BGGN 213

Introduction to UNIX

Barry Grant

UC San Diego

http://thegrantlab.org/bggn213

Recap From Last Time:

- Substitution matrices: Where our alignment match and mis-match scores typically come from
- Comparing methods: The trade-off between *sensitivity*, *selectivity* and *performance*
- Sequence motifs and patterns: Finding functional cues from conservation patterns
- Sequence profiles and position specific scoring matrices (PSSMs), Building and searching with profiles, Their advantages and limitations
- PSI-BLAST algorithm: Application of iterative PSSM searching to improve BLAST sensitivity
- Hidden Markov models (HMMs): More versatile probabilistic model for detection of remote similarities

Todays Menu

	Time	Topics
I	9:00-9:15 AM	**Setup and Motivation**
II	9:15-10:00 AM	**Beginning Unix**
	10:00-10:40 AM	Hands on session
III	10:40-11:00 AM	**Working with Unix**
IV	11:00-11:20 PM	**How to Get Working**
	11:20-11:45 PM	Hands on session

https://bioboot.github.io/bggn213_f17/setup/

Setup Checklist

Problems?

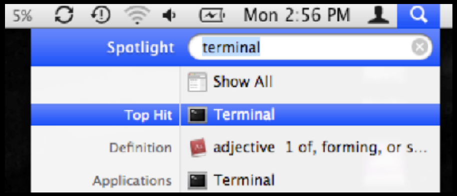
https://bioboot.github.io/bggn213_f17/setup/

- ☑ **Mac**: Terminal *or* **PC**: MobIXterm
- ☑ **PC**: MobIXterm CygUtils plugin (core UNIX tools)
- ☑ Downloaded our class specific jetstream keyfile (See email: This is required for connecting to jetstream virtual machines with your terminal.)
- ☑ Example data downloaded: bggn213_01_unix.zip

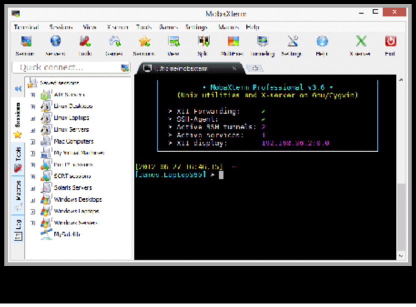
Lets get started…

Do it Yourself!

Mac
Terminal



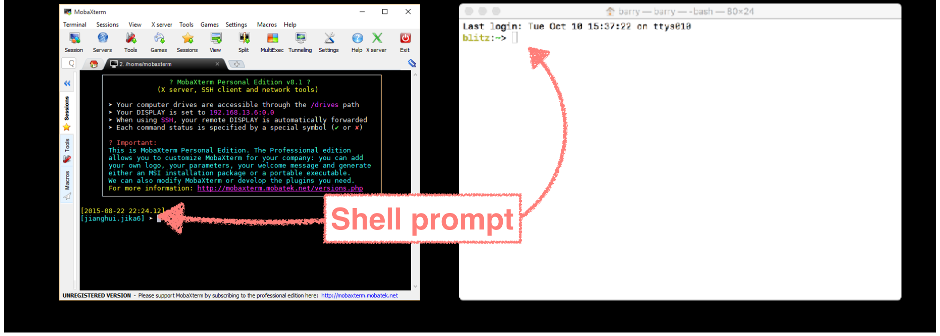
PC
MobaXterm



```
      • MobaXterm Professional v3.6 •
  (Unix utilities and X-server on Gnu/Cygwin)

 > X11 Forwarding:     ✓
 > SSH-Agent:          ✓
 > Active SSH tunnels: 2
 > Active services:    1
 > X11 display:        192.168.36.2:0.0

[2012 06 27 16:46:15]
[James.Laptop389] >
```

SideNote: Terminal *vs* Shell

- **Shell**: A command-line interface that allows a user to interact with the operating system by typing commands.
- **Terminal** [emulator]: A graphical interface to the shell (*i.e.* the window you get when you launch MobaXterm).



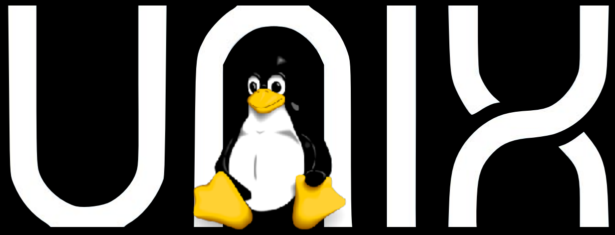
```
          ? MobaXterm Personal Edition v8.1 ?
        (X server, SSH client and network tools)

 ➤ Your computer drives are accessible through the /drives path
 ➤ Your DISPLAY is set to 192.168.13.6:0.0
 ➤ When using SSH, your remote DISPLAY is automatically forwarded
 ➤ Each command status is specified by a special symbol (✔ or ✘)

 ? Important:
 This is MobaXterm Personal Edition. The Professional edition
 allows you to customize MobaXterm for your company: you can add
 your own logo, your parameters, your welcome message and generate
 either an MSI installation package or a portable executable.
 We can also modify MobaXterm or develop the plugins you need.
 For more information: http://mobaxterm.mobatek.net/versions.php

[2015-08-22 22:24.12]
[jianghui.jika6] ➤
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

```
Last login: Tue Oct 10 15:37:22 on ttys010
blitz:~>
```

Shell prompt

Introduction To



Shell

Motivation

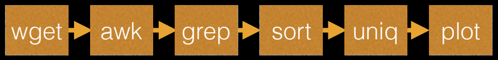
Why do we use Unix?

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

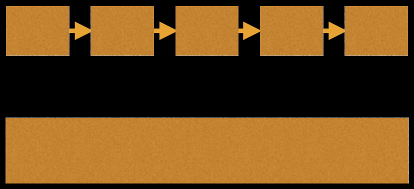
The Unix shell was designed to allow users to easily build complex workflows by interfacing smaller **modular programs** together.



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

Which would you prefer and why?



Modular

vs

Custom

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:

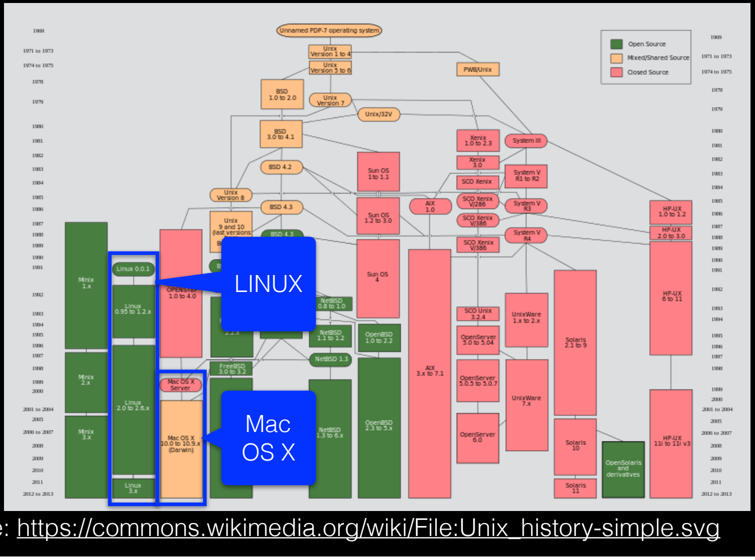
- Spot errors and figure out where they're occurring by inspecting intermediate results.
- Experiment with alternative methods by swapping out components.
- Tackle novel problems by remixing existing modular tools.

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 McIlory

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



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

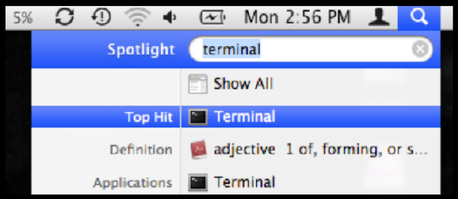
Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	chmod	grep	top
cd	cp	head	echo	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	curl	sudo	Crl-c
ssh	| (pipe)	touch	source	git	Crl-z
scp	> (write to file)		cat	R	bg
	< (read from file)		tmux	python	fg

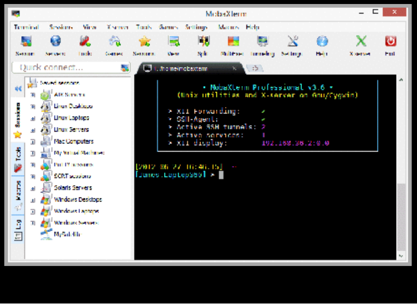
Basics	File Control	Viewing & Editing Files	Misc. useful	Power commands	Process related
ls	mv	less	chmod	grep	top
cd	cp	head	echo	find	ps
pwd	mkdir	tail	wc	sed	kill
man	rm	nano	curl	sudo	Crl-c
ssh	| (pipe)	touch	source	git	Crl-z
scp	> (write to file)		cat	R	bg
	< (read from file)		tmux	python	fg

Lets get started…

Do it Yourself!

Mac
Terminal





PC
MobaXterm

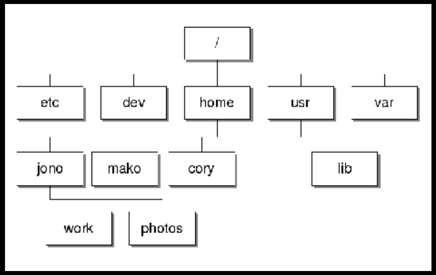
Beginning Unix

Getting started with basic Unix commands

Download the example data and mv to your Desktop
bggn213_01_unix.zip

File System Structure

- 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. **/home/jono/work/bggn213_notes.txt**

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. **/home/jono/work/bggn213_notes.txt**
- A relative path specifies a location starting from the current location. E.g. **../bggn213_notes.txt**

.	Single dot '**.**' (for current directory)
..	Double dot '**..**' (for parent directory)
~	Tilda '**~**' (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"
```

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)

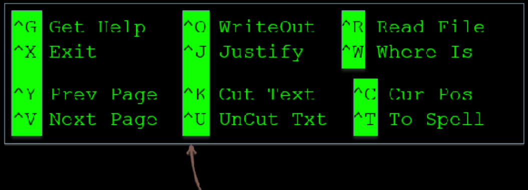
- **nano** is a simple text editor that is recommended for first-time users. Other text editors have more powerful features but also steep learning curves

Creating and editing text files with **nano**

Do it Yourself!

In the terminal type:

```
> nano yourfilename.txt
```



```
^G Get Help    ^O WriteOut    ^R Read File
^X Exit        ^J Justify     ^W Where Is

^Y Prev Page   ^K Cut Text    ^C Cur Pos
^V Next Page   ^U UnCut Txt   ^T To Spell
```

^ - Press Control

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

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

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

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

For example:

```
> ssh barry@bio3d.ucsd.edu
> ssh tb170077@IP_ADDRESS
```

Connecting to jetstream (with **ssh**)

- First we have to login online and fire-up a new “virtual machine”

- Full instructions here: https://bioboot.github.io/bggn213_f17/jetstream/boot/

- We will go through this process in the last section today!

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  bgrant@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	Crl-c
ssh	| (pipe)	touch	source	git	Crl-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)
Crl-c	Stop a job
Crl-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/

~20 mins

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	Crl-c
ssh	**|** (pipe)	touch	source	git	Crl-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 |

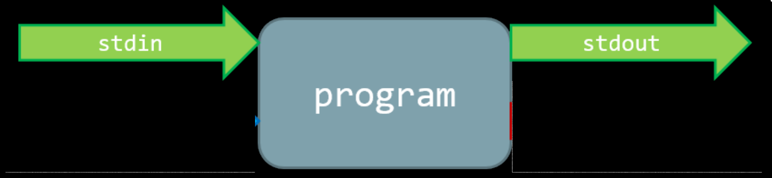
Side-Note: **Standard Input** and **Standard Output** streams

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.

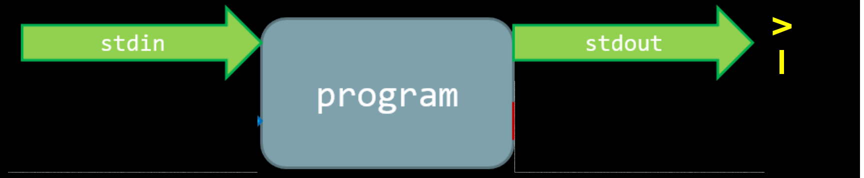
Output redirection and piping

Do it Yourself!



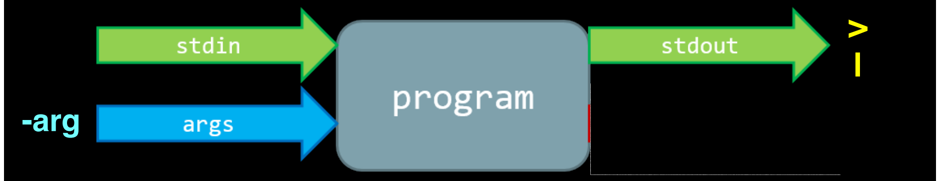
```
> ls /usr/bin  # stdin is “/usr/bin”; stdout to screen
```

Output redirection and piping



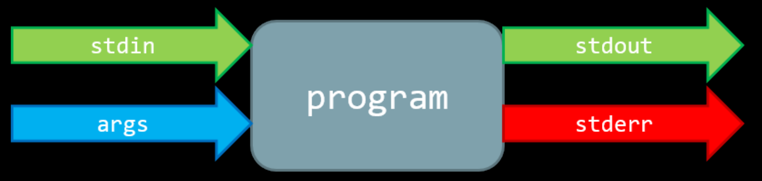
```
> ls /usr/bin  # stdin is “/usr/bin”; stdout to screen

> ls /usr/bin > binlist.txt  # stdout redirected to file

 > ls /usr/bin | less  # sdout piped to less (no file created)
```

Output redirection and piping

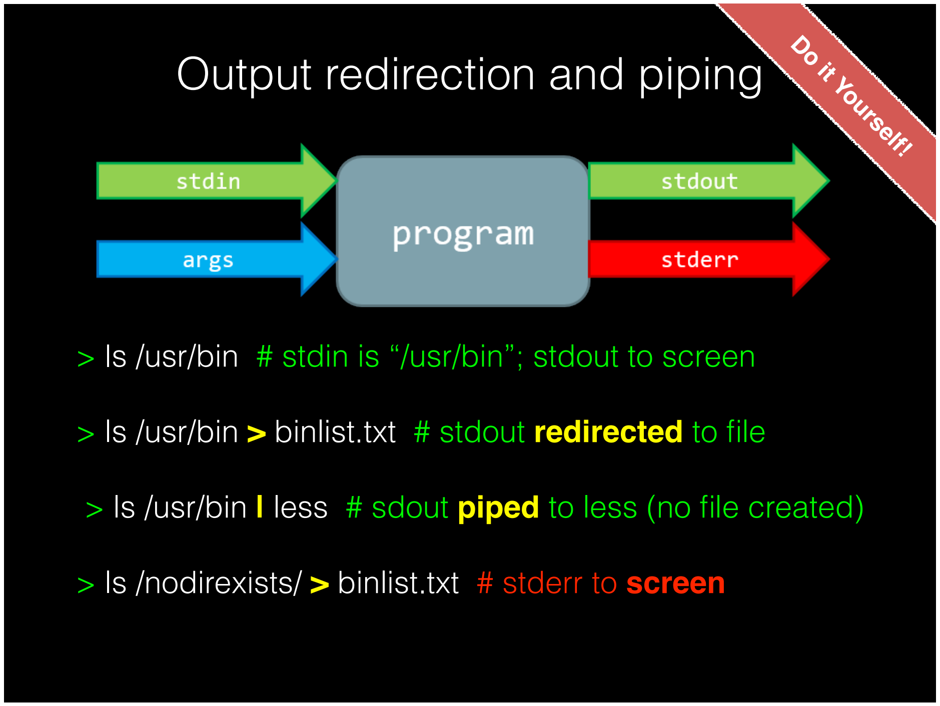


```
> ls /usr/bin  # stdin is “/usr/bin”; stdout to screen

> ls /usr/bin > binlist.txt  # stdout redirected to file

 > ls /usr/bin | less  # sdout piped to less (no file created)

> ls -l /usr/bin  # extra optional input argument “-l”
```

Output redirection and piping

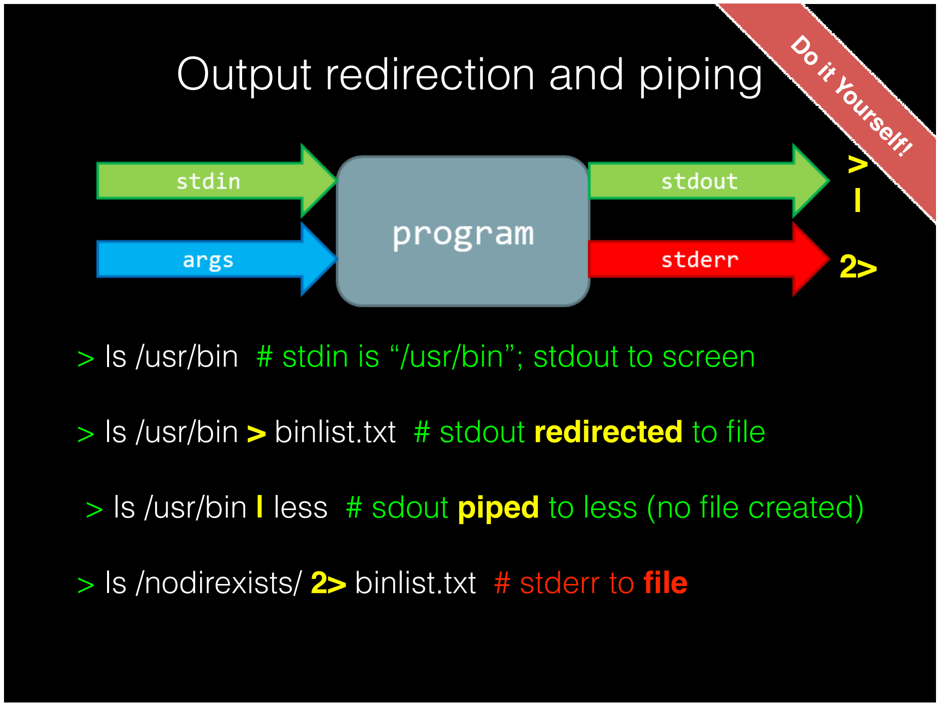


```
> ls /usr/bin  # stdin is “/usr/bin”; stdout to screen

> ls /usr/bin > binlist.txt  # stdout redirected to file

 > ls /usr/bin | less  # sdout piped to less (no file created)

> ls /nodirexists/   # stderr to screen
```



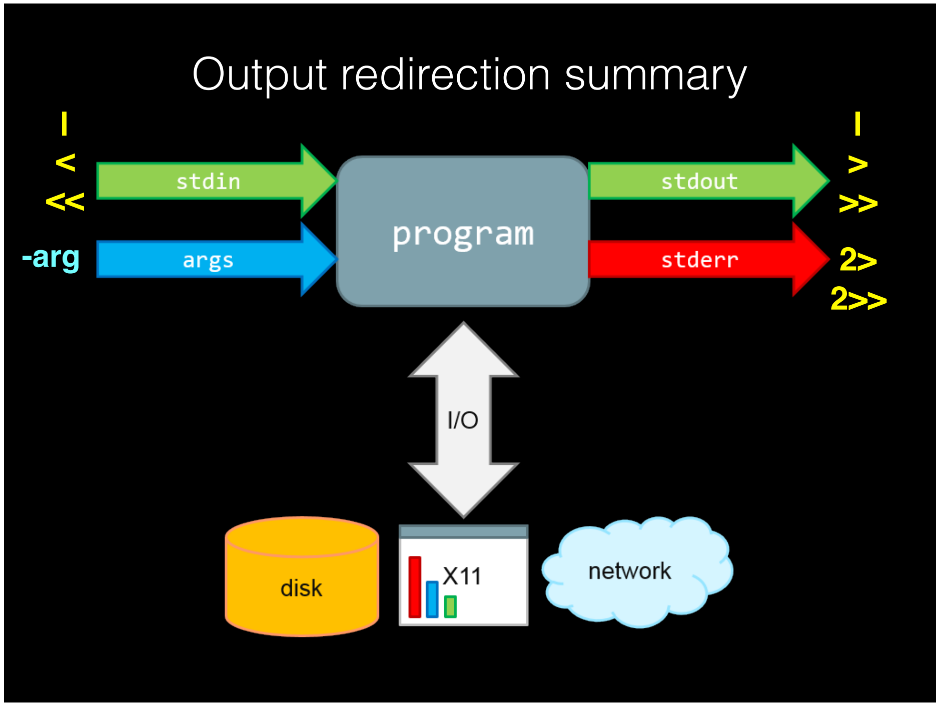
Output redirection and piping

```
> ls /usr/bin  # stdin is “/usr/bin”; stdout to screen

> ls /usr/bin > binlist.txt  # stdout redirected to file

> ls /usr/bin | less  # sdout piped to less (no file created)

> ls /nodirexists/ > binlist.txt  # stderr to screen
```

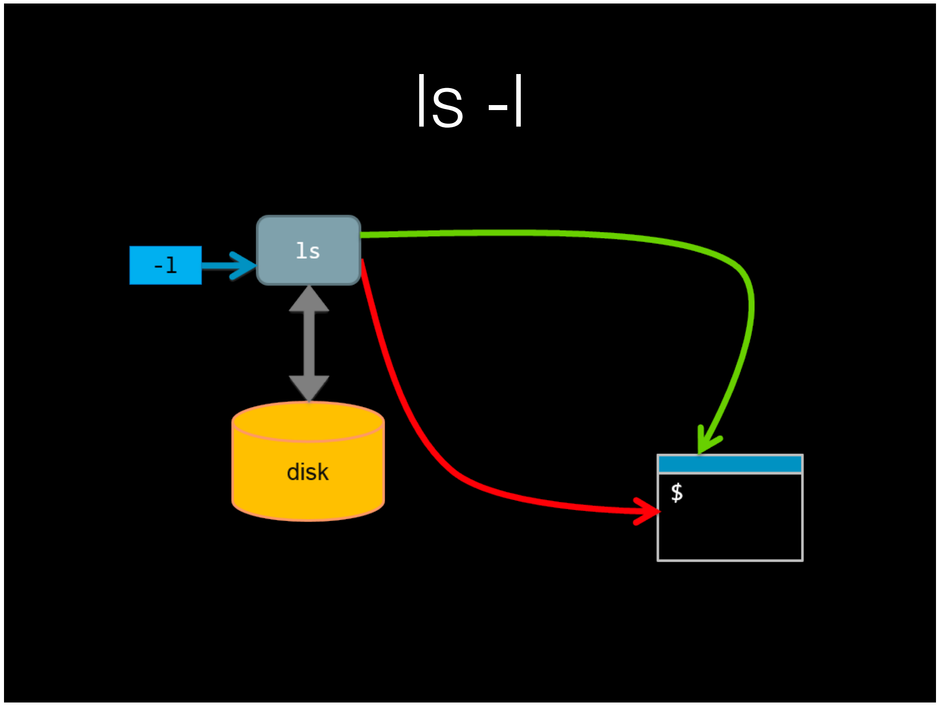


Output redirection and piping

```
> ls /usr/bin  # stdin is “/usr/bin”; stdout to screen

> ls /usr/bin > binlist.txt  # stdout redirected to file

> ls /usr/bin | less  # sdout piped to less (no file created)

> ls /nodirexists/ 2> binlist.txt  # stderr to file
```

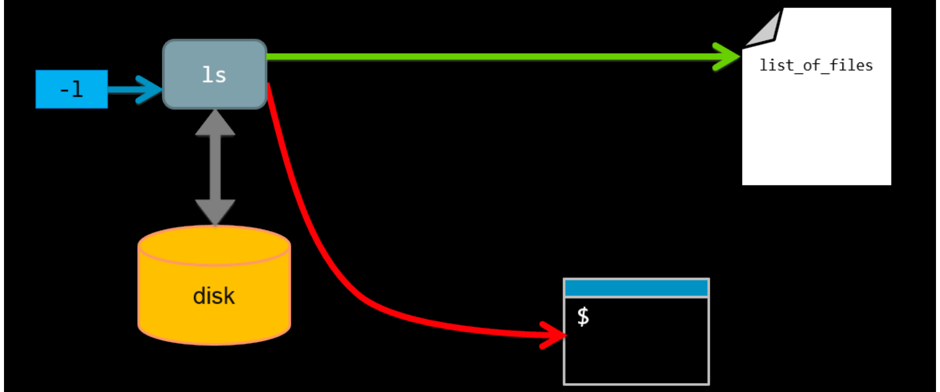


Output redirection summary

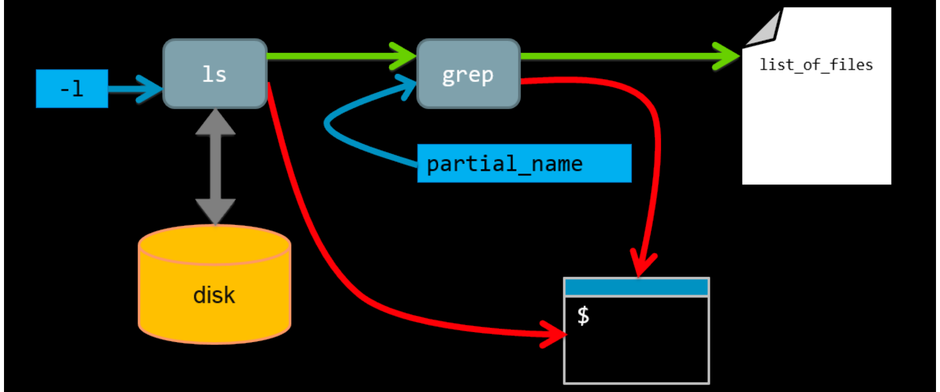


ls -l

ls -l > list_of_files



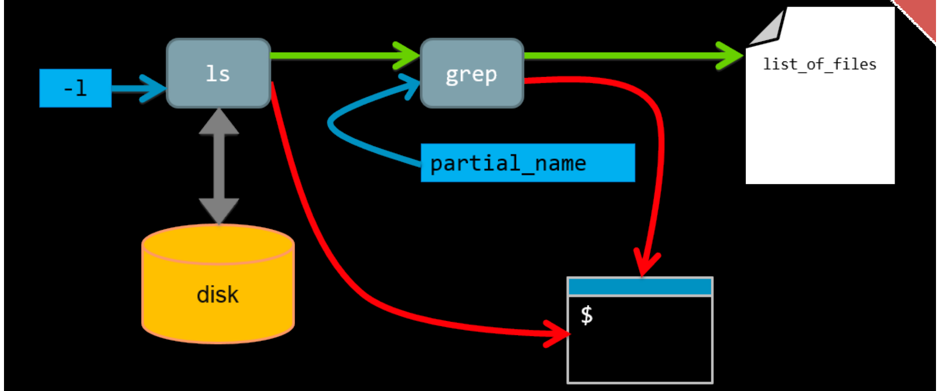
ls -l | grep partial_name > list_of_files



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

ls -l /usr/bin/ | grep “tree” > list_of_files

Do it Yourself!



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

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	Crl-c
ssh	| (pipe)	touch	source	git	Crl-z
	> (write to file)		cat	R	bg
	< (read from file)			python	fg

Side-Note: **grep** ‘power command’

Do it Yourself!

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

```
> grep --color "GESGKS" sequences/data/seqdump.fasta
```

```
REVKLLLLGAGESGKSTIVKQMKIIHEAGYSEEECKQYK
```

- 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.

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 "[^ATGC]"
```

Exercises:

(1). Use “man grep” to find out what the **-v** argument option is doing!

(2). 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’)

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 McIlory

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.
- Kunth wrote a literate programming solution that was 7 pages long, and also highly customized to this problem (e.g. Kunth 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.

Do it Yourself!

Hands-on time

Section 4 of software carpentry UNIX lesson
https://swcarpentry.github.io/shell-novice/

~15 mins

Shell scripting

Do it Yourself!

```
#!/bin/bash
# This is a very simple hello world script.
echo "Hello, world!”
```

Exercise:

- Create a "Hello world"-like script using command line tools and execute it.
- Copy and alter your script to redirect output to a file using **>** along with a list of files in your home directory.
- Alter your script to use **>>** instead of **>**. What effect does this have on its behavior?

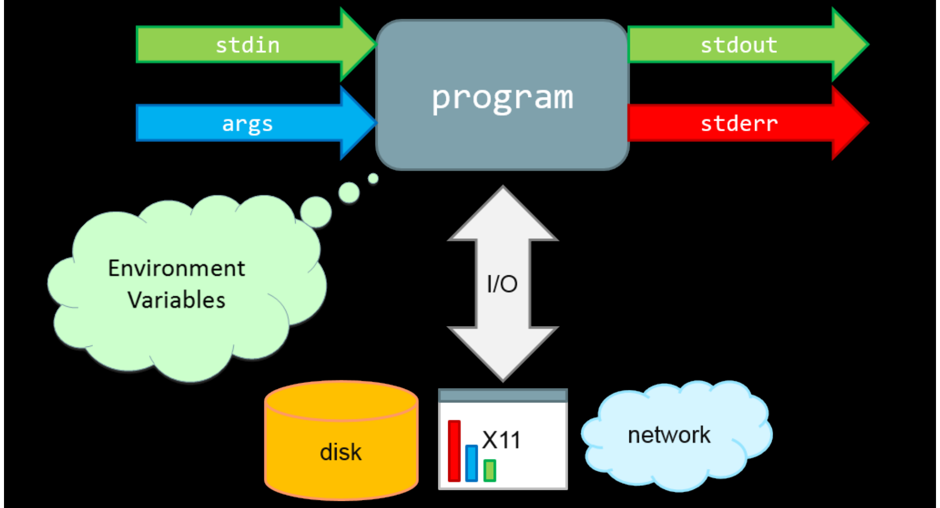
Variables in shell scripts

Do it Yourself!

```
#!/bin/bash
# Another simple hello world script
message='Hello World!'
echo $message
```

- “message” - is a **variable** to which the string 'Hello World!' is assigned
- echo - prints to screen the contents of the variable "$message"

Side-Note: Environment Variables



$PATH ‘special’ environment variable

- What is the output of this command?

  > echo $PATH

- Note the structure: <path1>:<path2>:<path3>
- **PATH** is an environmental variable which Bash uses to search for commands typed on the command line without a full path.
- *Exercise*: Use the command **env** to discover more.

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…

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	Crl-c
ssh	**|** (pipe)	touch	source	git	Crl-z
	> (write to file)		cat	R	bg
	< (read from file)			python	fg

How to Get Working

1) Connecting to jetstream

2) 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/