

Topic 2: Programming for Biologists

Included in this topic

... and the things that I think are most important

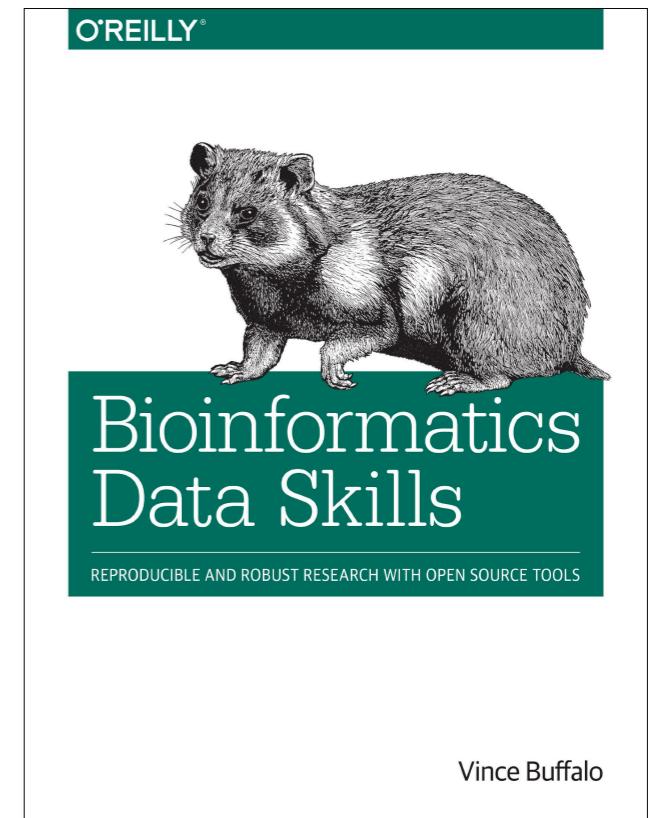
- Reproducibility
- The command line environment
- Paths (absolute versus relative)
- Modularity
- Metacharacters
- Different programming languages and why
- Learn a good text editor!

Ask for help!

There are lots of sources of help around - make use of them!

Sometimes the solution to a problem is simply knowing the right terms to Google or to search on StackOverflow

I highly recommend Vince Buffalo's "Bioinformatic Data Skills"



Non-command line options

There are a number of non-command line options out there

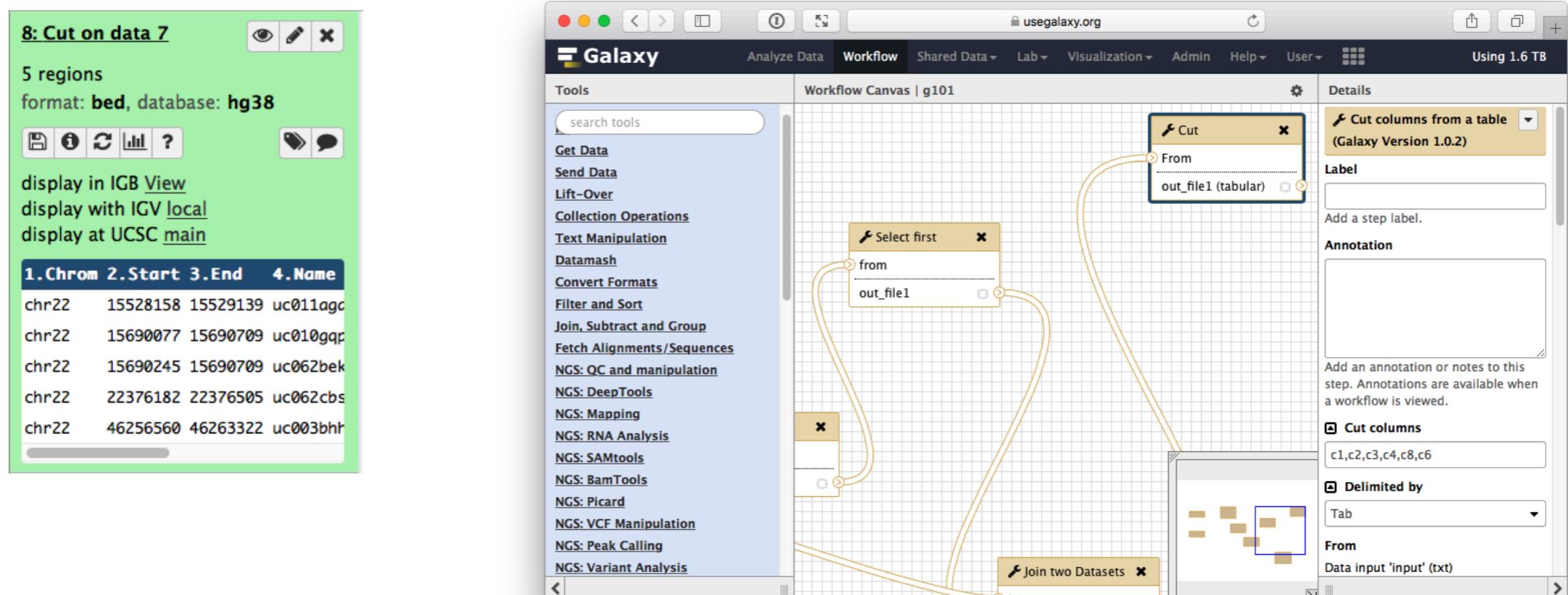
For example

- Galaxy (<https://www.galaxyproject.org/>)
Open Source, runs many of the common tools
-  geneious
Commercial (a license costs \$200 per year for a student)

GUI = Graphical User Interface

Galaxy

- Run pre-established tools, setup is form-based
- Jobs submitted to a cluster of servers (running the software stack)



Comprehensive suite of molecular biology and sequence analysis tools

[FREE TRIAL >](#)

[PRICING >](#)



Tons of features and supports most of the common types of analyses

- Assembly
- Alignment
- *Primer design, probe design*
- Variant calling
- Annotation
- Even phylogenetics!

So why use the command line?

So why use the command line?

- Efficiently work with large files
- Steps recorded, repeatable, auditable
- Powerful text editing tools. Powerful code revision tools
- Generally faster than GUI based methods
- Most scientific programs do not have a GUI (necessity)
- Highly portable and communicable
- Allows different programs to be combined arbitrarily
- It's free and makes you feel like a hacker

Typical things that you might do at the command line

- Download and install packages
- Download, decompress and validate datasets
- Run software
 - Fine tune parameters
 - Analyse data

Reproducibility

One of our responsibilities as scientists is to present our methods so that others could reproduce them

In bioinformatics, keeping a record of what you do at the computer is as important as keeping detailed notes in your lab book

Your worst enemy when working on a computer is yourself from one year ago



Reproducibility

There are many ways to practice reproducibility

Clearly identify programs and dependencies

Documented shell scripts

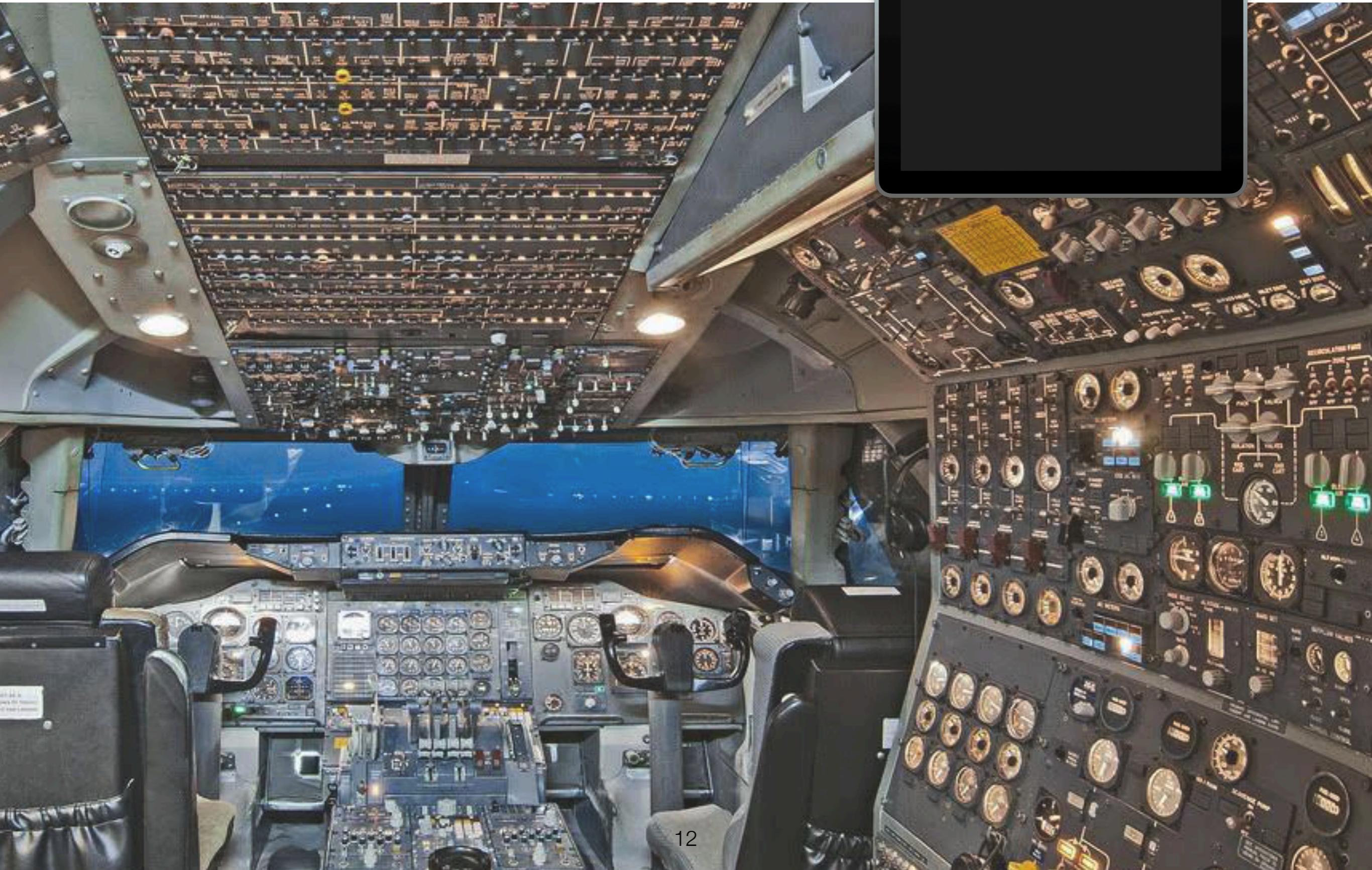
Variable labels for all inputs

Version control



The UNIX SHELL

>_



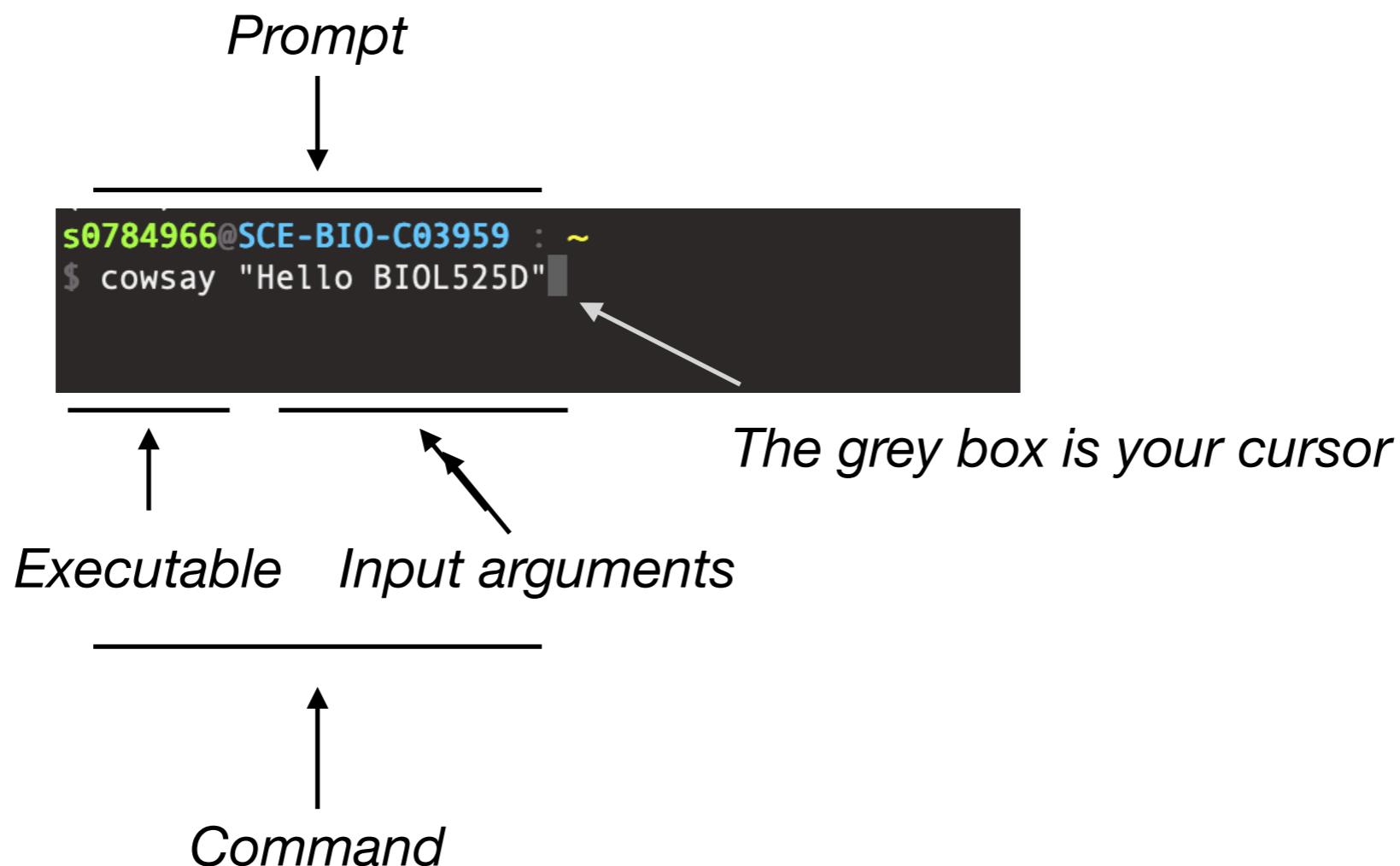
Interface: prompt

When you open the Terminal on Mac or Linux systems or open a shell instance through Putty you'll get faced with something like this:



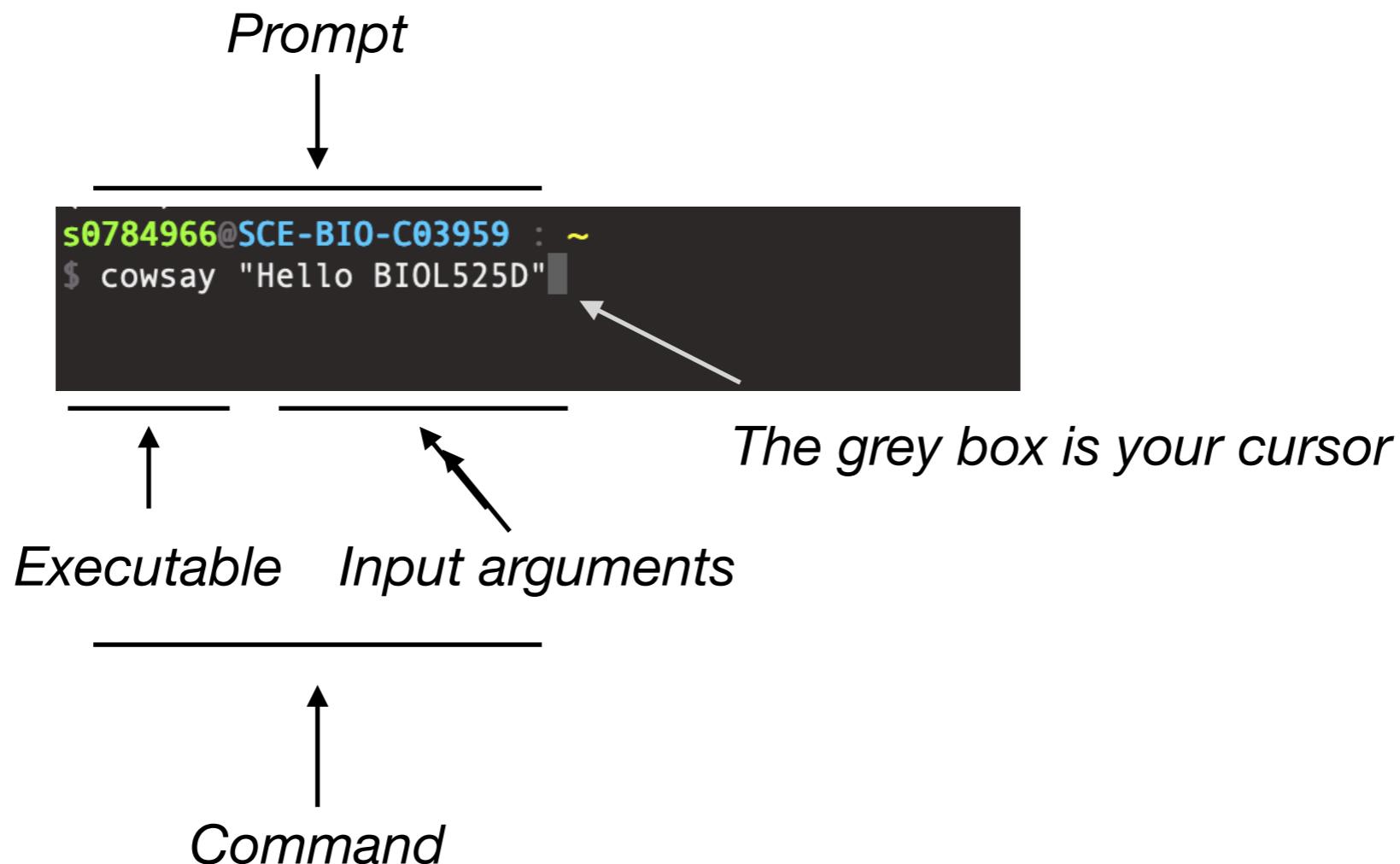
Interface: prompt

When you open the Terminal on Mac or Linux systems or open a shell instance through Putty you'll get faced with something like this:



Interface: prompt

When you open the Terminal on Mac or Linux systems or open a shell instance through Putty you'll get faced with something like this:



Interface: output

```
s0784966@sce-bio-c03959 : ~  
$ cowsay "Hello BIOL525D"  
  
< Hello BIOL525D >  
-----  
 \ ^__^  
  (oo)\_____  
   (__)\       )\/\  
    ||----w |  
    ||     ||  
  
(base)
```

← *Output (STDOUT)*

Interface: arguments

```
s0784966@sce-bio-c03959 : ~
$ cowsay "Hello BIOL525D"

< Hello BIOL525D >
-----
      \  ^__^
       \  (oo)\_____
          (__)\       )\/\
              ||----w |
              ||     ||

(base)
s0784966@sce-bio-c03959 : ~
$ cowsay -t "I'm tired now"

< I'm tired now >
-----
      \  ^__^
       \  (--) \_____
          (__)\       )\/\
              ||----w |
              ||     ||

(base)
s0784966@sce-bio-c03959 : ~
$ cowsay -e'$$' -W 5 "I'm a high roller"

/ I'm \
| a   |
| high |
| roll |
\ er  /
-----
      \  ^__^
       \  ($$)\_____
          (__)\       )\/\
              ||----w |
              ||     ||

(base)
```

The “-t” option specifies that the cow should be sleepy

The “-e’\$\$’” option specifies that the cow should have dollar signs for eyes

The “-W 5” option specifies that the text should be ‘wrapped’ every 5 characters (or so)

Interface: grammar (GNU syntax)

Programs take options and parameters. For example:

```
rm -i -f --one-file-system --interactive=always A.txt B.gz
```

rm: the executable/program. always first. the other arguments are specific to that program.

-i -f: short options. single ``-``. relative order is generally unimportant.collapsible, e.g-if

--one-file-system: long options. double ``-``. some short and long forms are equivalent.

=always: some options take a param value. ``=`` introduces the value applied to the preceding long option. Short options can take a parameter, but without ``=`` , e.g.: `tar -cF.gz mydir`

•**A.txt B.gz** Positional parameters. Variable number of them can be provided. Depending on the program order matters.

Some common commands

There are many, many UNIX commands to learn and get used to.

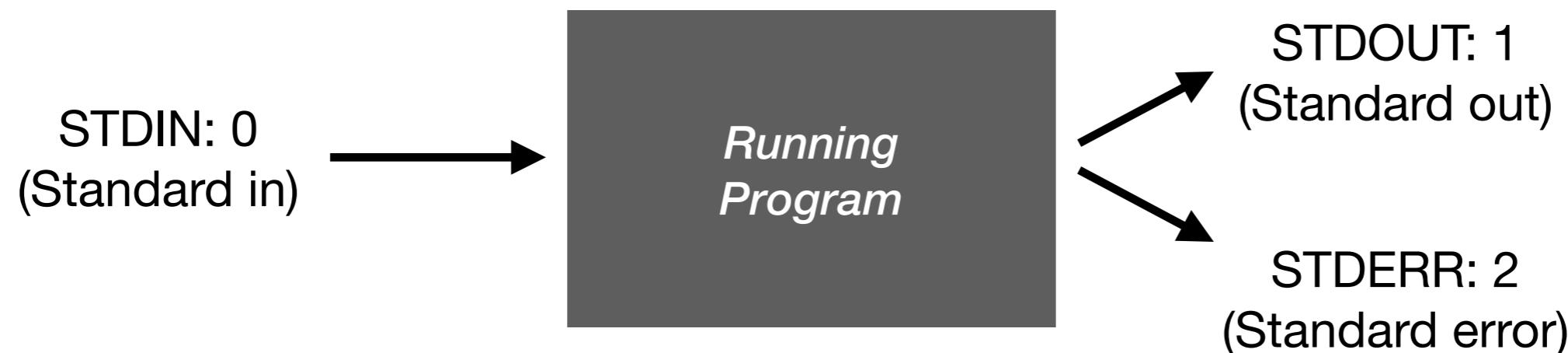
Here's a very silly list

ls	cd	wc	top	cp
ps	cat	mkdir	head	mv
less	more	which	pwd	rm
grep	awk	sed	chmod	ssh
seq	echo	who	kill	passwd

The “man” (short for manual) command gives you details about each program
e.g. “\$ man cowsay”

Interface: stdio

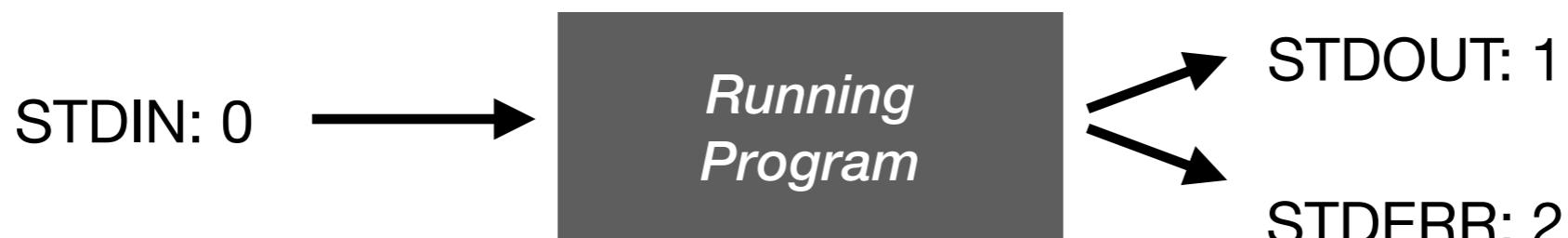
There are three files (or streams) that programs have access to



Under UNIX (the ancestor of Linux, MacOS android)
EVERYTHING IS A FILE

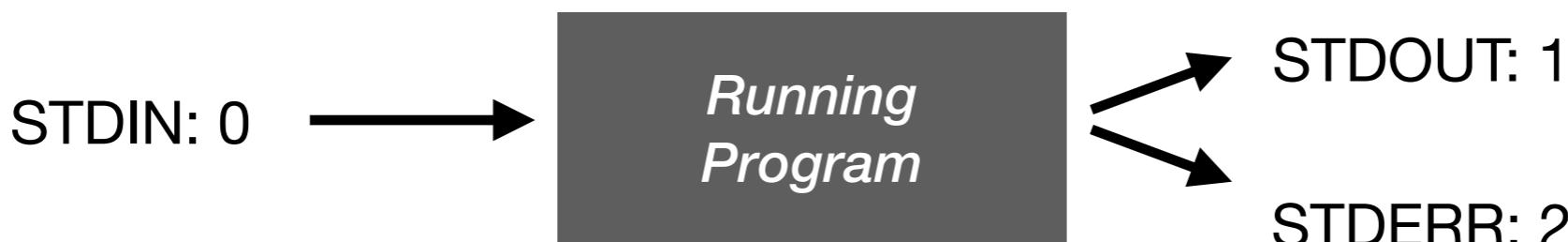
Programs, user input, output, directories (folders) and files are all files and have specific locations

Redirect and pipes



You can manage the different streams using
“redirects” (>) and “pipes”(|)

Redirect and pipes

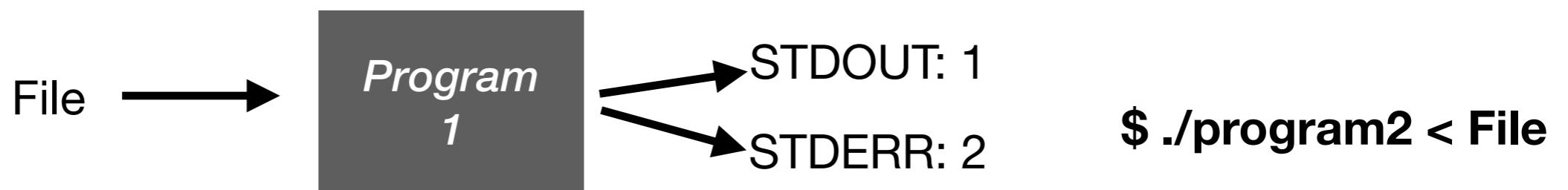


You can manage the different streams using
“redirects” (>) and “pipes”(|)

The “>” symbol redirects STDOUT



The “<” symbol redirects STDIN



Redirect and pipes

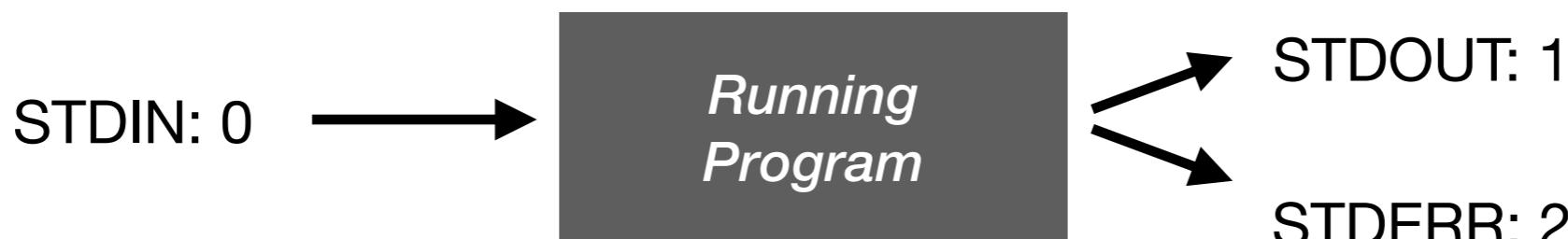
```
s0784966@sce-bio-c03959 : ~/temp/temp
[$ ls
./          CoolFile1.txt  temp1.txt      temp1b.txt    temp2.txt
../          CoolFile2.txt  temp1a.txt      temp1c.txt    temp3.txt
(base)
s0784966@sce-bio-c03959 : ~/temp/temp
[$ ls > fileList.txt
(base)
s0784966@sce-bio-c03959 : ~/temp/temp
[$ cat fileList.txt
./
...
CoolFile1.txt
CoolFile2.txt
fileList.txt
temp1.txt
temp1a.txt
temp1b.txt
temp1c.txt
temp2.txt
temp3.txt
(base)
```

Use **ls** to list all the files present

Redirect **ls** to “fileList.txt” to keep a record of the files present

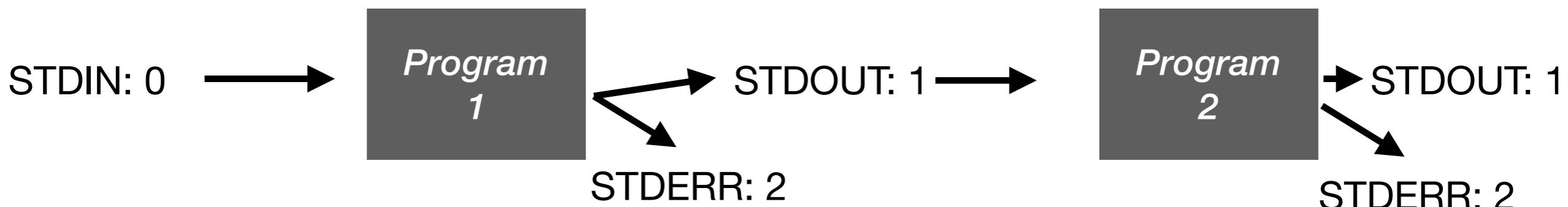
Inspect the contents of “fileList.txt”

Redirect and pipes



You can manage the different streams using
“redirects” (>) and “pipes”(|)

Pipes sets the STDOUT of one process to STDIN for another



Redirect and pipes

Pipe example

```
s0784966@sce-bio-c03959 : ~/temp
$ ls
./          CoolFile1.txt  temp1.txt      temp1b.txt    temp2.txt
..          CoolFile2.txt  temp1a.txt      temp1c.txt    temp3.txt
(base)
s0784966@sce-bio-c03959 : ~/temp
$ ls | sort -r
temp3.txt
temp2.txt
temp1c.txt
temp1b.txt
temp1a.txt
temp1.txt
CoolFile2.txt
CoolFile1.txt
./
..
(base)
s0784966@sce-bio-c03959 : ~/temp
$ ls | sort -r | head -n2
temp3.txt
temp2.txt
(base)
s0784966@sce-bio-c03959 : ~/temp
$
```

List files using **ls**

Pipe the output of **ls** to **sort** and use the reverse order option

Pipe the output of **ls** to **sort** and use the reverse order option then pipe to **head** to display the first 2 items

Metacharacters

The pipe and redirect symbols (|, > and <) are special characters (called meta characters)

When working at the command line, there are a number of other metacharacters that you should be aware of

Metacharacters

The asterix (*)

- Used to match any and all characters

The question mark (?)

- Used to match a specific character

```
s0784966@sce-bio-c03959 : ~/temp
$ ls
./          temp1.txt      temp2.txt
../         temp1a.txt     temp3.txt
CoolFile1.txt temp1b.txt
CoolFile2.txt temp1c.txt
(base)

s0784966@sce-bio-c03959 : ~/temp
$ ls temp*
temp1.txt   temp1b.txt   temp2.txt
temp1a.txt  temp1c.txt   temp3.txt
(base)

s0784966@sce-bio-c03959 : ~/temp
$ ls temp?.txt
temp1.txt  temp2.txt  temp3.txt
(base)

s0784966@sce-bio-c03959 : ~/temp
$
```

Metacharacters

Brackets []

- Used to match a specific set of parameters

Hyphen - *(when within brackets) -*

- Used to specify an alphanumeric range

```
s0784966@sce-bio-c03959 : ~/temp$ ls  
./ temp1.txt temp2.txt  
../ temp1a.txt temp3.txt  
CoolFile1.txt temp1b.txt  
CoolFile2.txt temp1c.txt  
(base)  
s0784966@sce-bio-c03959 : ~/temp$ ls temp1[a,b]*txt  
temp1a.txt temp1b.txt  
(base)  
s0784966@sce-bio-c03959 : ~/temp$ ls temp[1-2]*txt  
temp1.txt temp1b.txt temp2.txt  
temp1a.txt temp1c.txt  
(base)  
s0784966@sce-bio-c03959 : ~/temp$ ls temp1[b-c]*txt  
temp1b.txt temp1c.txt  
(base)  
s0784966@sce-bio-c03959 : ~/temp$
```

Metacharacters

When working at the command line, there are a number of special characters (called metacharacters) that you should be aware of

Ampersand - “&”

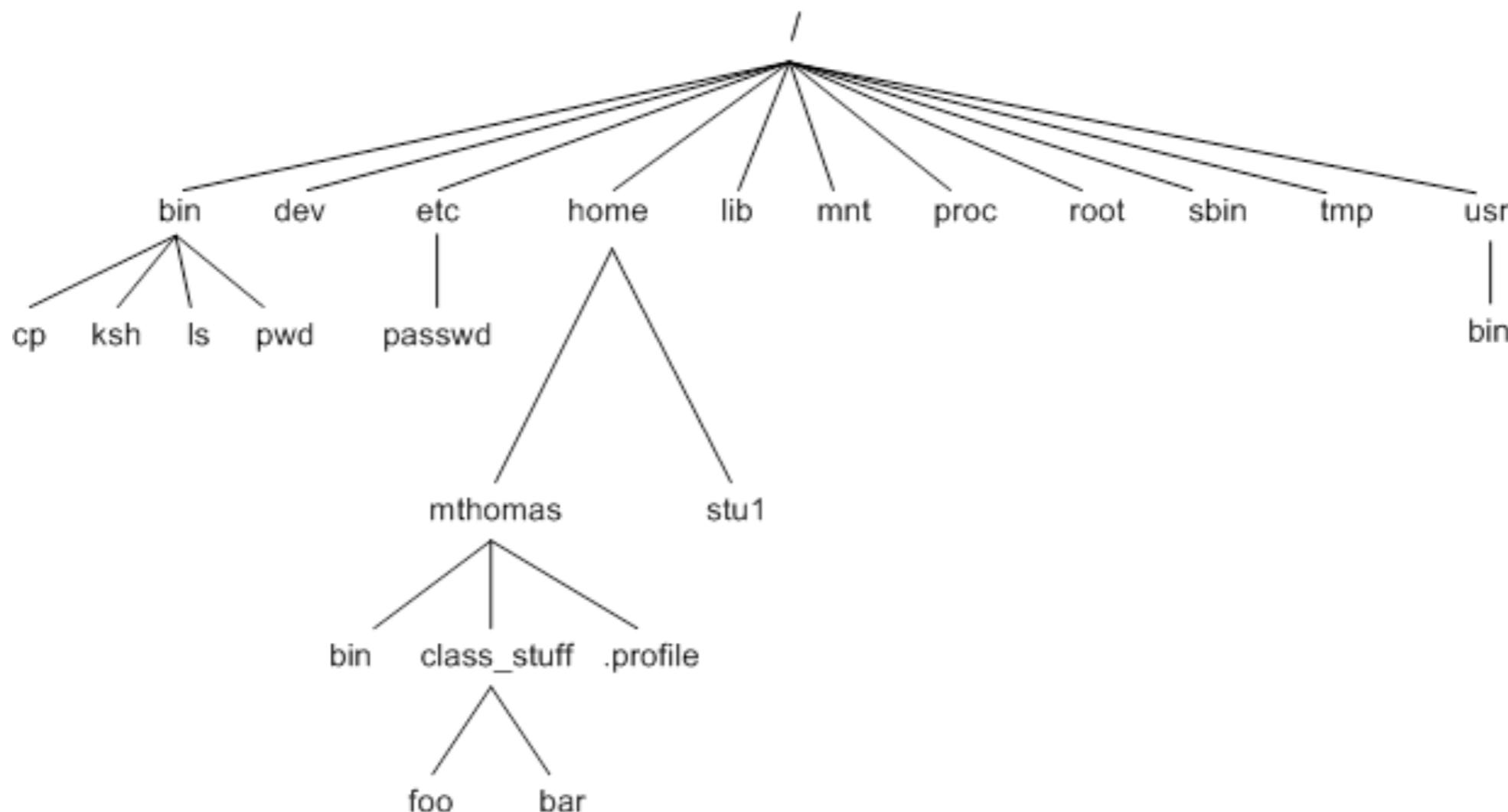
- Used to “background” a process

This leaves a process running in the background allowing you to keep working on other things while it finishes

We'll make use of this in the tutorial

The UNIX Filesystem

EVERYTHING IS A FILE!



Learning to navigate this file hierarchy is critical

Paths: relative v. absolute

Most files that you will analyse have a location (path) in the hierarchy of files

You can specify the location of a file using it's full address (absolute path) or where it is in relation to your current location (the relative path)

Paths: relative v. absolute

```
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff
$ ls
./          .DS_Store    BIOL525D/      VEG/
../          Admin/       ComputerSetup/  mac/
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff
$ pwd
/Users/s0784966/UBC/ZoologyStuff
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff
$ cat BIOL525D/mycmd.sh
#!/bin/sh

age=31

echo "I am $age years young"
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff
$ cat /Users/s0784966/UBC/ZoologyStuff/BIOL525D/mycmd.sh
#!/bin/sh

age=31

echo "I am $age years young"
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff
$ cd
(base)
s0784966@sce-bio-c03959 : ~
$ cat BIOL525D/mycmd.sh
cat: BIOL525D/mycmd.sh: No such file or directory
(base)
s0784966@sce-bio-c03959 : ~
$ cat /Users/s0784966/UBC/ZoologyStuff/BIOL525D/mycmd.sh
#!/bin/sh

age=31

echo "I am $age years young"
(base)
s0784966@sce-bio-c03959 : ~
```

ls (list)

tells you the files present in your current location

pwd -(print working directory)

tells you where you are in the file system

I can access a file in the “BIOL525D” folder using it’s relative path (i.e. making use of my current location)

Or, I can access a file in the “BIOL525D” folder using it’s absolute path (i.e. making use of the file’s location)

If I leave the /Users/s0784966/UBC/ZoologyStuff directory, the relative path I used before no longer works

The absolute path works wherever you are though

Paths: relative v. absolute

The text “./“ refers to the directory that you are in

The text “..“ refers to one directory back in the file system

“..“ can be chained together to refer back multiple locations

```
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ ls
./          .DS_Store      AllFiles/      Topic_1/
..          2020_profiles/  Images/       Topic_2.key*
                                         biol525d/    biol525d 2/
                                         mycmd.sh*
                                         software/
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ cd software/
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D/software
[$ cat mycmd.sh
cat: mycmd.sh: No such file or directory
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D/software
[$ cat ./mycmd.sh
#!/bin/sh

age=31

echo "I am $age years young"
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D/software
$
```

Here's the contents of the “BIOL525D” directory

I'll move into the “software” directory

I can not access the “mycmd.sh” file

I can refer to the “mycmd.sh” file if I tell the system that it is one directory back in the file system

Interface: permissions

```
drwxr-xr-x 12 niteshb users 4.0K Apr  8 20:51 testdir
|[-][-][-]  [----] [---]
| | | | | | +-----> Group
| | | | | +-----> Owner
| | | | +-----> Others Permissions
| | | +-----> Group Permissions
| | +-----> Owner Permissions
+-----> File Type
```

Each file has a set of permissions that dictate what can and can't be done to that file, and by whom

r - permission to read a file (i.e. see it's contents)

w - permission to write to the file (i.e. to change it in some way)

x - permission to execute the file...

Executable files

If a file is marked as “executable” then you can run it as a standalone program

```
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ ls -l mycmd.sh
-r--r--r-- 1 s0784966 staff 48 24 Nov 09:44 mycmd.sh
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ cat mycmd.sh
#!/bin/sh
age=31

echo "I am $age years young"
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ ./mycmd.sh
-bash: ./mycmd.sh: Permission denied
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ chmod u=rwx mycmd.sh
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ ls -l mycmd.sh
-rwxr--r-- 1 s0784966 staff 48 24 Nov 09:44 mycmd.sh*
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ ./mycmd.sh
I am 31 years young
(base)
s0784966@sce-bio-c03959 : ~/UBC/ZoologyStuff/BIOL525D
$ ]
```

Check permissions

Inspect the contents of the file

Attempt to run file - failure

Alter permissions

Check permissions again

Attempt to run file - success!

Executable files

If a file is marked as “executable” then you can run it as a standalone program

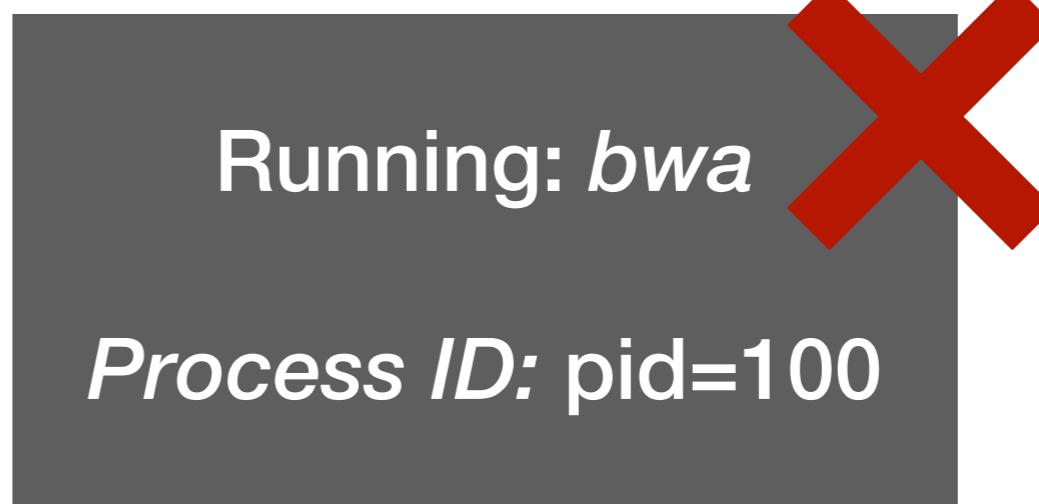
```
s0784966@sce-BIO-C03959 : ~/UBC/ZoologyStuff/BIOL525D
[$ cat mycmd.sh
#!/bin/sh
age=31
echo "I am $age years young"
(base)
```

This piece of text “`#!/bin/sh`” is called a **shebang**
It tells the system where to look for the interpreter to run this script

On my computer, the “sh” command is located in “`/bin/`“

Interface: signals

```
$ kill -KILL 100
```



Signals are one-way messages sent to control running programs

SIGINT (interrupt) (CTRL-C on terminal)

SIGSTOP(pause) / SIGCONT (CTRL-Z on terminal / fg)

SIGKILL (kill the process immediately) (use "kill -9 PID")

Compute Canada servers

- Use a scheduling system
- Tasks must be submitted in specific bash scripts and will run when they get priority
- You have to specify how much RAM and time you need and how many CPUs you want to use.
- You can get an interactive job, which is like working on your server (e.g. the *salloc* command)



Learn a text editor!!

Working at the command line, you often need to edit text files

There are numerous command line text editors, you should invest some time in learning one

Three common ones to take a look at:

```
;;
iLE88Dj. :jD88888Dj:
.LGitE888D.f8GjjjL8888E;
iE :8888Et. .G8888.
;i E888, ,8888,
D888, :8888:
W88W, :8888:
W88W, :8888:
W88W: :8888:
DG8D: :8888:
:8888:
:W888:
:8888:
E888i
tW88D
```



Nano



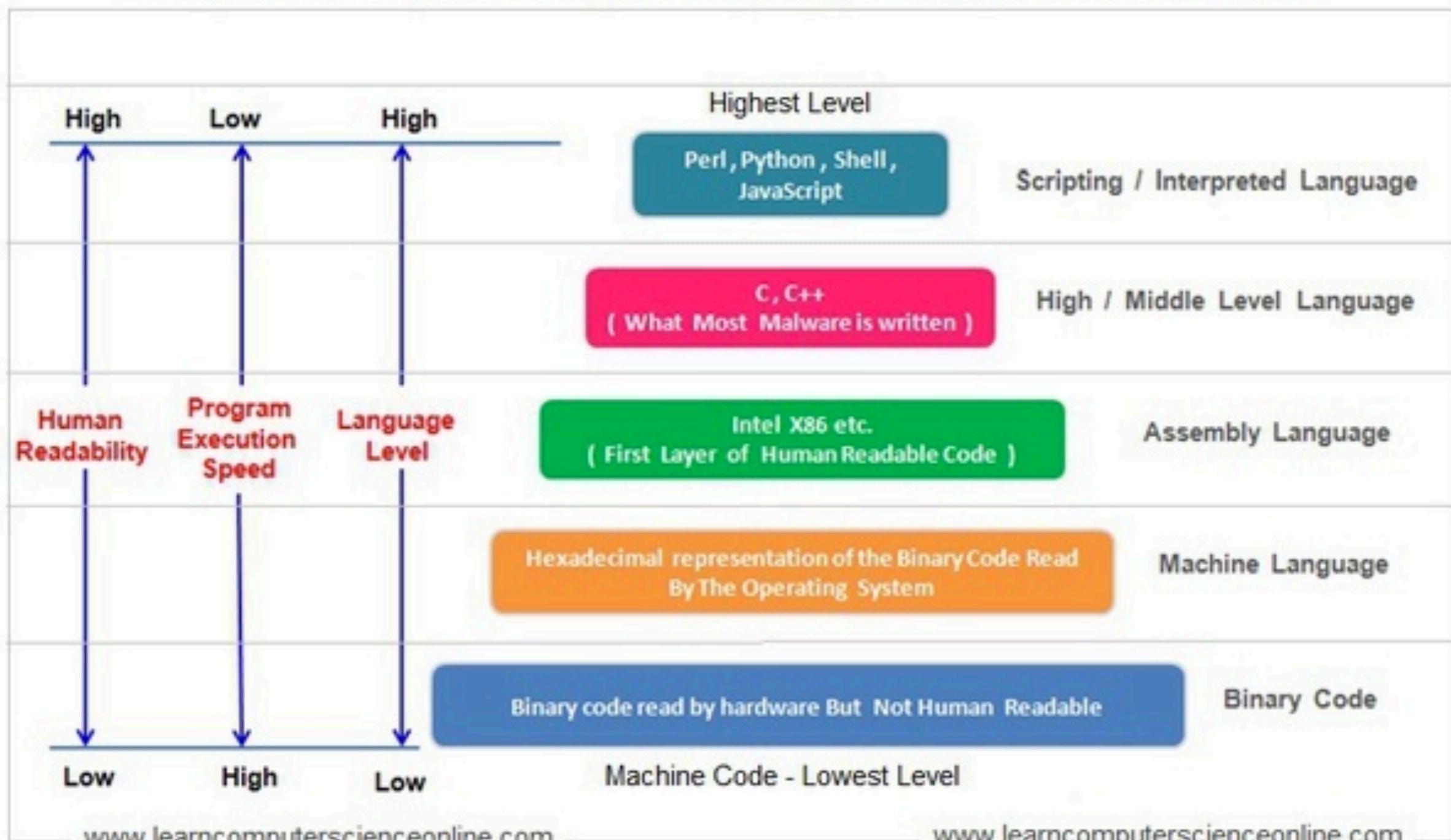
Vim

Emacs

A very short introduction to programming

Programming Languages

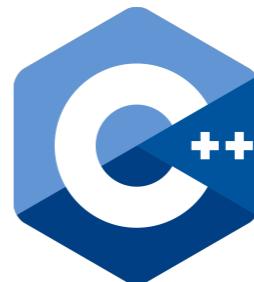
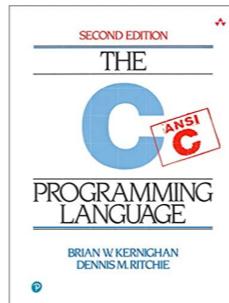
Computer Programming Language - Types And Levels



Programming Languages

That you might come across in Bioinformatics

C/C++ (.cpp)



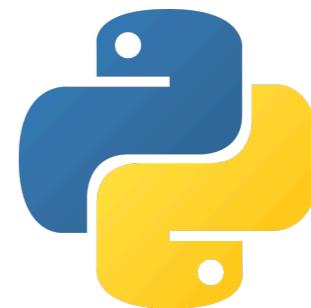
Shell scripting (.sh)



Perl (.pl)



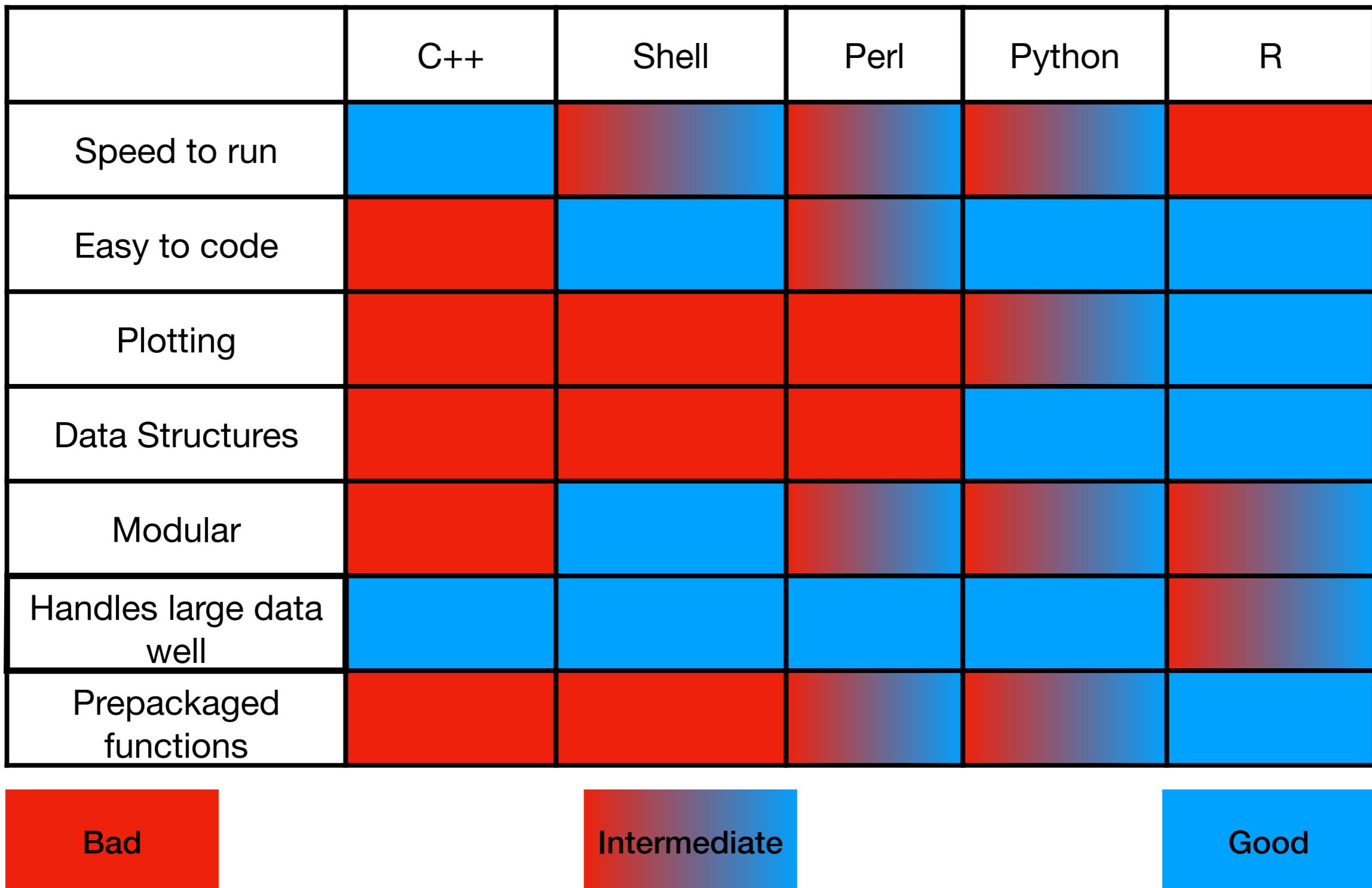
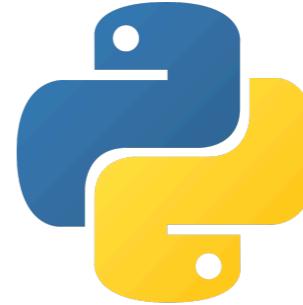
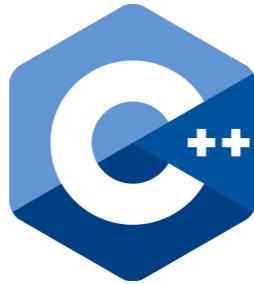
Python (.py)



R (.r)



The text in parentheses refer to the commonly used file extensions for scripts written in the corresponding languages



Recommendations

Genomic dabbler

- Shell and R

Genomic scientist

- Shell, R and Python/Perl

Bioinformatician

- Shell, R, Python/Perl and C++

Recommendations

Numerous repetitive tasks demand speed

Software development demands time

General order:

1. Make it work
2. Make it go fast
3. Make it pretty

Language choice considerations:

- Meld with existing codebase
- Your current and future colleagues' expertise

Take homes

If you are new to UNIX, there has been a lot of information thrown at you very fast and I think it's better to focus on concepts rather than specific commands

- Reproducibility
- The command line environment
- Paths (absolute versus relative)
- Modularity
- Metacharacters
- Different programming languages and why
- Learn a good text editor!

(do bioinformatics at)
How do you get to Carnegie Hall?

(do bioinformatics at)

How do you get to Carnegie Hall?

Practice, practice, practice!!

I know this seems like a cop out, but there is really no better teacher than experience
You'll develop your skills much faster when you have a project that you are working towards



“Fun” coding/bioinformatics problems to test your skills
<http://rosalind.info/problems/locations/>



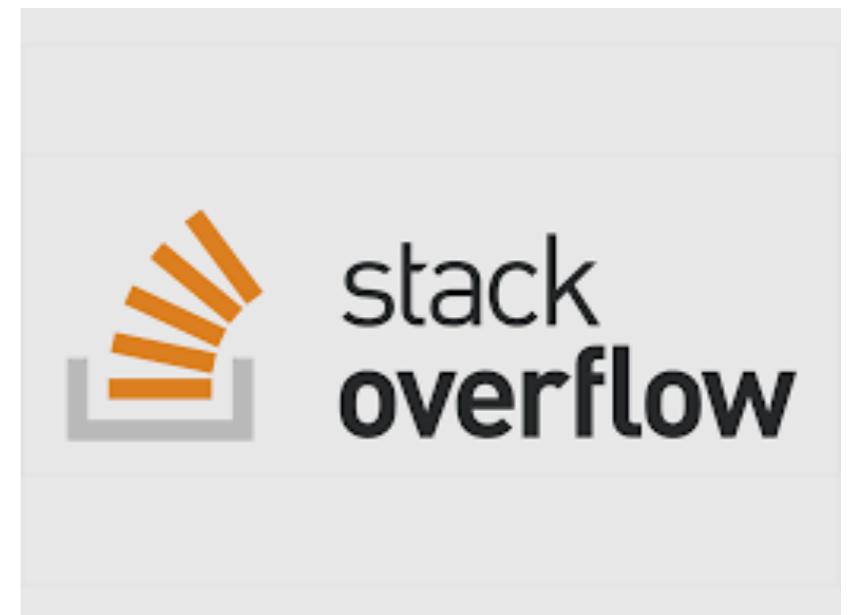
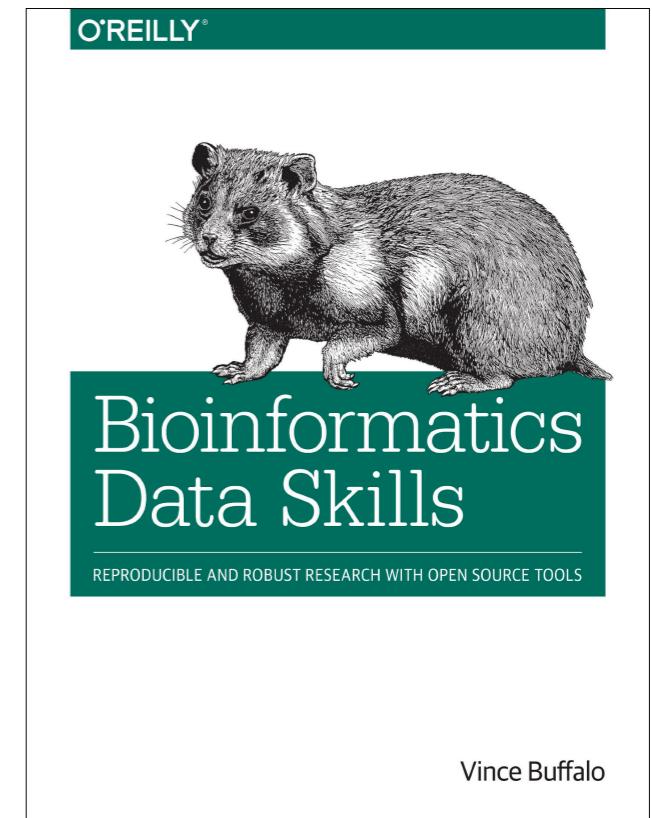
A Python course tailored to Biologists (I personally recommend it)
<https://pythonforbiologists.com/>

Ask for help!

There are lots of sources of help around - make use of them!

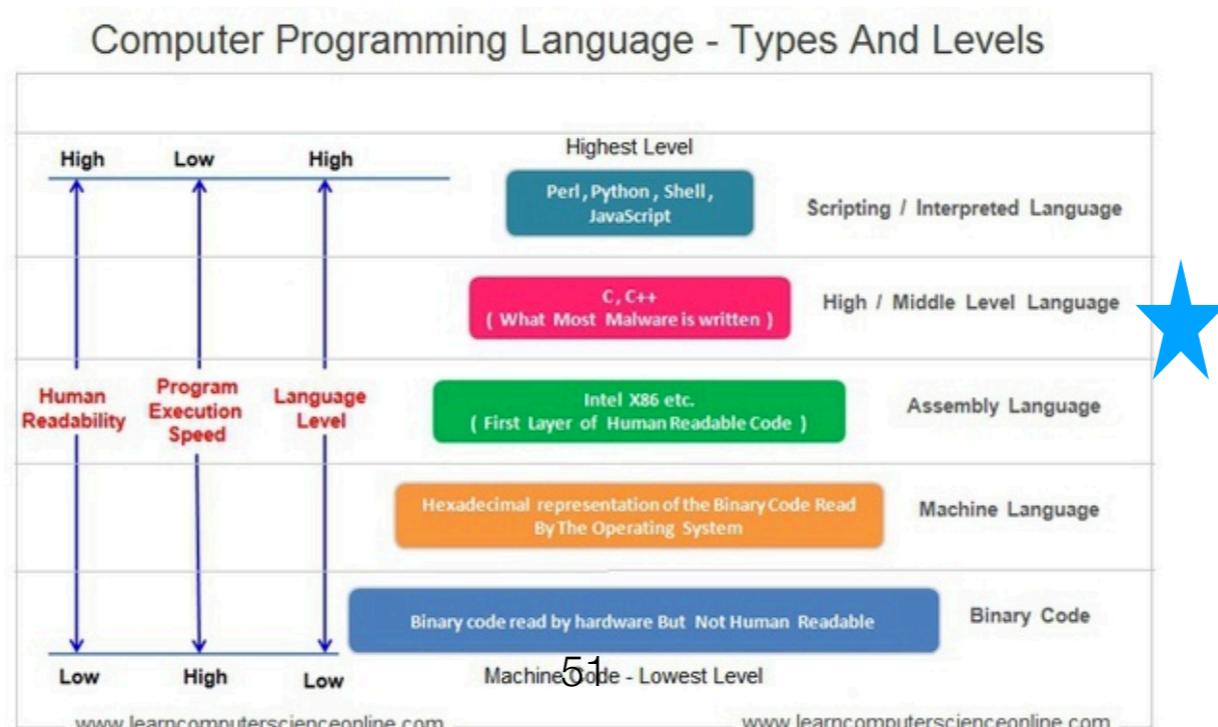
Sometimes the solution to a problem is simply knowing the right terms to Google or to search on StackOverflow

I highly recommend Vince Buffalo's "Bioinformatic Data Skills"



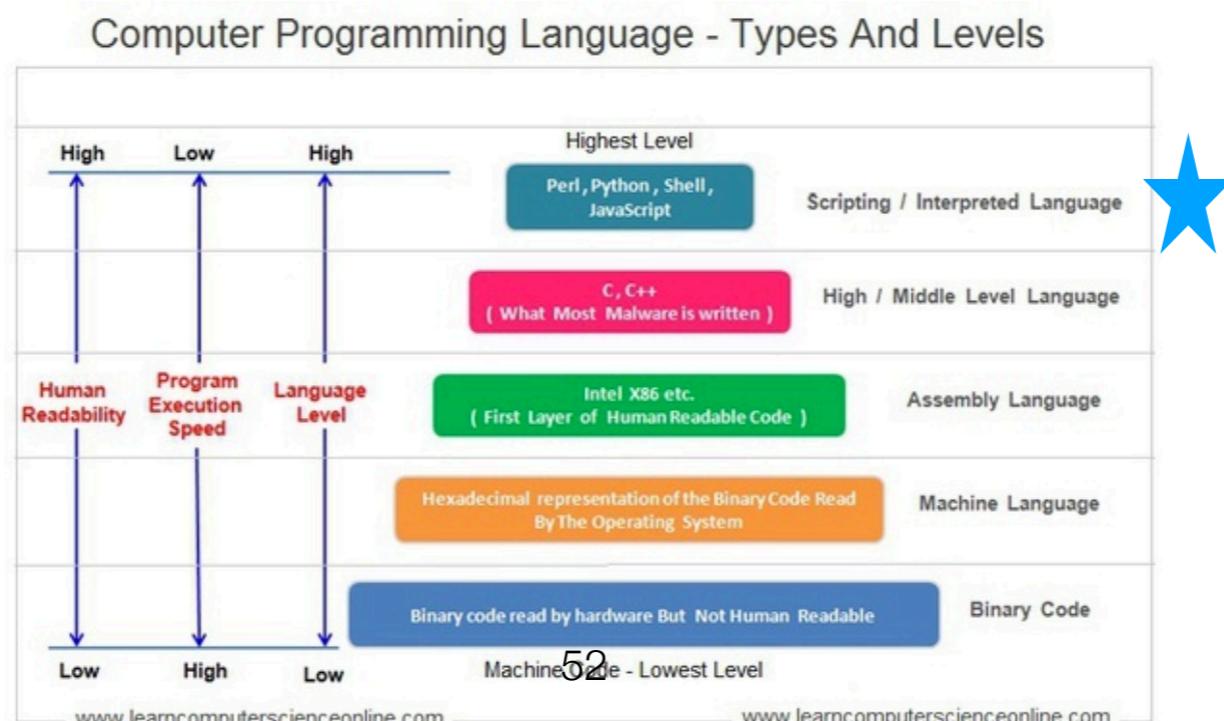
C++

- *Compiles into very fast executable binary files*
- There are very few prepackaged tools
 - *This gives you detailed control*
 - But requires more specialised knowhow and takes longer to write
- Harder to tweak and debug (because you have to compile before running)
- *Good for making low-level tools/libraries*



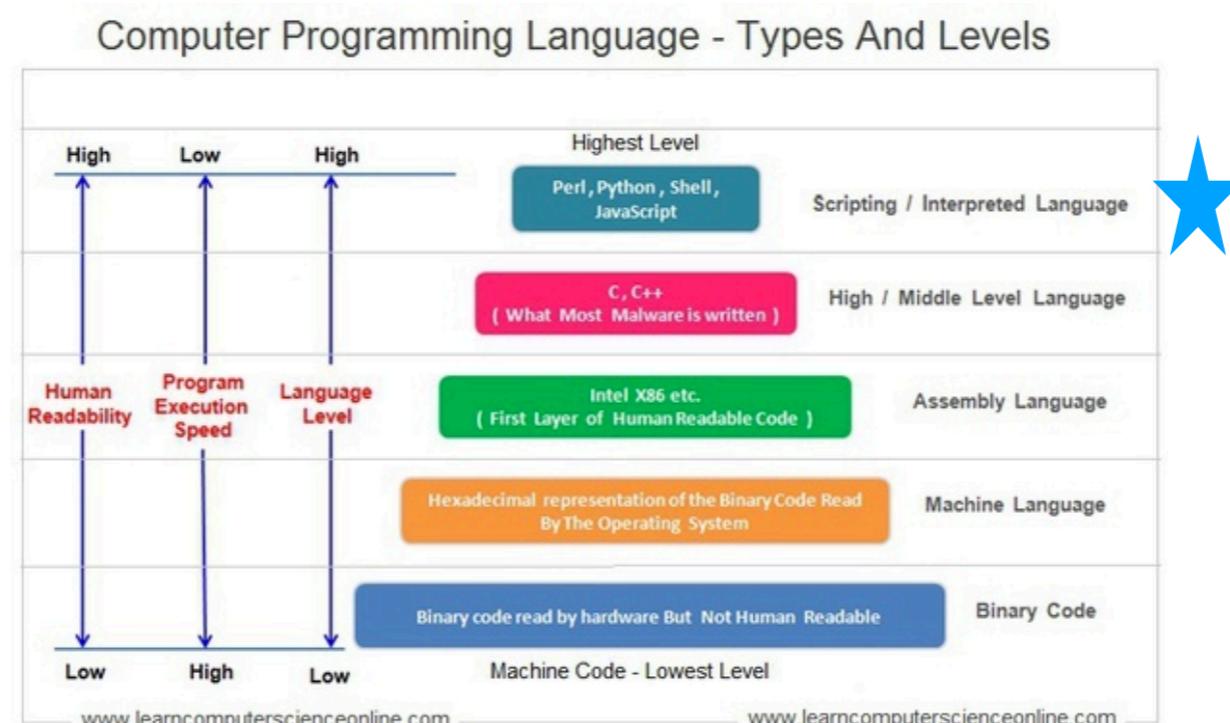
Shell scripting

- *Chain together command line arguments to make repeatable and reusable scripts*
- *Easy to write*
- *Provides the glue between programs written in different languages*
- *Connects naturally with external programs (e.g. sed awk, grep)*
- Needs external programs for data manipulation and analysis



Perl

- *Excels at text processing*
- *Some prepackaged functions*
- Lacks complicated data structures
- Very concise (but at the expense of readability)
- *Fast for a high level language*



Python

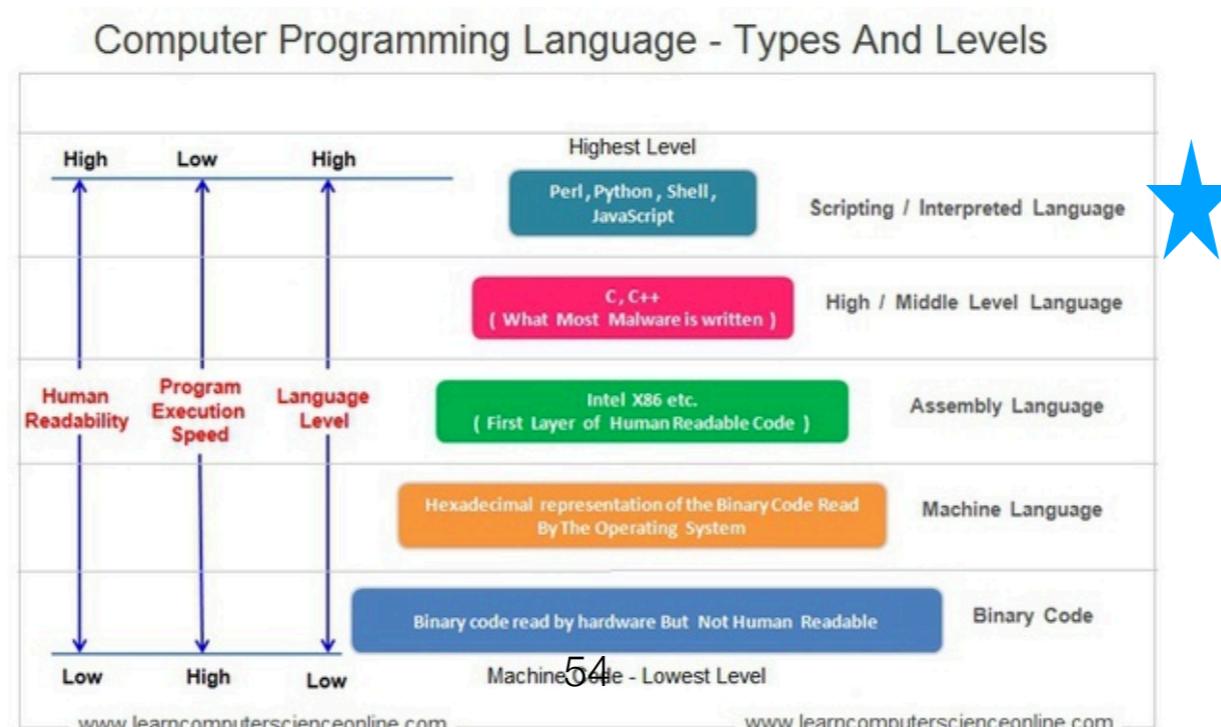
Slightly slower than Perl.

Batteries included. Scientific methods and plotting

Most popular language in the world right now, so there's lots of forum support

Allows rapid development and prototyping.

Good for reformatting data. Good data structures (e.g. handles CSV files well)



*My personal favourite!

R

Many packages for specific scientific tasks and statistics

Great at plotting

Harder to use with big data (GB+ files), although work arounds exist

Slowest option

