Welcome to BCB/EEOB546!

Computational Skills for Biological Data

Instructors:

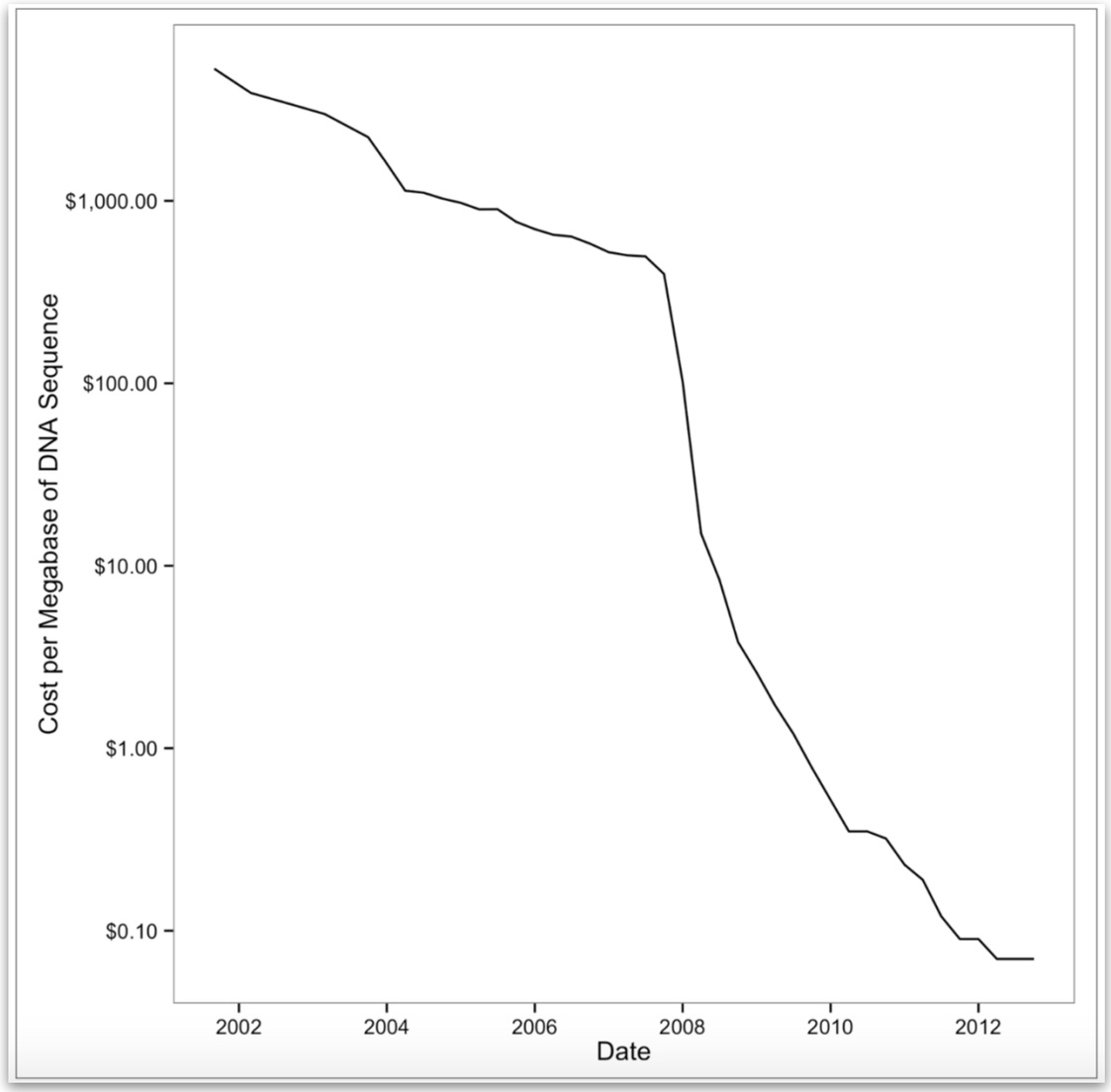
Matt Hufford

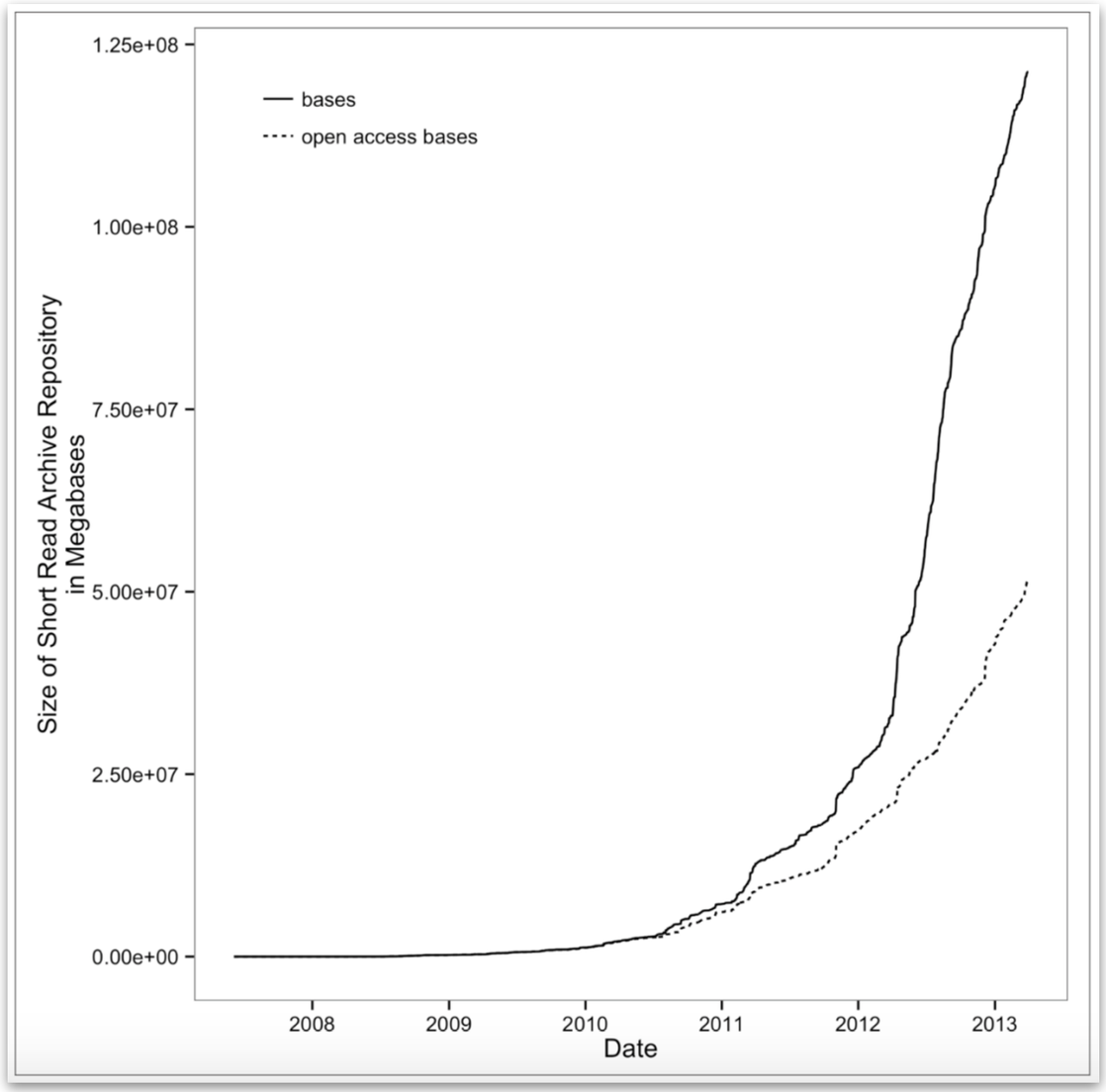
Corrinne Grover

Dennis Lavrov

Introduction and Basic Unix

What motivated us to teach this class?





Introduction and Basic Unix

What motivated you to take this class?

- Many of you likely have interest in more specific applications (e.g., transcriptomics, formal sequence analysis, GWAS, etc…)
  - This course will focus on basic skills that will be necessary for working with large data sets and will be useful in these applications…it's a first step
- You all are drinking from the data firehose!

Introduction and Basic Unix

Our Objectives

By the end of this course, you should:

- Navigate through your computer, create and modify files and directories, and process data using basic Unix commands

- Become familiar with basic R syntax and data structures and implement these in data analysis and plotting.

- Utilize the Python scripting language for more sophisticated data processing.

Introduction and Basic Unix

Our Objectives

By the end of this course, you should:

- Become familiar with various genomic data types (range, sequence, and alignment data) and learn how to write scripts and analysis pipelines for working with these data.

- Become familiar with high performance computing resources at Iowa State as well as how and when to employ these resources.

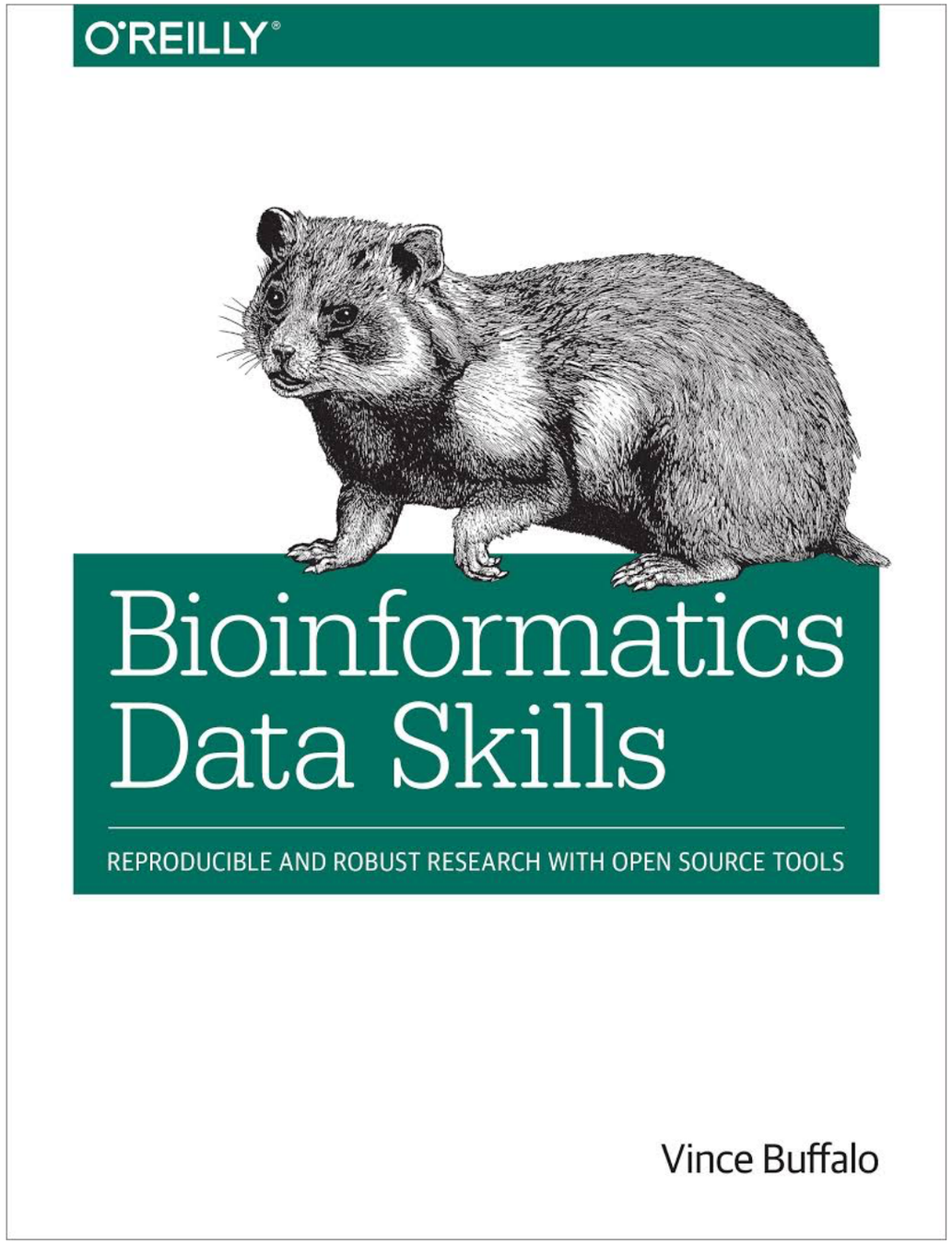
- Explore additional resources/topics in computational biology including manuscript preparation in LaTeX and Overleaf and creation of NSF-style Data Management Plans.

Introduction and Basic Unix

Our Textbooks

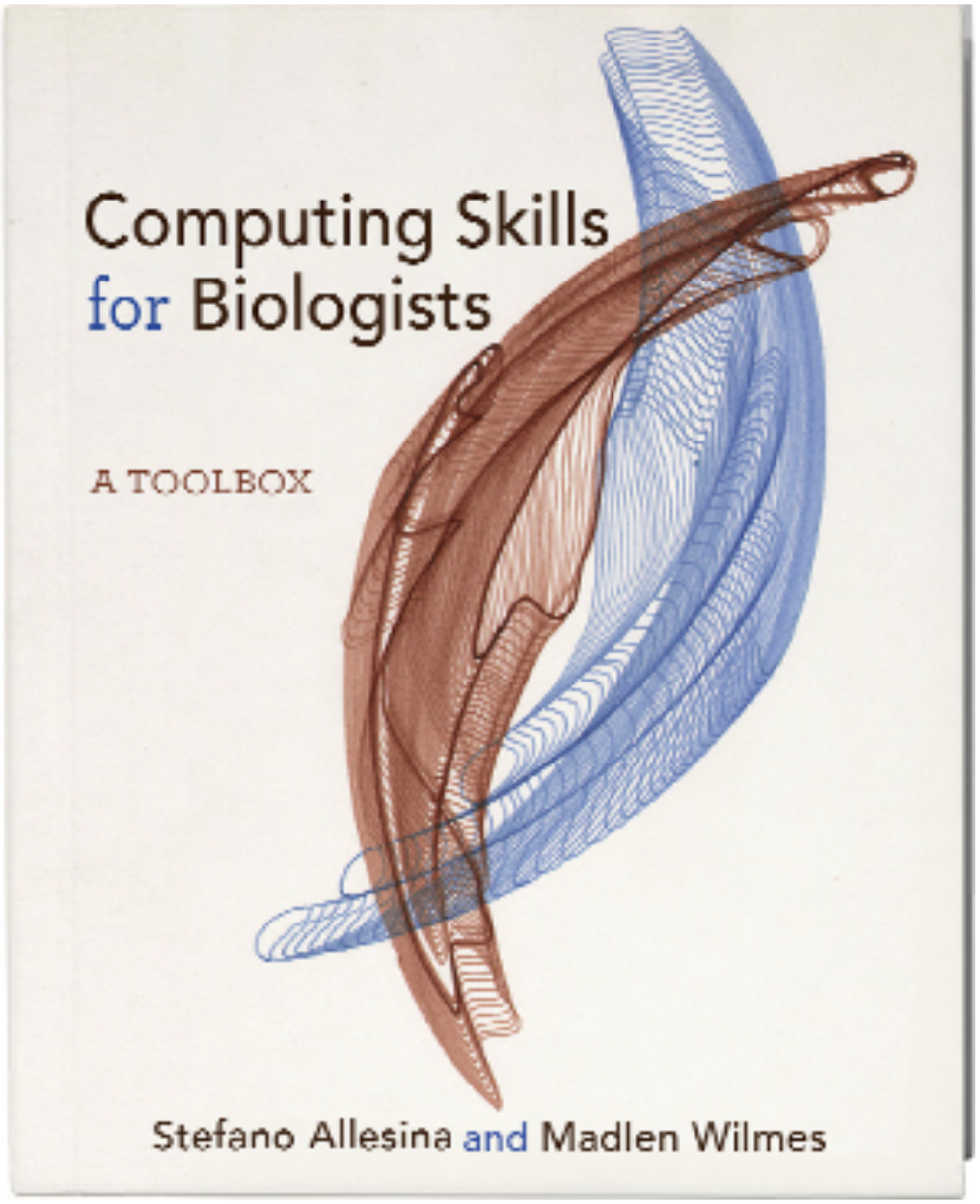
- Written to help address sudden need in biology to be able to handle Big Data

- Available through Amazon (hard copy), O'Reilly (hard copy and eBook), and ISU Library (eBook, FREE!!)



Introduction and Basic Unix

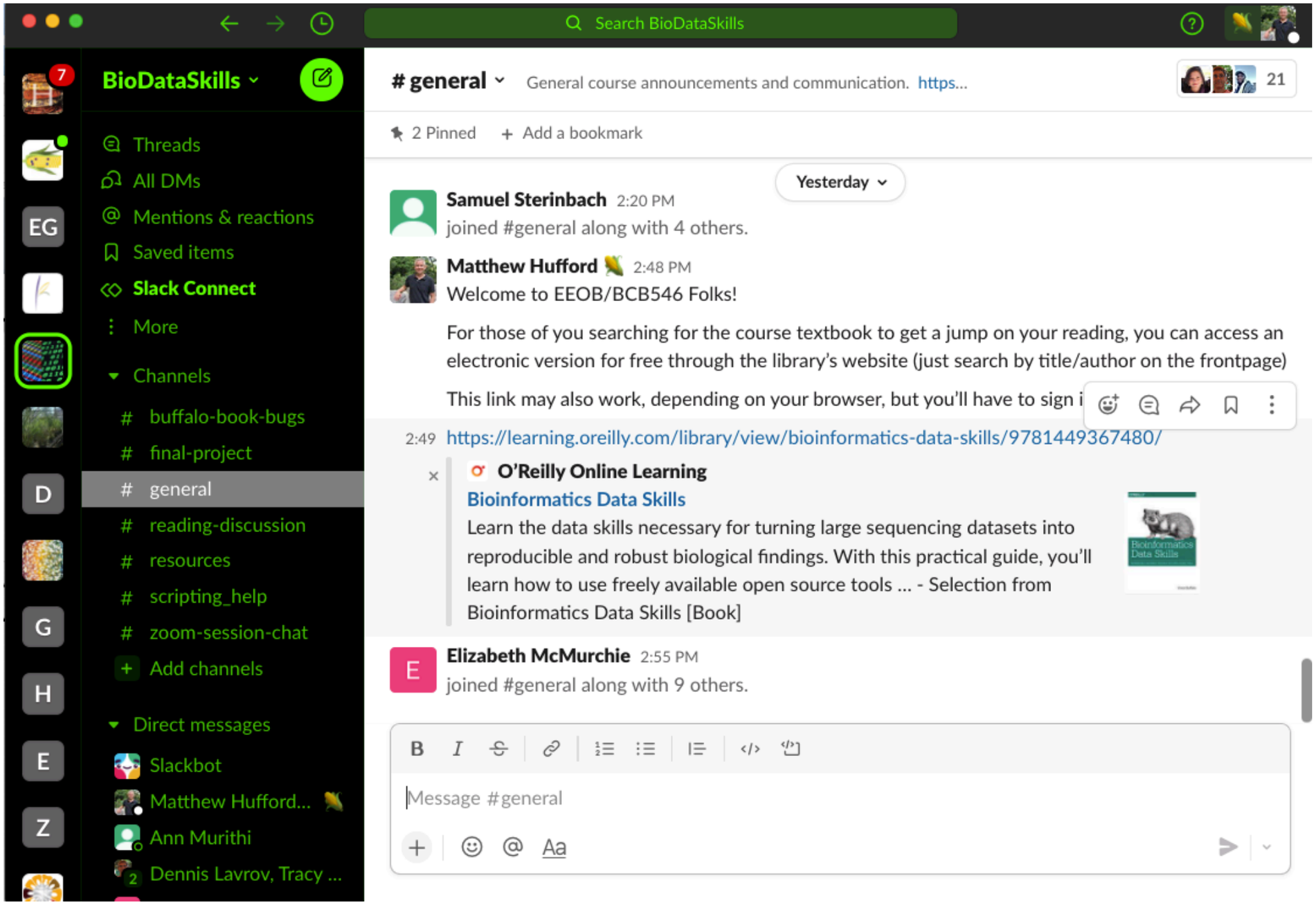
Our Textbooks



Introduction and Basic Unix

How will we communicate?

Slack



Introduction and Basic Unix

What is our schedule?

Google Sheet

https://docs.google.com/spreadsheets/d/1uTrA6o7Xk-6TK-KvEIHFRm6SYvOr5pon3GSvmNsvOUw/edit?gid=0#gid=0

Introduction and Basic Unix

How will grades be assigned?

Grading:

Assignment 1: Unix	15%
Assignment 2: R	15%
Assignment 3: Python	15%
Assignment 4: Data Management Plan	15%
Group Project and Presentation	40%

Chapter 1

- Our two main goals in bioinformatics are to have research that is *reproducible* and *robust*

- How can we make our analysis reproducible?

- How can we make our analysis robust?

Chapter 1

- ✦ Writing code for humans makes it reproducible, but it must still be readable by your computer

```perl
#! user/bin/perl
# PatHaps.pl by Matthew Hufford
use strict;
use warnings;


                                        #define a hash of allele frequencies from the bulked seed
open AF, "<AllFreq.txt" or die "fail!\n\n";
my %data;
while (<AF>) {                          #always name "IN" file uniquely to avoid corrupting files
    chomp;                              # removing endlines
    my $AllFreq = $_;
    my ($locus, $allele, $freq) = split("\t", $AllFreq);
    $data{$locus}->{$allele}=$freq;     #creating a hash of hashes--locus and allele are keys to the $freq value
}
close AF;


                                        #read in the set of dad haplotypes from file "PatGenosIN.txt"
open PG, "<PatGenosIN(-9).txt" or die "fail\n\n";
my ($patallele_id, @genos) = <PG>;      #first line turned into string, rest of file turned into an array
close PG;
print `cp Pools2Format.txt outfile.txt`;
my @patallele_id = split("\t", $patallele_id);      #creating array of loci names from string of first line
open OUT, ">>outfile.txt";
print (OUT "\n");

foreach my $dad (@genos) {
    chomp $dad;
    my ($dadID, @linedata) = split ("\t", $dad);            #splitting paternal ID from array of genotypes
    print (OUT "$dadID\t0\t0\t");                               #printing dad ID with a tab

    for my $a (0..$#linedata) {                             #creating a list from 0 to the number of scalars in each line of the array; the $# notation indicates the total number in arra
        my $allele=$linedata[$a];                           #putting list into the array and creating a string of allele calls that are at those positions.

        if ($allele =~ m/&/) {                              #matching all alleles that contain an ampersand
            my ($allele1, $allele2) = split ("&", $allele); #creating an array of the two possible alleles
            my $freq1=$data{$patallele_id[$a]}->{$allele1}; #creating frequency scalar for each allele 1
            my $freq2=$data{$patallele_id[$a]}->{$allele2}; #creating frequency scalar for each allele 2

            if (rand()<($freq1/($freq1+$freq2))) {          #assessing whether random number between 0 and 1 is less than freq 1
                print (OUT "$allele1\t\-9\t");
            }
            else {
                print (OUT "$allele2\t\-9\t");
            }
```

Chapter 1

- Adding in tests for your code helps avoid the dreaded silent errors and makes your research more robust

```
def add(x, y):
"""Add two things together.""" return x + y

def test_add():
"""Test that the add() function works for a variety of numeric types.""" assert(add(2, 3) == 5)
assert(add(-2, 3) == 1)
assert(add(-1, -1) == -2)
assert(abs(add(2.4, 0.1) - 2.5) < EPS)
```

Chapter 1

- If a library already exists for what you want to do, why not use it?
- Do not modify your raw data directly (treat as "Read Only")
- If you're going to use a script multiple times, turn it into a tool:
  - document it
  - create versions
  - make your command-line arguments clear
  - sharing in a version-controlled repository

Chapter 1

- Publish both your scripts and data
- Also publish your documentation and document everything!
- *What's the difference between documenting a script and a project? How might we do both?*
- Make an analysis and the figures showing the results of an analysis the product of a script

Intro. to Computational Methods
UNIX

- UNIX is an operating system originally developed by AT&T's Bell Labs in the 1960's (then Novell, then The Open Group)

- "Operating System" = Suite of programs that make your computer work

- macOS is one flavor of UNIX; others are Linux, Solaris, BSD

Intro. to Computational Methods

UNIX

The UNIX OS has three components:

(1) The Kernel: OS Hub; allocates memory and time

(2) The Shell: Interface between user and the kernel; the shell searches for command files called by user and passes requests to the kernel

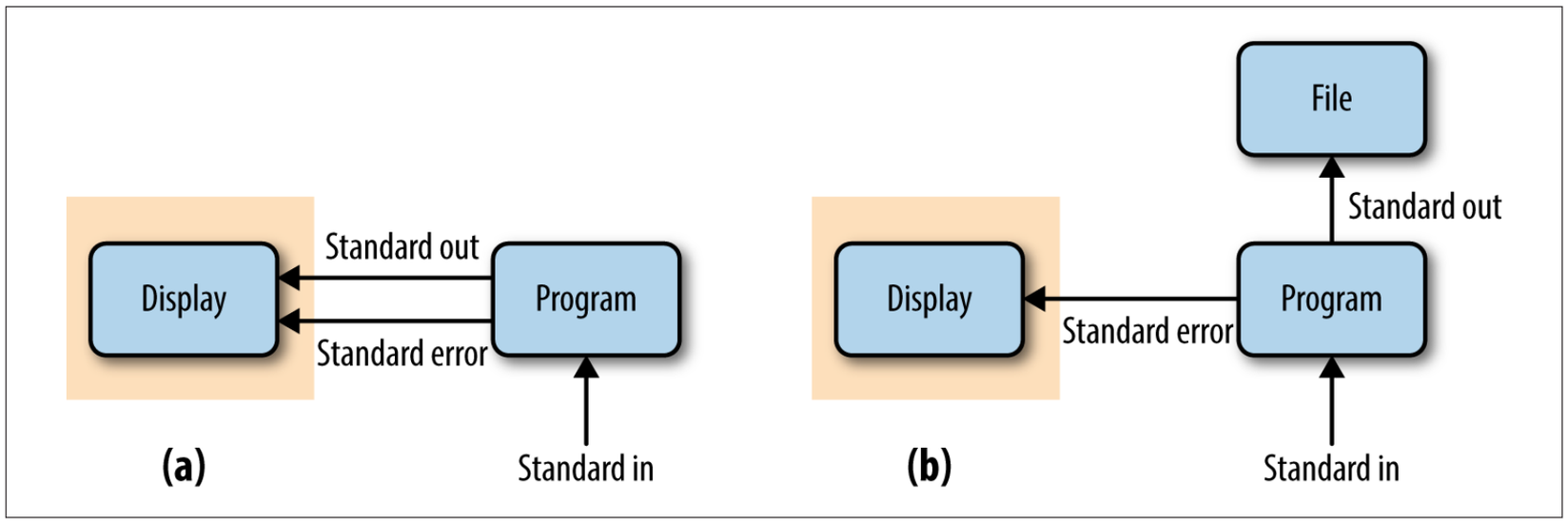
(3) Programs: Commands called by the user

Intro. to Computational Methods

UNIX

- UNIX is modular: What does this mean?
- UNIX handles data as a stream
- A given program generates standard output and standard error streams: What is the difference?

- How can we redirect streams?



Introduction and Basic Unix

Our Computational Goals for Today

1. Make sure everyone has a Shell solution

2. Installation of GitBash if necessary

3. Clone the Git repository for the textbook and the course

4. Work through a Basic Unix example

Introduction and Basic Unix

Where to from here?

1. If the basic Unix commands in our example were all new (and even if they weren't!), you should consider working through the Unix portions of these tutorials :
   https://sites.google.com/site/eeob563/computer-labs/Lab-1
   http://korflab.ucdavis.edu/Unix_and_Perl/

2. If you haven't already, read Chapters 1-3 of Buffalo
   - For Chapter 1, create a text snippet in Slack with a few favorite points and any questions on points that were not clear, and we'll discuss these on Friday
   - We'll also discuss and work through examples from Chapter 3 on Friday