

Introduction To UNIX Shell



2. class-material (bash)

```
# Print Working Directory: a.k.a. where the hell am I? This is a comment line
> pwd This is our first UNIX command :-)
```

Don't type the ">" bit it is the "shell prompt"!

```
# List out the files and directories where you are
> ls
```

Do it Yourself!

Q. What do you see after each command?

Q. Does it make sense if you compare to your Mac: *Finder* or Windows: *File Explorer*?

```
# On Mac only :(
> open .
# Note the [SPACE] is important
```

2. class-material (bash)

```
# Download any file to your current directory/folder
> curl -O https://bioboot.github.io/bggn213_S18/class-material/bggn213_01_unix.zip
```

Q. Does what you see at each step make sense if you compare to your Mac: *Finder* or Windows: *File Explorer*?

2. class-material (bash)

```
# Download any file to your current directory/folder
> curl -O https://bioboot.github.io/bggn213_S18/class-material/bggn213_01_unix.zip

# List out the files and directories where you are (NB: Use TAB for auto-complete)
> ls bggn213_01_unix.zip

# Un-zip your downloaded file
> unzip bggn213_01_unix.zip

# Change directory (i.e. move to the folder named bggn213_01_unix)
> cd bggn213_01_unix

> ls
> pwd
```

Q. Does what you see at each step make sense if you compare to your Mac: *Finder* or Windows: *File Explorer*?

2. class-material (bash)

```
# Practice moving around the file system...
> cd projects
> ls
> pwd
> cd ..
```

Motivation

Why do we use Unix?

Modularity	Core programs are modular and work well with others
Programmability	Best software development environment
Infrastructure	Access to existing tools and cutting-edge methods
Reliability	Unparalleled uptime and stability
Unix Philosophy	Encourages open standards

Modularity	Core programs are modular and work well with others
Programmability	Best software development environment
Infrastructure	Access to existing tools and cutting-edge methods
Reliability	Unparalleled uptime and stability
Unix Philosophy	Encourages open standards

Modularity

The Unix shell was designed to allow users to easily build complex workflows by interfacing smaller **modular programs** together.

wget → awk → grep → sort → uniq → plot

An alternative approach is to write a **single complex program** that takes raw data as input, and after hours of data processing, outputs publication figures and a final table of results.

All-in-one custom 'Monster' program

Advantages/Disadvantages

The 'monster approach' is **customized to a particular project** but results in **massive, fragile** and difficult to modify (therefore **inflexible, untransferable, and error prone**) code.

With **modular workflows**, it's easier to:

- **Spot errors** and figure out where they're occurring by inspecting intermediate results.
- **Experiment** with alternative methods by swapping out components.
- **Tackle novel problems** by remixing existing modular tools.

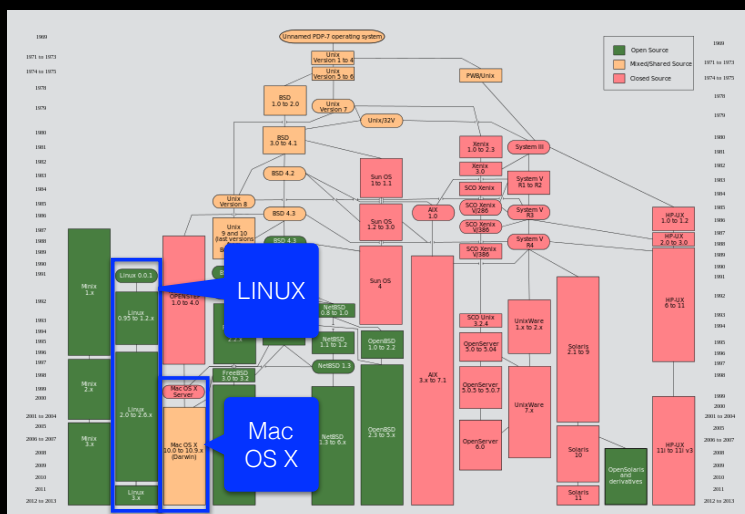
Unix 'Philosophy'

"Write programs that do one thing and do it well. Write programs to work together and that encourage open standards. Write programs to handle text streams, because that is a universal interface."



— Doug McIlroy

Unix **family** tree [1969-2010]



Source: https://commons.wikimedia.org/wiki/File:Unix_history-simple.svg

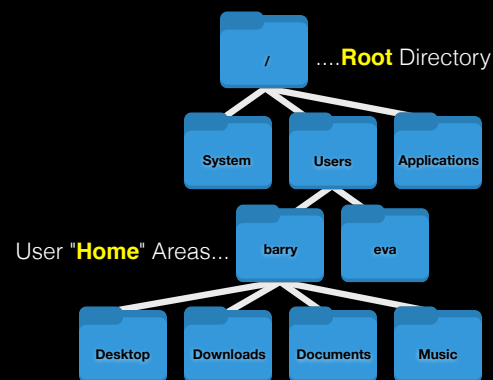
Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	curl	grep	top
cd	cp	head	chmod	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	echo	sudo	Ctl-c
ssh	 (pipe)	touch	source	git	Ctl-z
scp	> (write to file)		cat	R	bg
	< (read from file)		tmux	python	fg

Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	curl	grep	top
cd	cp	head	chmod	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	echo	sudo	Ctrl-c
ssh	 (pipe)	touch	source	git	Ctrl-z
scp	> (write to file)		cat	R	bg
	< (read from file)		tmux	python	fg

Beginning Unix

Getting started with basic Unix commands

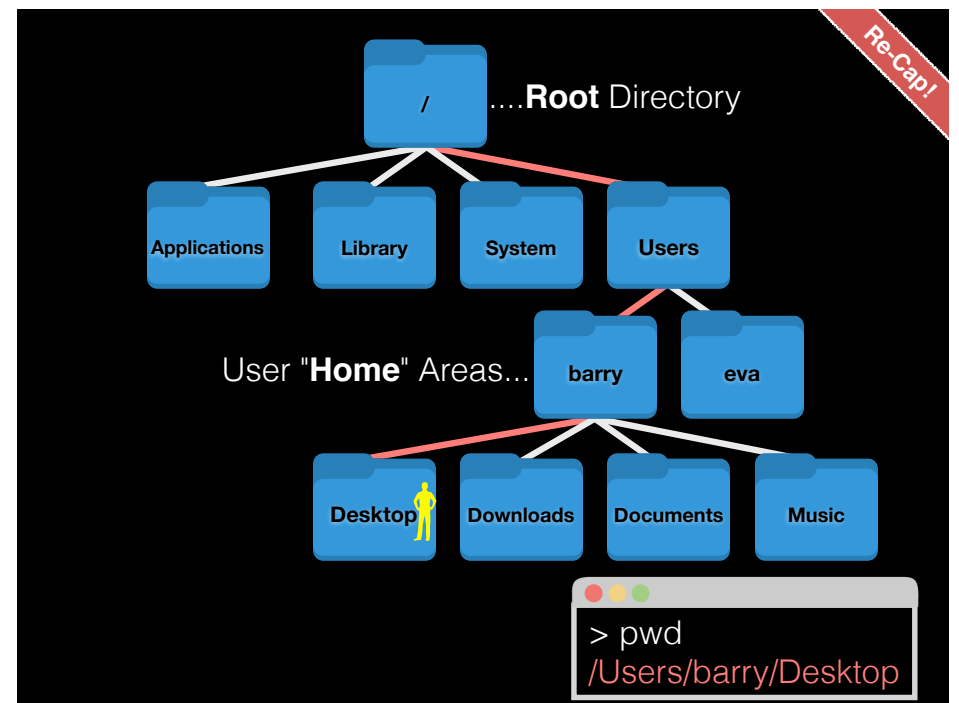
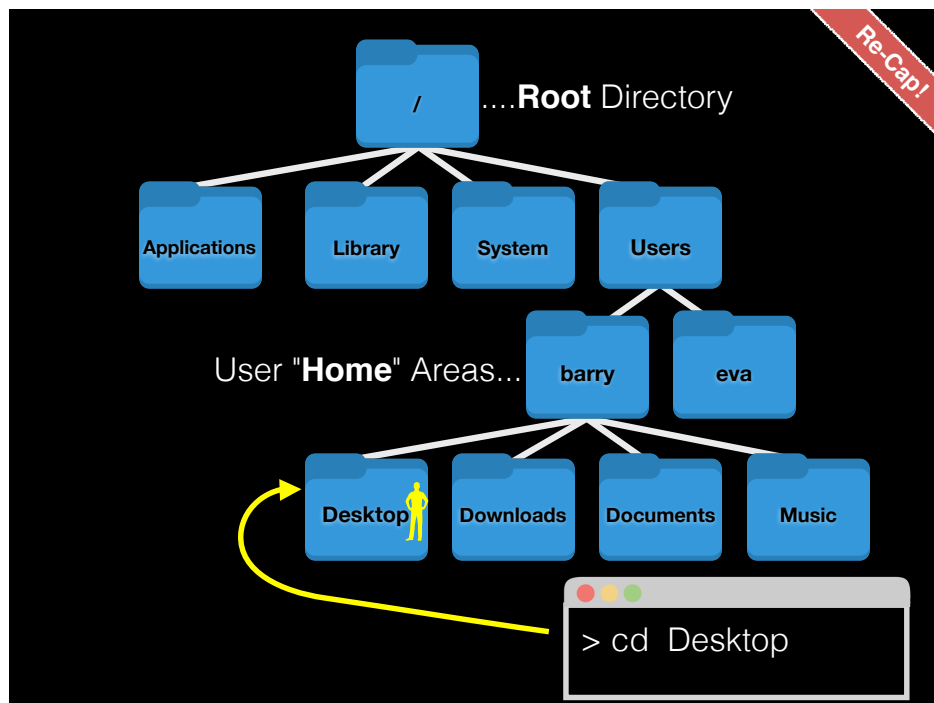
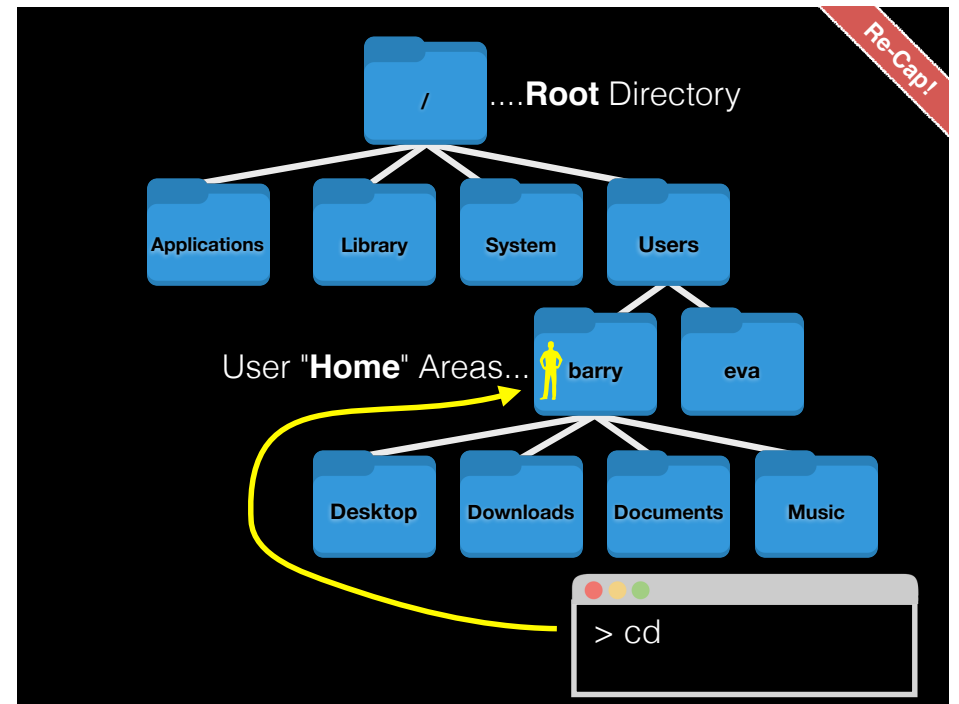
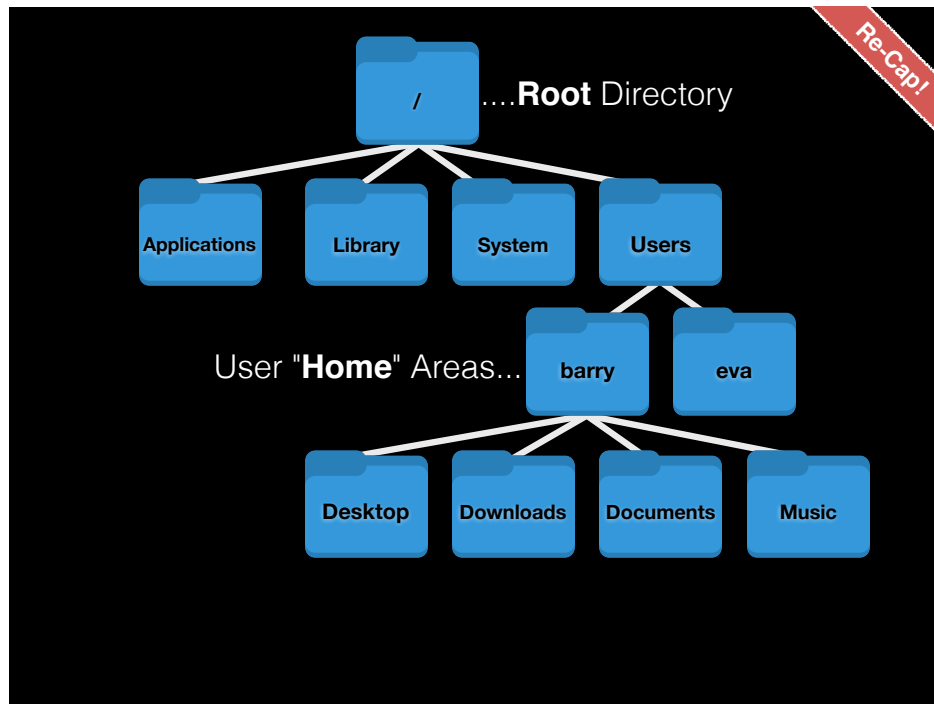
Information in the **file system** is stored in files, which are stored in **directories** (folders). Directories can also store other directories, which forms a **directory tree**.

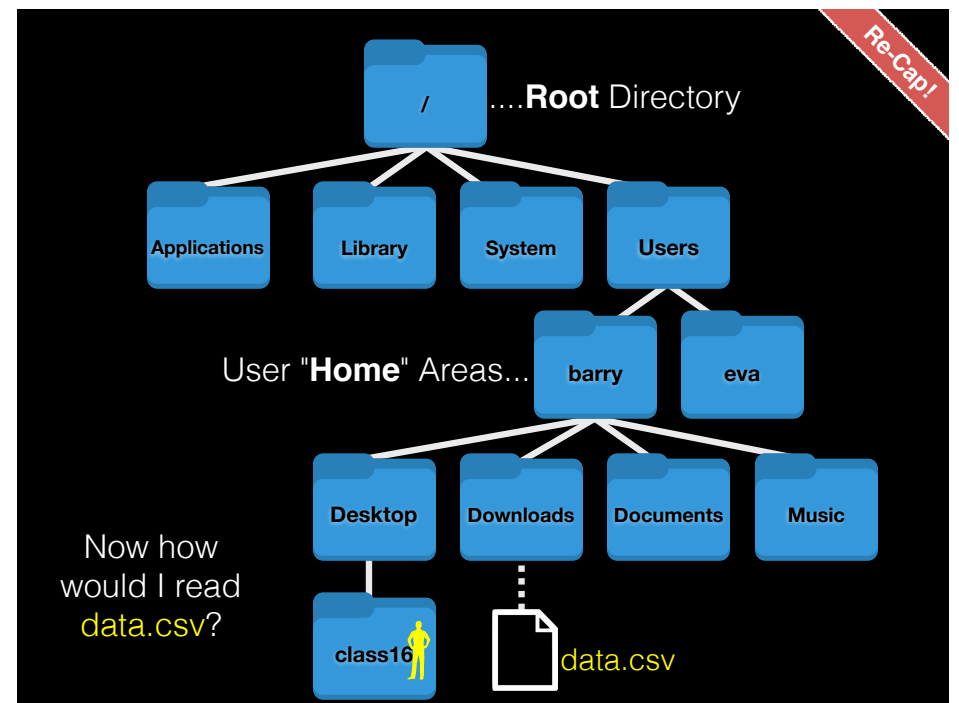
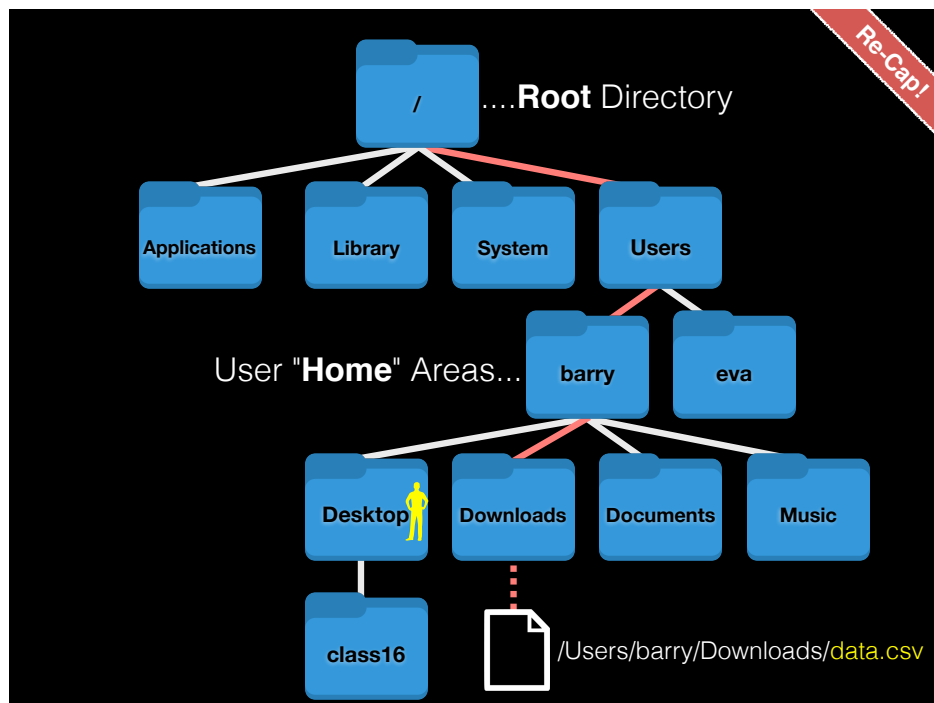
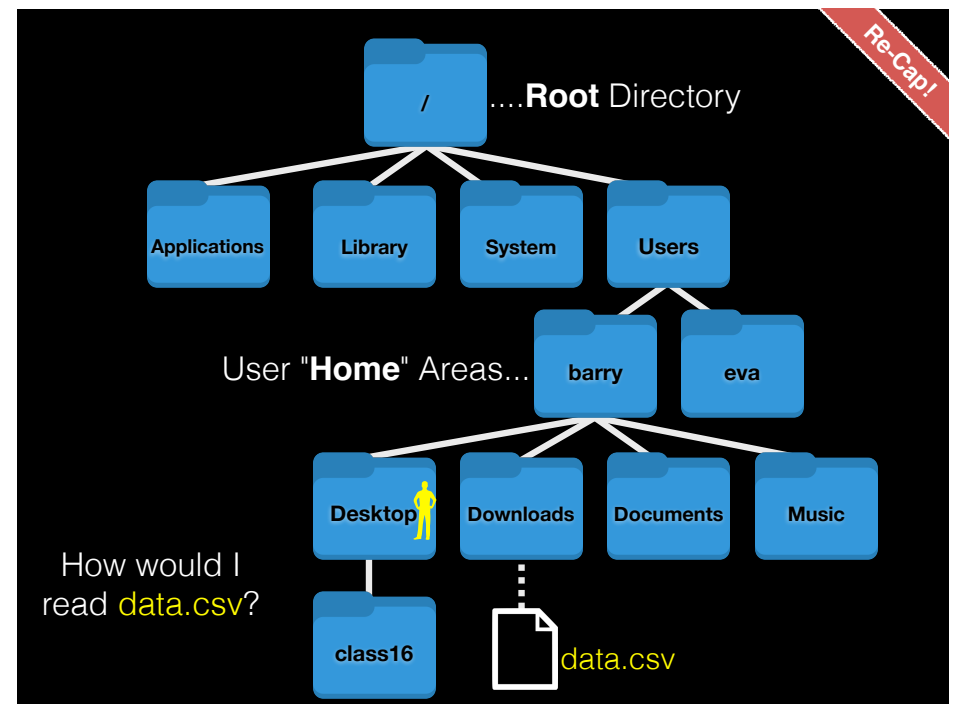
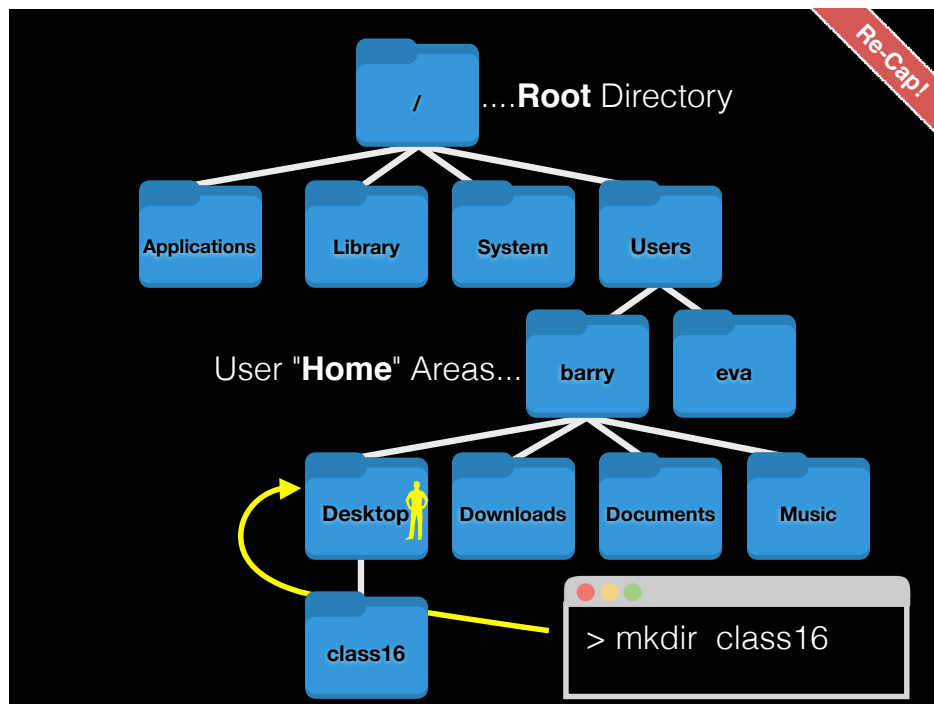


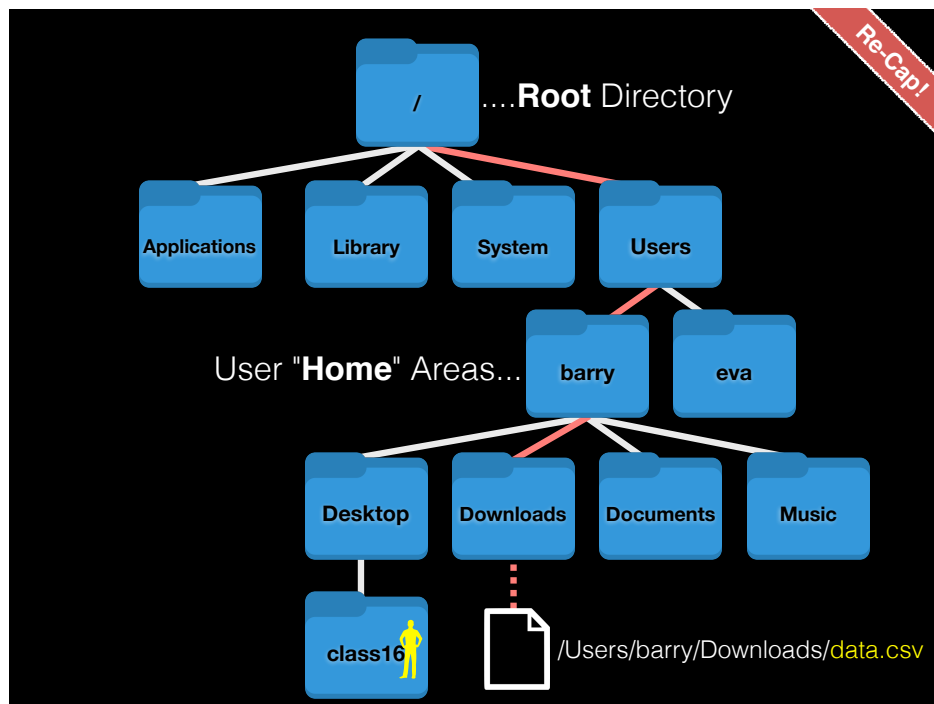
The forward slash character **'/'** is used to represent the **root** directory of the whole file system, and is also used to separate directory names.
E.g. **/Users/barry/Desktop/myfile.csv**

UNIX Basics: Using the filesystem

ls	List files and directories
cd	Change directory (i.e. move to a different 'folder')
pwd	Print working directory (which folder are you in)
mkdir	MaKe a new DIRectories
cp	CoPy a file or directory to somewhere else
mv	MoVe a file or directory (basically rename)
rm	ReMove a file or directory







Side Note: File Paths

- An **absolute path** specifies a location from the root of the file system. E.g. `/Users/barry/Downloads/data.csv`
`~/Downloads/data.csv`
- A **relative path** specifies a location starting from the current location. E.g. `../data.csv`

.	Single dot '.' (for current directory)
..	Double dot '..' (for parent directory, back one dir in our tree)
~	Tilda '~' (shortcut for your home directory)
[Tab]	Pressing the tab key can autocomplete names

Finding the Right Hammer (**man** and **apropos**)

- You can access the manual (i.e. user documentation) on a command with **man**, e.g:

> man pwd

- The man page is only helpful if you know the name of the command you're looking for. **apropos** will search the man pages for keywords.

> apropos "working directory"

Windows only: > pwd --help

Not on Windows :-(

Inspecting text files

- **less** - visualize a text file:
 - use arrow keys
 - page down/page up with "space"/"b" keys
 - search by typing "/"
 - **quit** by typing "q"
- Also see: **head**, **tail**, **cat**, **more**

Creating text files

Creating files can be done in a few ways:

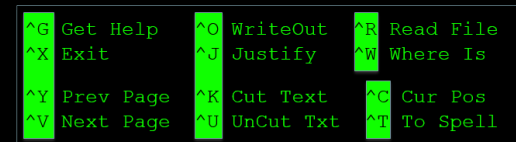
- With a **text editor** (such as **nano**, **emacs**, or **vi**)
- With the **touch** command (`> touch a_file`)
- From the command line with **cat** or **echo** and **redirection** (more on this later)
- **nano** is a simple text editor that is recommended for first-time users. Other text editors have more powerful features but also steep learning curves

Creating and editing text files with **nano**

Do it Yourself!

In the terminal type:

```
> nano yourfilename.txt
```



[^] - Press Control

- There are many other text file editors (e.g. **vim**, **emacs** and **sublime text**, **atom** etc.)

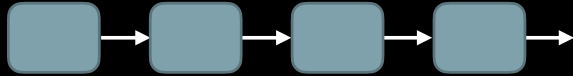
Working with Unix

How do we actually use Unix?

Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	chmod	grep	top
cd	cp	head	echo	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	curl	sudo	Ctrl-c
ssh	 (pipe)	touch	source	git	Ctrl-z
	> (write to file)		cat	R	bg
	< (read from file)			python	fg

Combining Utilities with **Redirection** (>, <) and **Pipes** (|)

- The power of the shell lies in the ability to combine simple utilities (*i.e.* commands) into more complex algorithms very quickly.



- A key element of this is the ability to send the output from one command into a file or to pass it directly to another program.
- This is the job of >, < and |

Side-Note: Standard Input and Standard Output streams

Two very important concepts that unpin Unix workflows:

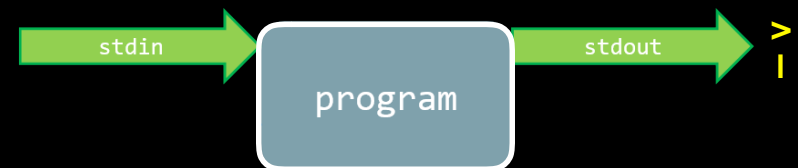
- Standard Output (**stdout**) - default destination of a program's output. It is generally the terminal screen.
- Standard Input (**stdin**) - default source of a program's input. It is generally the command line.

Output redirection and piping



> ls ~/Desktop # stdin is "~/Desktop"; stdout to screen

Output redirection and piping



> ls ~/Desktop # stdin is "~/Desktop"; stdout to screen

> ls ~/Desktop > mylist.txt # stdout **redirected** to file

> ls ~/Desktop | less # stdout **piped** to less (no file created)

Output redirection and piping



- > ls ~/Desktop # stdin is "~/Desktop"; stdout to screen
- > ls ~/Desktop > mylist.txt # stdout **redirected** to file
- > ls ~/Desktop | less # stdout **pipelined** to less (no file created)
- > ls -l ~/Desktop # extra optional input **argument** "-l"

Output redirection and piping



- > ls ~/Desktop # stdin is "~/Desktop"; stdout to screen
- > ls ~/Desktop > mylist.txt # stdout **redirected** to file
- > ls ~/Desktop | less # stdout **pipelined** to less (no file created)
- > ls /nodirexists/ 2> binlist.txt # stderr to **file**

Output redirection and piping



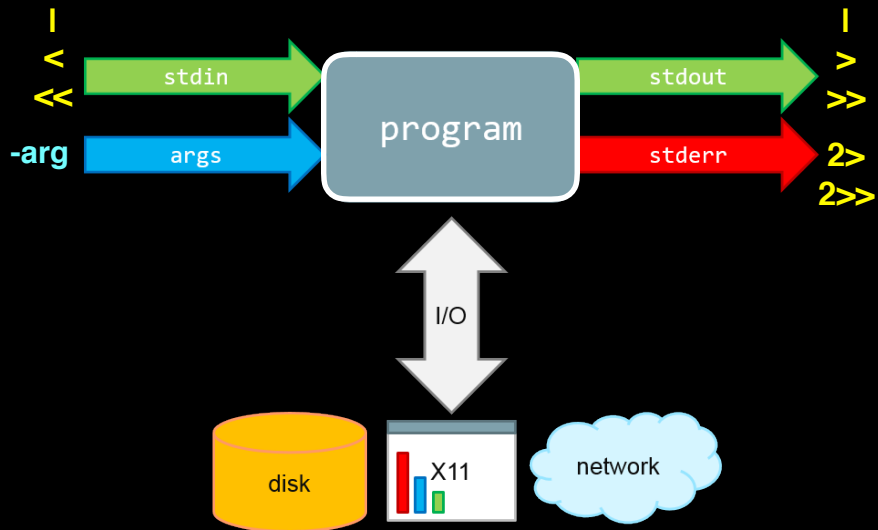
- > ls ~/Desktop # stdin is "~/Desktop"; stdout to screen
- > ls ~/Desktop > mylist.txt # stdout **redirected** to file
- > ls ~/Desktop | less # stdout **pipelined** to less (no file created)
- > ls /nodirexists/ 2> binlist.txt # stderr to **file**

Output redirection and piping

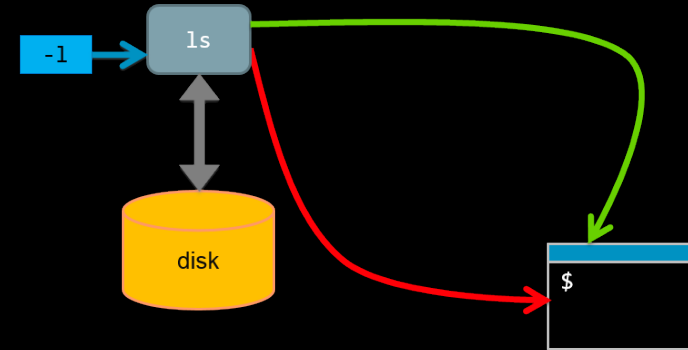


- > ls ~/Desktop # stdin is "~/Desktop"; stdout to screen
- > ls ~/Desktop > mylist.txt # stdout **redirected** to file
- > ls ~/Desktop | less # stdout **pipelined** to less (no file created)
- > ls /nodirexists/ 2> binlist.txt # stderr to **file**

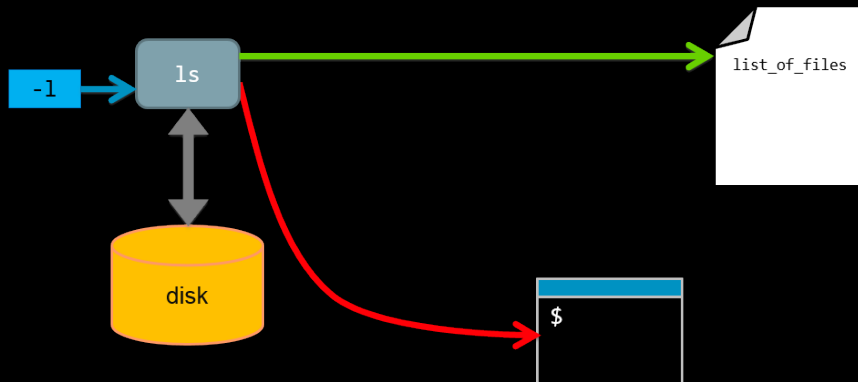
Output redirection summary



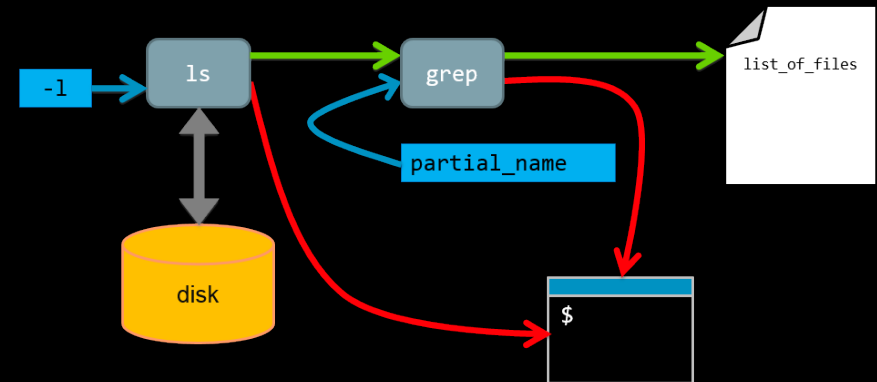
ls -l



ls -l > list_of_files

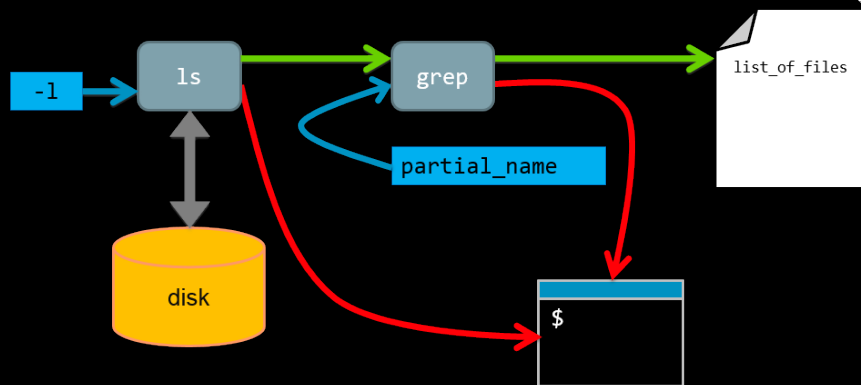


ls -l | grep partial_name > list_of_files



We have **piped** (|) the **stdout** of one command into the **stdin** of another command!

ls -l /usr/bin/ | grep "tree" > list_of_files



grep: prints lines containing a string.
Also searches for strings in text files.

Do it Yourself!

Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	chmod	grep	top
cd	cp	head	echo	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	curl	sudo	Ctrl-c
ssh	 (pipe)	touch	source	git	Ctrl-z
scp	> (write to file)		cat	R	bg
	< (read from file)		tmux	python	fg

Side-Note: **grep** 'power command'

- grep** - prints lines containing a string pattern. Also searches for strings in text files, e.g.

> cd ~/bggn213_01_unix/projects/transducin/sequences/data/

> grep --color "GESGKS" seqdump.fasta

REVKLLLLGAG**GESGKS**TIVKQMKIIHEAGYSEEECKQYK

- grep is a 'power tool' that is often used with pipes as it accepts **regular expressions** as input (e.g. "**G..GK[ST]**") and has lots of useful options - see the [man page](#) for details.

Do it Yourself!

grep example using regular expressions

- Suppose a program that you are working with complains that your input sequence file contains non-nucleotide characters. You can eye-ball your file or ...

> grep -v "^>" seqdump.fasta | grep --color "[^ATGC]"

Exercises:

- Use "man grep" (or on windows "grep --help") to find out what the -v argument option is doing!
- How could we also show line number for each match along with the output?
(tip you can grep the output of "man grep" for 'line number')

Do it Yourself!

grep example using regular expressions

Do it Yourself!

- Suppose a program that you are working with complains that your input sequence file contains non-nucleotide characters. You can eye-ball your file or ...

```
> grep -v "^>" seqdump.fasta | grep --color -n "[^ATGC]"
```

- First we remove (with **-v** option) lines that start with a ">" character (these are sequence identifiers).
- Next we find characters that are *not* A, T, C or G. To do this we use **^** symbols second meaning: *match anything but* the pattern in square brackets. We also print line number (with **-n** option) and color output (with **--color** option).

Key Point: Pipes and redirects avoid unnecessary i/o

- Disc i/o is often a bottleneck in data processing!
- Pipes prevent unnecessary disc i/o operations by connecting the stdout of one process to the stdin of another (these are frequently called "**streams**")

```
> program1 input.txt 2> program1.stderr | \
program2 2> program2.stderr > results.txt
```

- Pipes and redirects allow us to build solutions from modular parts that work with **stdin** and **stdout streams**.

Unix 'Philosophy' Revisited

"Write programs that do one thing and do it well. Write programs to work together and that encourage open standards. Write programs to handle text streams, because that is a universal interface."

— Doug McIlroy



Pipes provide speed, flexibility and sometimes simplicity...

- In 1986 "*Communications of the ACM magazine*" asked famous computer scientist Donald Knuth to write a simple program to count and print the *k* most common words in a file alongside their counts, in descending order.
- Knuth wrote a literate programming solution that was 7 pages long, and also highly customized to this problem (e.g. Knuth implemented a custom data structure for counting English words).
- Doug McIlroy replied with one line:

```
> cat input.txt | tr A-Z a-z | sort | uniq -c | sort -rn | sed 10q
```

Key Point:

You can chain any number of programs together to achieve your goal!



This allows you to build up fairly complex workflows within one command-line.

Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	chmod	grep	top
cd	cp	head	echo	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	curl	sudo	Ctrl-c
ssh	 (pipe)	touch	source	git	Ctrl-z
	> (write to file)		cat	R	bg
	< (read from file)			python	fg

Hands-on time

Using AWS for Bioinformatics

- Using remote UNIX machines (Part I)
- Using remote UNIX machines (Part II)
- Using remote UNIX machines (Part III)

~75 mins

Do it Yourself!

New commands

sudo	Execute a command with root permissions
apt-get	Package handling utility for updating & installing software
curl	Download data
gunzip	File compression and decompression
blastp	Command line BLAST
shmlast	Mapping orthologs from RNA-seq data
?	Use man to find out about other new commands

Connecting to remote machines (with **ssh**)

Most high-performance computing (HPC) resources can only be accessed by **ssh** (Secure **S**hell)

```
> ssh [user@host.address]
```

For example:

```
> ssh barry@bio3d.ucsd.edu
      User      Host address
```

Connecting to AWS (with **ssh**)

- We will go through this process together in the last hands-on sections today!

✓ Downloaded our class specific **keyfile** (See web link on board or your email: This is required for connecting to AWS virtual machines later.)

✓ Use your **username** and assigned **IP_ADDRESS**

```
> ssh -i ~/Downloads/ssh_key ubuntu@IP_ADDRESS
      Optional key file      User      Host address
```

Copying to and from remote machines (**scp**)

- The **scp** (Secure CoPy) command can be used to copy files and directories from one computer to another.

```
> scp [file] [user@host]:[destination]
> scp localfile.txt barry@bigcomputer.net:/remotedir/.
```

Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	chmod	grep	top
cd	cp	head	echo	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	curl	sudo	Ctrl-c
ssh	 (pipe)	touch	source	git	Ctrl-z
scp	> (write to file)		cat	R	bg
	< (read from file)		tmux	python	fg

Process refers to a running instance of a program

top	Provides a real-time view of all running processes
ps	Report a snapshot of the current processes
kill	Terminate a process (the “force quit” of the unix world)
Ctrl-c	Stop a job
Ctrl-z	Suspend a job
bg	Resume a suspended job in the background
fg	Resume a suspended job in the foreground
&	Start a job in the background

Do it Yourself!

Hands-on time

Sections 1 to 3 of software carpentry UNIX lesson

<https://swcarpentry.github.io/shell-novice/>

Do This at Home!

<https://explainshell.com>

- Many bioinformatics tools can only be used through a command line interface, or have extra capabilities in the command line version that are not available in the GUI.
- The shell makes your work less error-prone, more reproducible and less boring allowing you to automate repetitive tasks and concentrate on more exciting things.
- Many bioinformatic tasks require large amounts of computing power and can't realistically be run on your own machine. These tasks are best performed using **remote computers** or **cloud computing**, which can only be accessed through a shell.

How to Get Working

Best practices for organizing your
computational biology projects

Read: Noble PLoS Comp Biol (2009)

- "A Quick Guide to Organizing Computational Biology Projects"
<http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000424>

All files and directories used in your project should live in a single project directory.

- Use sub-directories to divide your project into sub-projects.
- Do not use spaces in file and directory names!

Document your methods and workflows with plain text README files

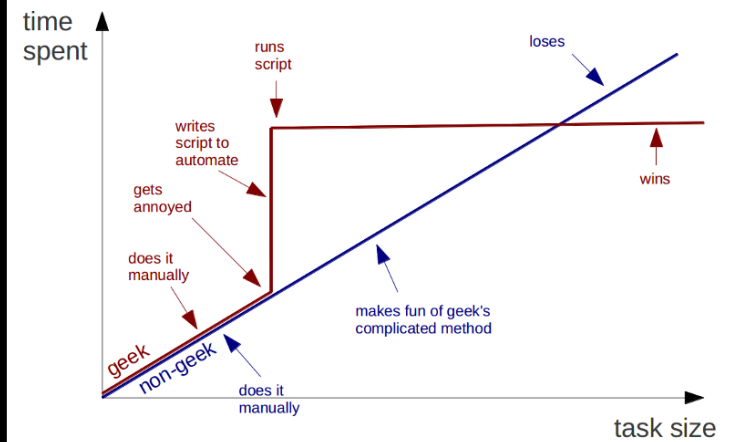
- Also document the origin of all data in your project directory
- Also document the versions of the software that you ran and the options you used.
- Consider using Markdown for your documentation.

Use version control and backup to multiple destinations!

Be reproducible:

<http://ropensci.github.io/reproducibility-guide/sections/introduction/>

Geeks and repetitive tasks



Summary

- Built-in unix shell commands allow for easy data manipulation (e.g. sort, grep, etc.)
- Commands can be easily combined to generate flexible solutions to data manipulation tasks.
- The unix shell allows users to automate repetitive tasks through the use of shell scripts that promote reproducibility and easy troubleshooting
- Introduced the 21 key unix commands that you will use during ~95% of your future unix work...