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 arr
        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

- Mac OSX 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?