Reading and Writing

- Input data from files
- Read data from a program
- Print data into a file or print data out to a program

Input/Ouput

```
1 open(IN, "input.txt") || die $!;
2 # read a line in
3 my $line = <IN>;
4 # read the whole file
5 while(<IN>) {
6  my ($col1, $col2) = split;
7 }
8 close(IN);
```

Filehandles

Filehandles can also be stored in variables

```
 1 my $fh;
 2 open($fh => "gene.dat") || die $!;
 3 while(<$fh>) {
 4  print $_;
 5 }
 6 #I like to use this in one line
 7 open(my $fh2 => "gene2.dat") || die $!;
 8 while(<$fh2>) {
 9  print $_;
10 }
```

Writing to a file

One can also write data out to a file

```
1 open(my $fh => ">outputfile.txt") || die $!;
2 print $fh join("\t", qw(NAME RANK TOWN)), "\n";
3 print $fh join("\t", qw(GRIFFITH SHERIFF MAYBERRY)), "\n";
4 print $fh join("\t", qw(RAWLS DEPUTY BALTIMORE)), "\n";
5 close($fh);
```

Data embedded in a script

```
1 while(<DATA>) {
2  my ($col1,$col2) = split(/\s+/,$_);
3 }
4
5 __DATA__
6 Color  Size Model
7 red    10   Jumbo
8 yellow 8    Large
9 pink   2    Mini
```

Pipes for processes

You can combine operations that are on the command line with the `|` operator in UNIX. This can also be used in Perl when specifying an `open` command to actually run a program and have the output sent back to the Perl program. This can be used to obtain data from a program. For example here we run the grep cmmand to find lines in a file that have the string 'gene'. Only those lines will be returned and thus be seen by the Perl program.

```
1 open(IN,"grep 'gene' gene.dat | ") || die $!;
2 while(my $line = <IN>) {
3  print "line is $line\n";
4 }
```

Or it can be used to send data to a program. For example this script will print out data to a program which will then compress the data. Notice how the pipe comes at the beginning because we plan to send data into the program.

```
1 open(my $fh => "| gzip -c > file.gz") || die $!;
2 print "hello there\n";
3 print "can you see this?\n";
```

Now look at the file in your directory. It is compressed - you can read it with `more` and see it is binary file. However you can read it with `zmore` or you can uncompress it with `gunzip`.

Pipe trick

Can use it to open a compressed file on the fly.

```
1 open(my $fh => "zcat file.gz |") || die $!;
2 while(<$fh>) {
3  # process data in a file that was compressed, without making a new copy of the file as un
4 }
```

Read data from the web with cmdline

```
 1 my $url = 'http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=1636
 2 # -S option will not print any statistics
 3 open(my $fh => "curl -S '$url' |") || die $!;
 4 while(<$fh>) {
 5   print $_;
 6 }
 7
 8 open($fh => "GET '$url' |") || die $!;
 9 while(<$fh>) {
10   print $_;
11 }
```

Let's try this together

Login to biocluster, Download data files. Data are in this http://courses.stajich.org/public/gen220/data/ which represent some time points in growth for a fungus.

http://courses.stajich.org/public/gen220/data/Nc20H.expr.tab http://courses.stajich.org/public/gen220/data/Nc3H.expr.tab

```
(on biocluster)
wget http://courses.stajich.org/public/gen220/data/Nc3H.expr.tab
```

Write a script to read in the Nc20H and Nc3H data into a hash (one hash for each datafile). Store in the hash the gene name (the 1st column) and the FPKM data. Each gene will appear once in each file.

- Print out a new file which has the gene name, the expression in 3H and the expression in 20H.
- Extra - print it out so that it is sorted by the 3HR timepoint

A solution

```
 1 use strict;
 2 use warnings;
 3 my (%expr3H,%expr20H);
 4 open(my $fh => 'Nc3H.expr.tab') || die $!;
 5 while(<$fh>) {
 6  my @row = split("\t",$_);
 7  next if $row[0] eq 'gene_id'; # skip when it is the header line
 8  $expr3H{$row[0]} = $row[5];
 9 }
10
11 open($fh => 'Nc20H.expr.tab') || die $!;
12 while(<$fh>) {
13  my @row = split("\t",$_);
14  next if $row[0] eq 'gene_id'; # skip when it is the header line
15  $expr20H{$row[0]} = $row[5];
16 }
17
18 open(my $outfh => ">Combined.tab") || die $!;
19 my $gene;
20 for $gene ( keys %expr3H) {
21  print $outfh join("\t", $gene, $expr3H{$gene}, $expr20H{$gene}), "\n";
22 }
23
24 open($outfh => ">Combined_sorted.tab") || die $!;
25 for my $gene ( sort { $expr3H{$b} <=> $expr3H{$a} } keys %expr3H) {
26  print $outfh join("\t", $gene, $expr3H{$gene}, $expr20H{$gene}), "\n";
27 }
```

References

Reference are ways to refer to a complicated data structures as a single, scalar value. This lets one pass around multiple arrays and they stay separate. We also primarily use reference to store multiple things in a slot in an array.

- Reference to an array is done with `\` or `[]`
- Reference to a hash is done with `\` or `{}`

For example this lets one pass around multiple arrays and they aren't flattened into one. Consider this code.

```
1 my @array1 = qw(A B C D);
2 my @array2 = qw(W X Y Z);
3 my @array3 = (@array1, @array2);
4
5 print join(",", @array3), "\n";
```

Storing multiple items in an Array

```
 1 my $url = 'http://courses.stajich.org/public/gen220/data/Ncrassa_OR74A_InterproDomains.ta
 2 open(my $fh => "GET $url |") || die $!;
 3 my %genes;
 4 while(<$fh>) {
 5     my ($gene,$domain, $domain_name, $start,$end,$score) = split;
 6 # store an array as the value for each key by making it a reference to an array
 7 # using the @{$genes{$gene}} which is forcing what is the value
 8 # to be an array reference. Then we use push to add something to
 9 # this array
10 # Because perl will automatically initialize the value, based on the context
11 # we DON'T need to do this, but it is what is happening under the hood
12 # if this is the first time accessing this key
13 # $genes{$gene} = [];
14 push @{$genes{$gene}}, $domain_name;
15 }
16 # now unpack to print this out
17 for my $gene ( keys %genes ) {
18     my @domains = @{$genes{$gene}};
19 print join("\t", $gene, join(",", @domains)), "\n";
20 }
```

Subroutines

```
1 sub a_routine {
2  my @args = @_; # the arguments passed in are avaialable as @_;
3  print "the arguments are ", join(",", @args), "\n";
4 }
5 &a_routine('a','b','c');
```

Command line arguments

```
$ perl myscript.pl A B C

!perl
print join(",", @ARGS), "\n";
print "the first argument is ", $ARGS[0], "\n";

A,B,C
```

Use this to specify a data file to read in, or specific options you want to run.

Practice

Write a script that will open and print out the first 5 lines of a file. The name of the file to open should be passed in on the command line as the first argument.