CMSC423: Bioinformatic Algorithms, Databases and Tools
Lecture 3

Perl/Perl Modules

BioPerl

Bioinformatics applications

WHAT

- Sequencing
- Genome assembly
- Genome finishing
- Gene finding
- Gene annotation
- Pathway analysis
- Comparative analyses

- Gene expression studies
- Proteomics
- Genotyping
- ...

WHERE

- one organism
- mixture of organisms
- environments
- host/pathogen interactions
- single cells
- tissues
- whole organism/genome

Intro to Perl

- Just like C but worse
- Unfortunately: most widely used programming language in bioinformatics
- Why?
  - Easy to hack code together
  - Very easy to use regular expressions
  - Easy handling of file I/O
- The bad: very easy to write code nobody else will understand – not even you.

Alternatives to Perl

- Python – most serious candidate sofar
- Ruby
- Java
- C++

- You may use any of these for your projects/homeworks unless specifically required to use Perl by the assignment

Perl basics

- Getting help
  - ‘man perlfunc’ : help on Perl functions
  - ‘man perlop’ : help on Perl operators
  - ‘man perldata’ : help on Perl datastructures
  - ‘man perlre’ : help on Perl regular expressions
  - ...
  - ‘man perl’ : lists an overview of all documentation and tutorials available
  - ‘perldoc’ : read documentation embedded in the code – more on that later

A simple perl script

```
#!/usr/bin/perl –w          # warnings "on"
use strict;                 # declare variables

my $i = 0;
my $i = "Hello world\n";    # don't do this
my $text = "Printing number ";

for ($i = 0; $i < 10; $i++)
{
     print $text, $i, "\n";
}

printf("%s\n", $i);

exit(0);
```

Perl datastructures - arrays

```
my @numbers = ();

for (my $i = 0; $i < 100; $i++) {
	$numbers[100 - $i – 1] = $i;
}

for (my $i = 0; $i <= $#numbers; $i++){
	print $numbers[$i];
	if (($i + 1) % 10 == 0){
		print "\n";
	} else {
		print "\t";
	}
}
```

```
99  98  97  96  95  94  93  92  91  90
89  88  87  86  85  84  83  82  81  80
...
```

Perl datastructures – hash tables

```
my %ages = ();

$ages{'Bob'} = 27;
$ages{'Alice'} = 16;
$ages{'Mike'} = 22;

foreach my $name (keys %ages) {
	print "$name $ages{$name}\n";
}

print "\n\n";

my @names = keys %ages;
@names = sort {$a cmp $b} @names;

foreach $name (@names){
	print "$name $ages{$name}\n";
}
```

```
Bob 27
Mike 22
Alice 16


Alice 16
Bob 27
Mike 22
```

Note: 'values' returns the array of values for the elements in the hash

Regular expressions

```
while (<STDIN>){
       if ($_ =~ /[AG]*T+C{3,5}(.*)AAG?/){
              print "Gotcha: $1\n";
       } else {
              print "no luck\n";
       }
}
```

```
AGGAGCCCAGCCACAAG
TCCCCCAGCAGGCTAA
GGGGGGGGGTTTTTTTTTTCCCCCACGGCTAAG
AGTCCGACCTAAG

no luck
Gotcha: AGCAGGCT
Gotcha: ACGGCT
no luck
```

Using modules

```
#!/usr/bin/perl

# use lib "/usr/lib/perl5/5.8"

use Getopt::Long;

$err = GetOptions( "h|help" => \$help,
                   "D=s"    => \$database,
                   "N|number=i" => \$number );

if ($err == 0) {
       print "command line parsing failed\n";
       exit(1);
}

if ($help == 1) {
       print "program [-h] –D <db> -number <n>\n";
}
```

Remember: perldoc Getopt::Long