Programming and Data Science for Biology (PDSB)

Session 1
Spring 2018

An unfortunate limit on class size...

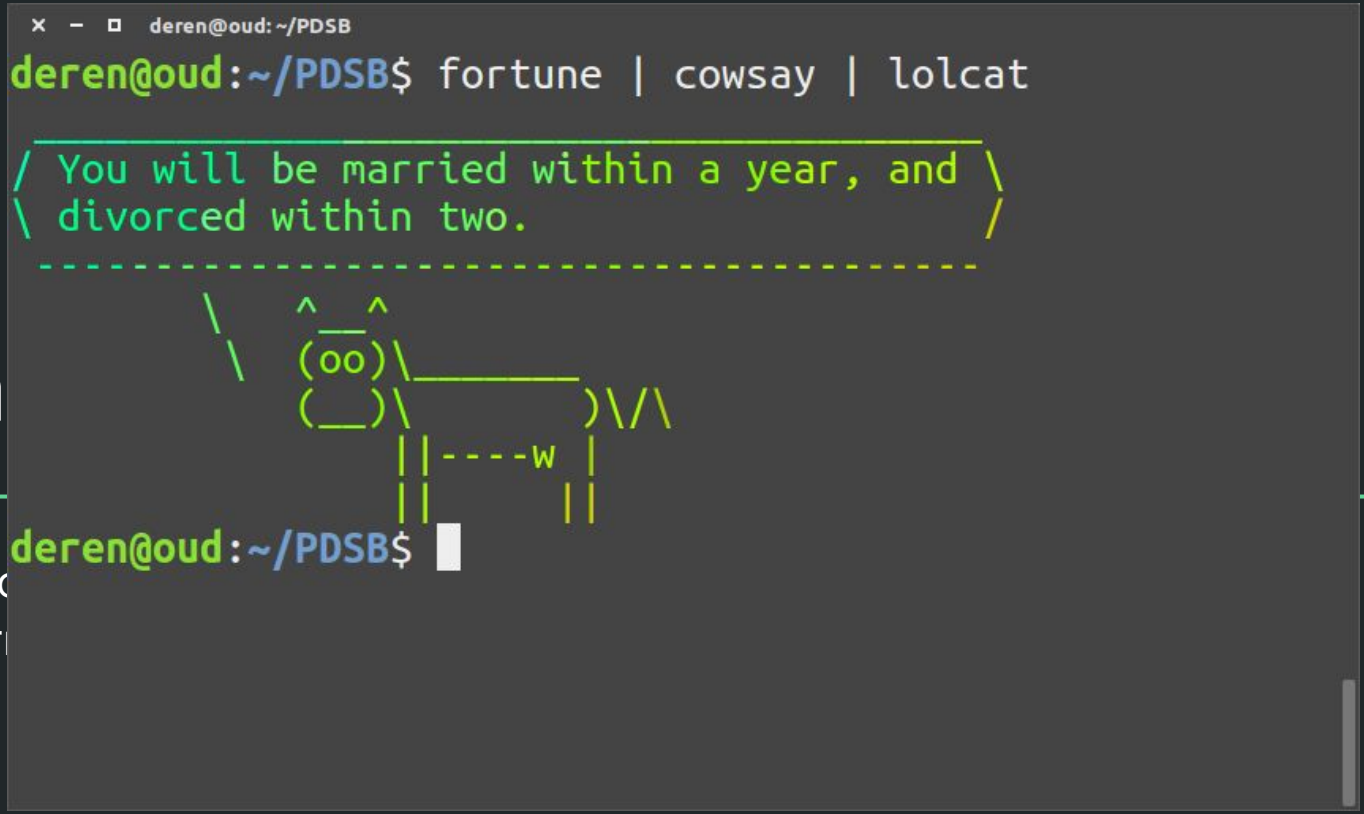
What is this class about?

What is programming?

A logical sequence of arguments to a computer so that it can perform a desired task.

Wh

A logic
perfor

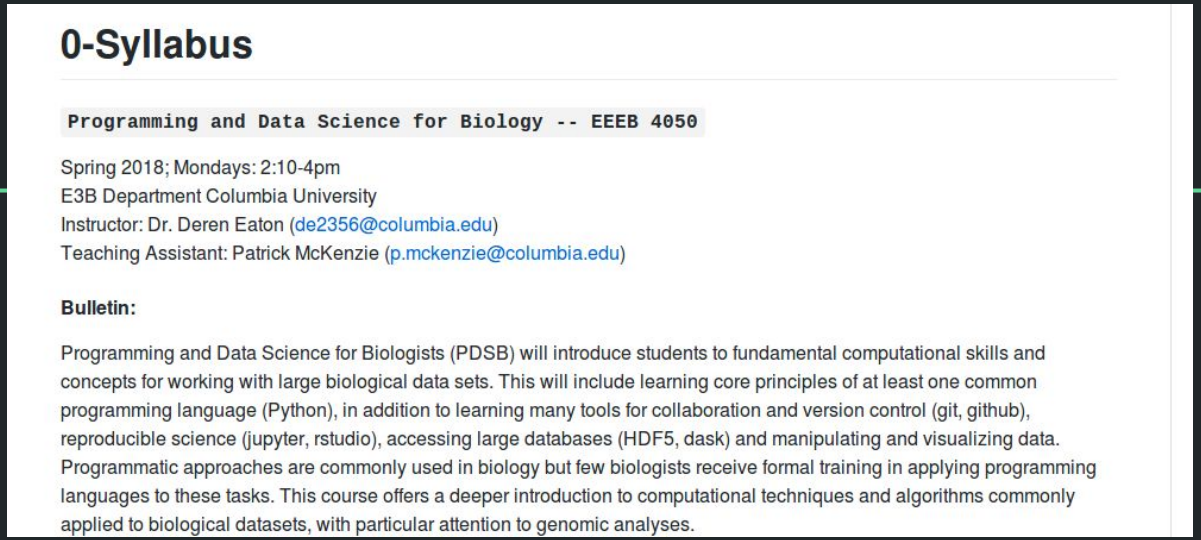


What are programming languages?

A dialect we can use to provide instructions to the computer.
Programming languages change over time, diverge from each other, or adopt from each other, just like spoken languages.

What will we learn?

Link to online syllabus



0-Syllabus

`Programming and Data Science for Biology -- EEEB 4050`

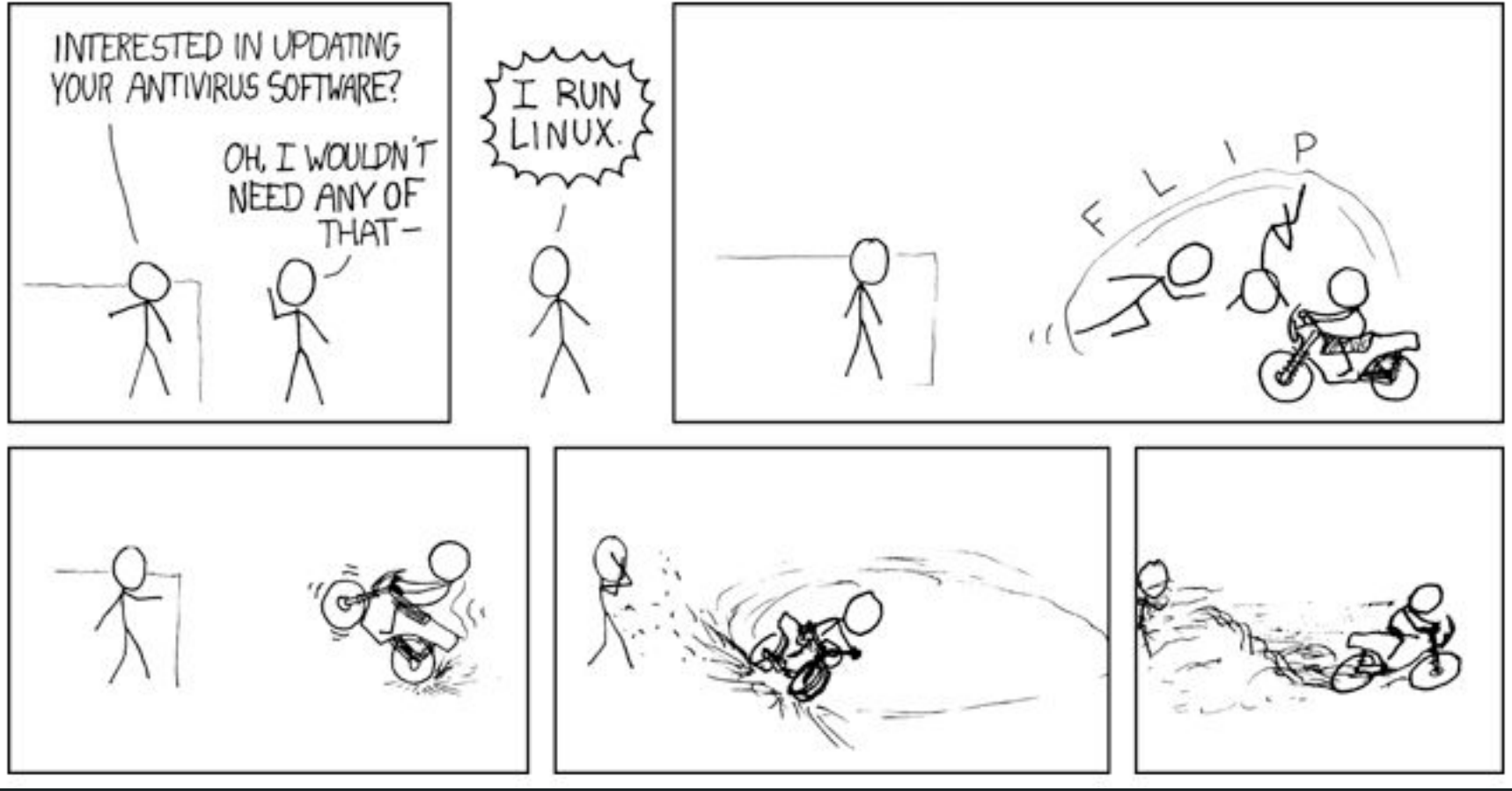
Spring 2018; Mondays: 2:10-4pm
E3B Department Columbia University
Instructor: Dr. Deren Eaton (de2356@columbia.edu)
Teaching Assistant: Patrick McKenzie (p.mckenzie@columbia.edu)

Bulletin:

Programming and Data Science for Biologists (PDSB) will introduce students to fundamental computational skills and concepts for working with large biological data sets. This will include learning core principles of at least one common programming language (Python), in addition to learning many tools for collaboration and version control (git, github), reproducible science (jupyter, rstudio), accessing large databases (HDF5, dask) and manipulating and visualizing data. Programmatic approaches are commonly used in biology but few biologists receive formal training in applying programming languages to these tasks. This course offers a deeper introduction to computational techniques and algorithms commonly applied to biological datasets, with particular attention to genomic analyses.

Do I have the right the kind of computer?

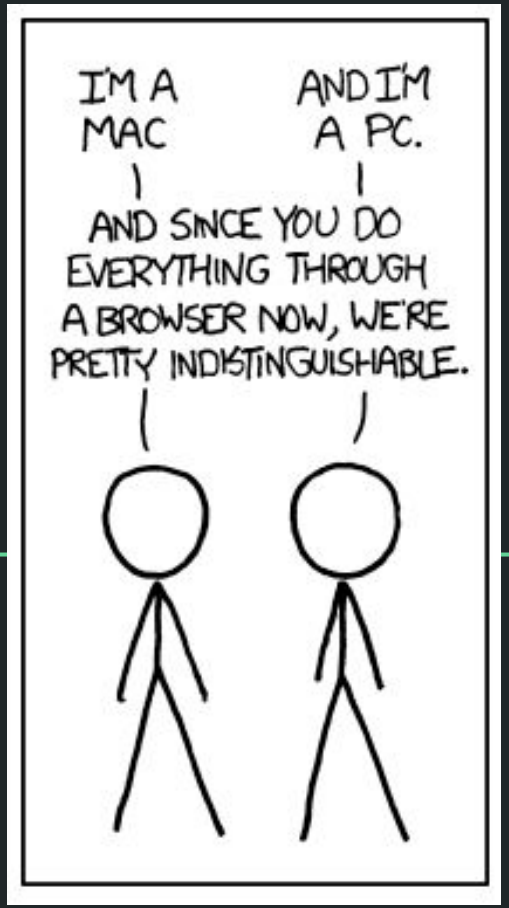
Today scientific computing is overwhelmingly done on Unix-based systems, primarily Linux and Mac OSX, however most programming can be done on any system, including PCs. In this course, we have resources for PC users to access remote Linux environments when needed.


INTERESTED IN UPDATING YOUR ANTIVIRUS SOFTWARE?
OH, I WOULDN'T NEED ANY OF THAT-
I RUN LINUX.
F L I P

If you are on a PC ...

Two options:

1. Install a git-bash terminal (has most tools we need for the class: Installation-instructions
2. Install a linux sub-system (full capabilities, but heavier footprint Installation-instructions



What’s the deal with Unix & Linux?

+ Unix and the C programming language were developed by AT&T Bell Labs in the 1970’s.

+ Unix systems have a modular design with a single unified filesystem and a set of simple tools that each perform a limited task, as well as a shell scripting language to combine the tools to perform complex tasks.

+ GNU/Linux is a free and open source Unix-like system (developed under the same principles in the 90s) that is now the most widely adopted unix-like OS. Apple’s Mac OSX (2000) is also Unix-like and open source.

Open source software

+ Open-source software (OSS) is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose (Wikipedia).

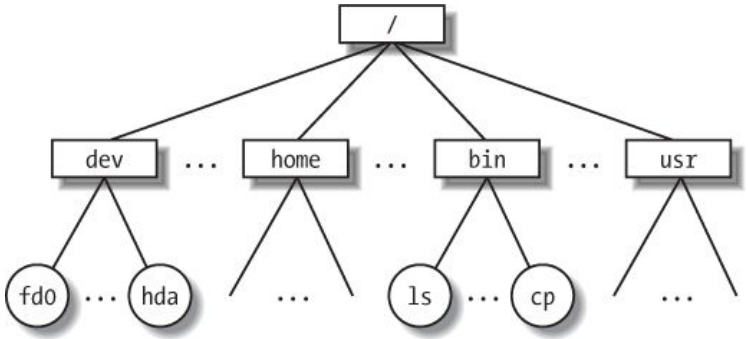
+ Many licenses are commonly used which vary in the restrictions on the right to copy, distribute, or modify code.

+ Open source projects, products, or initiatives embrace and celebrate principles of open exchange, collaborative participation, rapid prototyping, transparency, meritocracy, and community-oriented development (opensource.com)

The *bash* Terminal & the Unix Filesystem

To start, let’s all open a terminal...

The Unix filesystem




```
## home is: ~, /home/{name}/, $HOME
> cd $HOME

## this is your local space
> pwd

## write full path to see other paths
## e.g., /usr is where most software
## is installed.
> ls ./
> ls /usr/
```

The Unix filesystem

Basic syntax of shell commands

`Command -options target`

```
## no target or options.
> pwd

## target is $HOME, no options
> cd $HOME

## target is ‘/usr’, options is -l
> ls -l /usr/
```

The Unix filesystem

Where are my files and applications? Where am I? Why do I need permissions?

```
## show my current location
> ls

## show my location as list
> ls -l

## show as time-sorted list
> ls -lt
```

The Unix filesystem

Making and changing directories. Where am I?

```
## where am I?
> pwd

## make a new directory
> mkdir PDSB

## move into that directory
> cd PDSB

## show what is in new dir
> ls -lt
```

The Unix filesystem

Creating, editing, and reading files.

```
## make a new file in PDSB
> touch newfile.txt

## write some text to it
> echo “some text” > newfile.txt

## show what is new file
> cat newfile.txt

## show what is in new dir
> ls -lt
```

The Unix filesystem

A set of modular commands that together can perform a huge range of complex tasks

```
## some we've seen so far:
> echo "ls, mkdir, cd, cat, touch,
echo, pwd"

## get quick documentation for a tool
> man mkdir

## find more documentation
## google! And tutorials.

## show what is new file
> cat newfile.txt
```

File paths: `/home/deren/PDSB/`

What does it all mean?

File paths

Hierarchical structure: directories and nested in directories starting from the root.

```
## the root directory (system top)
> ls -l /

## read file in current dir
> cat newfile.txt

## what the above cmd implies
> cat ./newfile.txt

## relative vs. absolute paths:
> cat /home/deren/PDSB/newfile.txt
```

File paths

The path variables . and .. point to current and parent directories.

```
## make another nested dir inside
> mkdir testdir

## what is in there?
> ls testdir

## what is one dir above us?
> ls ..

## cd into testdir and then back out
> cd testdir
> cd ..
```

Copying & Moving files

The path variables `.` and `..` point to current and parent directories.

```
## move newfile.txt to testdir
> mv newfile.txt testdir/

## make a copy of newfile.txt to .
> cp testdir/newfile.txt .

## make another copy and name it diff
> cp testdir/newfile.txt newcopy.txt

## remove newfile.txt from .
> rm ./newfile.txt
```

Stdout, Redirection, and Pipes

Directing output

Many programs return text as a result, and you can decide what to do with it. Print to screen, save to file, or pass to another program.

```
## print some text to the terminal
> echo “hello world”

## direct text to write to a file
> echo “hello world” > hello.txt
> cat hello.txt

## direct text to write to file again
> echo “goodbye world” > hello.txt
> cat hello.txt
```

Directing output

“Write” vs. “Append”

```
## append to end of an existing file
> echo “hello again” >> hello.txt
> cat hello.txt

## overwrite file with new text
> echo “only this text” > hello.txt
> cat hello.txt
```

Directing output

“Pipe” to send output of one command to another command

```
## write several lines of text (\n)
> echo -e ‘dog\ncat\nbat’ > hello.txt
> cat hello.txt

## grep selects lines matching target
> grep cat hello.txt

## skip the file, pipe echo to grep
> echo -e ‘dog\ncat\nbat’ | grep cat
```

Directing output

Because most results are text, almost anything can be piped. Combined with ‘grep’ this is very powerful.

```
## find all files in /bin that have
## the letter ‘z’ in them
> ls /bin/ | grep ‘z’
```

Repeating commands with loops

Requires understanding variables and wildcards

Writing loops

The real power of the shell comes from repeating commands on multiple outputs

```
## A variable is a placeholder for
## some text such as a filename
> var=”hello”

## the variable is recalled with $var
> echo $var
```

Writing loops

The real power of the shell comes from repeating commands on multiple outputs

```
## A wildcard is symbol for matching
> alltxt=”*.txt”

## useful in selecting multiple items
## e.g. list all files ending in .txt
> ls $alltxt
```

Writing loops

The real power of the shell comes from repeating commands on multiple outputs

```
## the for-loop structure (don’t run)
for <variable> in <list>
do
  <tasks to be completed>
done

## a simple for-loop (run this)
> for txtfile in *.txt
> do
>   echo “file name=” $txtfile
> done
```

Writing loops

Loop over numbers with brackets. Run commands that loop over numerics.

```
## bracket numbers with .. is range
> for num {1..10}
> do
>   echo $num
> done

## e.g., make several dirs with files
> for num {1..3}
> do
>   mkdir dir$num
>   touch dir$num/file-$num.txt
> done
```

Regular expressions, grep, and sed

The real Master Class skill set

More on grep

grep selects text based on matching context. Optional flags extend its abilities greatly.

```
## get line number of match
> grep -n cat hello.txt

## get line numbers without match
> grep -nv cat hello.txt

## ingore case during match
> grep -i “CAT” hello.txt
```

More on grep

grep allows all sorts of wildcards

```
## match with wild-wild-’t’
> grep “..t” hello.txt

## match w/ c or b using brackets
> grep [cb]at hello.txt

## match only if starts with match
> ls /bin/ | grep “^z”

## -l returns filenames with match
## here find all txt files with cat
> grep -l “cat” *.txt
```

sed for substitution

A powerful tool for replacing text with another text

```
## general syntax (don’t run)
sed ‘s/<pattern>/<replace>/g’ file

## example, replace dog with bark
> sed ‘s/dog/bark/g’ hello.txt

## save output with redirection
> sed ‘s/dog/bark/g’ hello.txt > bark.txt

## or -i to change file in-place
> sed -i ‘s/dog/bark/g’ hello.txt
```

A few more fun tools

cut, sort, uniq

```
## cut can sample bytes from text
> echo longstring | cut -b 1-4

## or sample delimited fields from text
> echo -e “a,b,c\nd,e,f”
> echo -e “a,b,c\nd,e,f” | cut -d ‘,’ -f 2

## Use ‘\’ to break long commands across
## multiple lines.
> echo “a,b,c,d” | \
  cut -d ‘,’ -f 2-3 --output-delimiter=’\t’
```

A few more fun tools

cut, sort, uniq

```
## sort works on lines of text
> echo -e “z,y\na,b\nj,k” | sort

## reverse sort
> echo -e “z,y\na,b\nj,k” | sort -r

## random order
> echo -e “z,y\na,b\nj,k” | sort -R
```

A few more fun tools

cut, sort, uniq

```
## uniq removes duplicates
> echo -e “z,y\na,b\nj,k” | sort

## TEST: try to sort the following text
## by the number of occurrences and output
## as tab-delimited [n-occurrences]\t[text]
> test=“z,y\na,b\na,b\na,b\nj,k\nj,k”
```

A few more fun tools

cut, sort, uniq

```
## uniq removes duplicates
> echo -e “z,y\na,b\nj,k” | sort

## TEST: try to sort the following text
## by the number of occurrences and output
## as tab-delimited [n-occurrences]\t[text]
> test=“z,y\na,b\na,b\na,b\nj,k\nj,k”

## one possible answer:
> echo -e $test | uniq -c | sort -r | \
  cut -d “ “ -f 7- --output-delimiter=”\t”
```

A few more fun tools

curl, less, gzip, zcat

```
## curl downloads from the internet
> curl eaton-lab.org/pdsb/test.fastq.gz

## (z)less let's you peek at BIG files
> zless test.fastq.gz
> zcat test.fastq.gz

## gzip/gunzip compression of files
> gunzip test.fastq.gz
> gzip test.fastq
```

Test

Count how many reads there are from each sample in the data file.
Hint: zcat, grep, cut, uniq, sort

```
## pipe together several commands
> zcat test.fastq.gz | ... | ... | ...
```

Test

Count how many reads there are from each sample in the data file.
Hint: zcat, grep, cut, uniq, sort

```
## five commands piped together
> zcat test.fastq.gz | \
    grep '^@[0-9]' | \
    cut -d '.' -f 1 \
    uniq -c \
    sort -nr
```

Markdown: simple text formatting

You'll use this to format the first homework assignment
Click here for an example

Today we learned: bash basics

The basic command line utilities that allow efficient access to the file system, editing files, and monitoring resources

```
## where am I?
> pwd

## make a new directory
> mkdir PDSB

## move into that directory
> cd PDSB

## show what is in new dir
> ls -lt
```

Next session we'll learn: open science tools

jupyter, rstudio, git, and github together provide tools for hosting, sharing, and collaboratively editing code.

```
## You will learn to access course
## materials from online using git
> git clone
https://github.com/programming-for-bio/

## and to submit your homework w/ git
> git add assignment-1.ipynb
> git commit -m “assignment 1 uploaded”
> git push origin master

## and collaborate with other course
## members on projects
> git merge new-branch my-branch
```

Assignments and readings for next time

Assignment: Link to Session 1 repo
Readings: See in the repo above
Collaborate: Work together in this gitter chatroom