit to the number of pipes per command

PIPES

```
# view directory contents with less
  $ ls -al | less -S

# find all the times I've used the pwd command
  $ history | grep pwd

# pipe previously shown commands together
  $ cp penelope* july2014 | cat penelope* | sed '/^\s*$/d'
    >july2014/all_penelope.fa
```

Creating workflows and pipelines

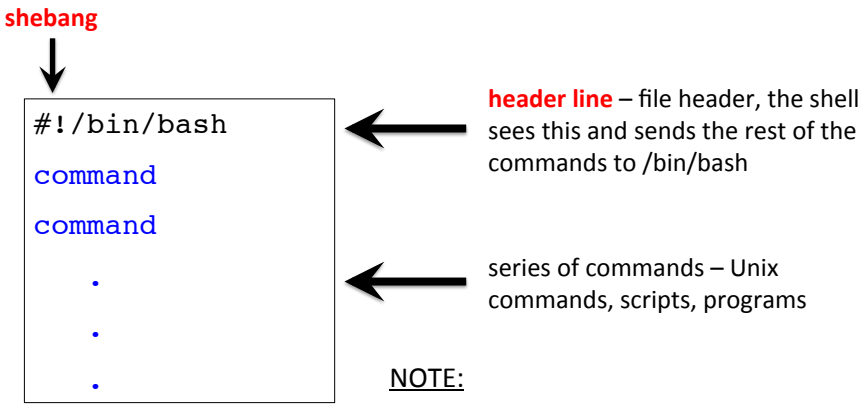
There are lots of ways to run multiple commands in sequence

1. Pipes
2. Semicolon-separated commands

   ```
   $ command;command;command
   ```

3. **Bash scripts**
   - an executable text file with a set of commands
   - in addition to automating your tasks, Bash scripts provide a permanent record of what you did*

Bash scripts



NOTE:

- The file must be executable
- Execute from the command line with:

```
$ ./bash_script1.sh
```

R on Terminal (MAC)

Open R:

```
$ R
```

Exit R:

```
>q()
```

R on Terminal (Git-PC)

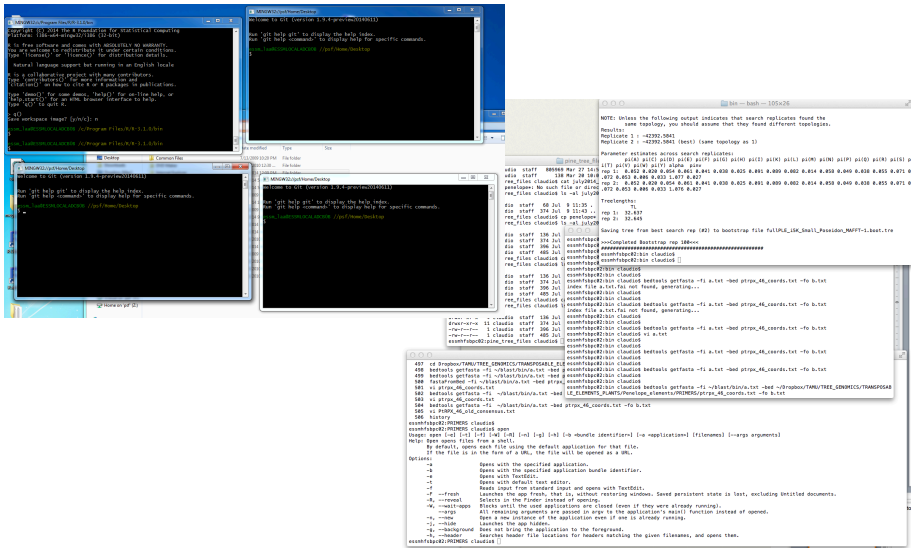
Open R:

```
$ cd /c/Program\ Files/R/R-3.1.0/bin/
$ R.exe
```

Exit R:

```
>q()
```

You can run multiple tasks in Terminal and R



Make your own *nix commands cookbook

Sometimes it will take hours of research on the internet and many, many attempts on Terminal to find the right combination of commands. Then, you may need those again a month or a year later. If you'll keep a small cookbook of the commands you seldom use, for example in a text file (not in a word file!), it will save you a lot of time in the future

Credits

Andrew J. Alverson – Assistant Professor
Department of Biological Sciences
University of Arkansas

Common File System Commands

`ls`	List names of all files in current directory
`ls` *filenames*	List only the named files
`ls -t`	List in time order, most recent first
`ls -l`	Long listing, more information. Also `ls -lt`
`ls -t`	List by time last used. Also `ls -lu`, `ls -lut`
`ls -r`	List in reverse order. Also `ls -rt`, `ls -rlt`, etc
`ed` *filename*	Edit named file
`cp` *file1 file2*	Copy *file1 file2*. Overwrite old *file2* if it exists
`mv` *file1 file2*	Move *file1 file2*. Overwrite old *file2* if it exists
`rm` *filenames*	Remove named files, irrevocably
`cat` *filenames*	Print contents of named files
`pr` *filenames*	Print content with header, 66 lines per pages (default)
`pr -`*n* *filenames*	Print in *n* columns
`pr -m` *filenames*	Print named files side by side in multiple columns
`wc` *filenames*	Count lines, words, and characters for each file
`wc -l` *filenames*	Count lines for each file
`grep` *pattern filenames*	Print lines matching *pattern*
`grep -v` *pattern filenames*	Print lines not matching *pattern*
`sort` *filenames*	Sort files alphabetically by line
`tail` *filename*	Print last 10 lines of file
`tail -`*n filename*	Print last *n* lines of file
`tail +`*n filename*	Start printing file at line *n*
`cmp` *file1 file2*	Print location of first difference
`diff` *file1 file2*	Print all differences between files

Shell Metacharacters

`>`	*prog* `>` *file* direct standard output to *file*
`>>`	*prog* `>>` *file* append standard output to *file*
`<`	*prog* `<` *file* take standard input from *file*
`|`	p_1 `|` p_2 connect standard output of p_1 to standard input of p_2
`<<here`	*here document*: standard input follows, up to next `here` on a line by itself
`*`	Match any string of zero or more characters in filenames
`?`	Match any single character in filenames
`[`*ccc*`]`	Match any single character from `[`*ccc*`]` in filenames. Ranges like `0-9` or `a-z` are legal
`;`	Command terminator: p_1 `;` p_2 does p_1, then p_2
`&`	Like `;` but does not wait for p_1 to finish
`` `…` ``	Run command(s) in … ; output replaces `` `…` ``
`(…)`	Run command(s) in … in a sub-shell
`{…}`	Run command(s) in … in current shell (rarely used)
`$1`, `$2`, *etc*	`$0` … `$9` replaced by arguments to shell file
`$`*var*	Value of shell variable *var*
`${`*var*`}`	Value of *var*; avoids confusion when concatenated with text
`\`	`\`*c* take character *c* literally, `\`*newline* discarded
‘…’	Take … literally
”…”	Take … literally after `$`, `` `…` `` and `\` interpreted
#	Text after # is a comment
var=*value*	Assign *value* to variable *var*
p_1 `&&` p_2	Run p_1; if successful, run p_2
p_1 `||` p_2	Run p_1; if unsuccessful, run p_2

Shell I/O Redirections

> *file*	direct standard output to *file*
>> *file*	append standard output to *file*
< *file*	take standard input from *file*
$p_1 \mid p_2$	connect standard output of program p_1 to input of p_2
^	obsolete synonym for |
n>*file*	direct output from file descriptor *n* to *file*
n>>*file*	append output from file descriptor *n* to *file*
n>&*m*	merge output from file descriptor *n* with file descriptor *m*
n<&*m*	merge input from file descriptor *n* with file descriptor *m*
<<*s*	here document: take standard input until next *s* at beginning of a line; substitute for `$`, `` `…` ``, and `\`
<<*s*	here document with no substitution
<< '*s*'	here document with no substitution

`grep` and `egrep` Regular Expressions
(decreasing order of precedence)

c	any non-special character *c* matches itself
c	turn off any special meaning of character *c*
^	beginning of line
$	end of line
.	any single character
[…]	any one of characters in …; ranges like a-z are legal
[^…]	any single character not in …; ranges are legal
n	what the *n*'th \(…\) matched (grep only)
*r**	zero or more occurrences of *r*
r+	one or more occurrences of *r* (egrep only)
r?	zero or more occurrences of *r* (egrep only)
$r_1 r_2$	r_1 followed by r_2
$r_1 \mid r_2$	r_1 or r_2 (egrep only)
\(*r*\)	tagged regular expression *r* (grep only); can be nested
(*r*)	regular expression *r* (egrep only); can be nested

No regular expression matches a newline.

Shell Built-in Variables

`$#`	the number of arguments
`$*`	all arguments to shell. "`$*`" is a single word
`$@`	similar to `$*`. "`$@`" is identical to the list of the arguments to shell
`$-`	options supplied to the shell
`$?`	return value of the last command executed
`$$`	process-id of the shell
`$!`	process-id of the last command started with `&`
`$HOME`	default argument for `cd` command
`$IFS`	list of characters that separate words in arguments
`$MAIL`	file that, when changed, triggers "you have mail" message
`$PATH`	list of directories to search for commands
`$PS1`	prompt string, default '`$` '
`$PS2`	prompt string for continued command line, default '`>` '

Shell Pattern Matching Rules

`*`	match any string, including the null string
`?`	match any single character
`[`*ccc*`]`	match any of the characters in *ccc* `[a-d0-3]` is equivalent to `[abcd0123]`
”…”	match … exactly; quotes protect special characters. Also ’…’
`\`*c*	match *c* literally
`a|b`	in `case` espressions only, matches either `a` or `b`
`/`	in filenames, matched only by an explicit `/` in the expression; in `case`, matched like any other character
`.`	as the first character of a filename, is matched only by an explicit `.` in the expression