File Operations

Aims

- Understand how to read a file into a program
- Understand how to write out data to a file
- Why File ::Slurp is good
- What a filehandle is, and how to use one
- How to append as well as write a new file

File operations

- Your end user is not going to want to provide everything on the command line, or permanently interact with a running script
- Also, they are not going to want to 'screen scrape' the end data
- Most data will come from/stored into a file or a database
- Database interaction beyond the scope of this course

File ::Slurp

- There is a great module of code you can use to help with read and write file operations

  use File ::Slurp;
- We get the methods *read_file* and *write_file*
- We'll look at both, and the filehandle versions to compare

Reading a file – File ::Slurp

- Let's start with reading a file

  bin/01-reading_a_file.pl
- Slurp in a whole book

  my $slobbit = read_file('data/the_slobbit');
- Now we have some data in the program, we can view it

  say $slobbit;

Reading a file – File ::Slurp

- we can do something to it
- Nobody wants me to be the hero of the story - *$slobbit* is just a string

  $slobbit =~ s/Andy/Bilbo/gm;

  say 'Hero changed';

  say $slobbit;
- but we haven't changed the file - take a look at *data/the_slobbit*

Reading a file – File ::Slurp

- our data may be in some table like format, so we probably don't want it all in one scalar
- slurp in a qseq file, with rows going into an array

  my @reads = read_file('data/1234_1_qseq.txt');

  say @reads;

Reading a file – File ::Slurp

- We want just the DNA sequences

```
foreach my $read ( @reads ) {
  my @data = split /\s+/m, $read;
  say $data[8];
}
```

- In both of these cases, we are going to need to watch the memory, as all the file is being read in at once.

Reading a file - Filehandle

- In many cases, always take advantage of File::Slurp
- However, if you want to do it completely by yourself, you need to go through some steps and open a *filehandle*
- A *filehandle* is a *scalar reference* to the file, which allows you to read/write to the file

Reading a file - Filehandle

- open a file handle to the file you want to read

```
open my $fh, '<', 'data/the_slobbit' or die 'Could not open data/the_slobbit for reading';
```

- *$fh* is a file handle
- *'<'* means for reading only
- *'data/the_slobbit'* is the filename
- always give an option to do something if the file can't be opened, in this case die

Reading a file - Filehandle

```
while ( <$fh> ) {
  s/Gran/Uncle/xms;
  print;
}
```

- The file handle can act just like an array, so we can (for example) loop on it, processing a line at a time

Reading a file - Filehandle

```
close $fh or die 'Could not close the filehandle for file data/the_slobbit';
```

- You must close the filehandle, with an option to do something if it can't close (this is unlikely to happen)

Reading a file - Filehandle

- The advantage of this is that you will only read out of the file the next line to process, so in the case of a million+ line fastq file, you won't fill your memory up.
- The disadvantage is that your files may become altered/vanish/locked during this time, causing problems either for your program, or other users.

Writing a file

- we looked at the data source file, but you are going to want your results to go somewhere else, and are likely to want to store them in a file.

  bin/02-writing_a_file.pl
- In this script is an array of sequences that we will want to write out to a file (they would have been generated in some way)

Writing a file – File::Slurp

```
write_file( 'data/short_reads.seq', @sequences );
```

- check the file *data/short_reads.seq*. You'll see all the reads are now in there, ready to be passed on, archived, processed further...

Writing a file – filehandle

- As with reading, there is the full way of doing it. This is very useful if you need to write periodically (e.g. a log file) or you need to write specific data, or just modify data in the file
- first, direct equivalent to write_file

Writing a file – filehandle

- open a file handle to the file you want to write to

```
open my $fh, '>', 'data/my_book' or die 'could not open data/my_book';
```

- *$fh* as with reading, we need a file handle
- *'>'* this means for writing, overwriting any existing file with this name
- *'data/my_book'* the file we want to write into
- *or die* again always error handle an open

Writing a file – filehandle

```
foreach my $seq ( @sequences ) {
  print {$fh} 'Andy ' . $seq or die 'Unable to print to filehande: ' . $fh;
}
```

- In this loop, as we process the elements, we print to the filehandle our data.
- Advantage of this, if it takes a lot of processing to create each sequence, then we can print as we generate, rather than collect them all in one go

Writing a file – filehandle

print {$fh} 'Andy ' ;

- we put the filehandle in *{ }* so that the interpreter knows that it is 'where to print to', rather than needing to check at runtime if it is a variable to print, or something to print to

close $fh or die 'could not close filehandle: ' . $fh;

- Exactly as with reading, we close the file handle

Appending to a file – filehandle

- If you want to append to a file (example, you have a script which runs every hour on a cron, then you don't want to kill data generated in previous hours)

open $fh, '>>', 'data/my_book' or die 'could not open data/my_book';

- *'>>'* this means for appending, creating the file if it did not already exist

Appending to a file – filehandle

foreach my $seq (@sequences) {

print {$fh} 'James ' . $seq or die 'Unable to print to filehande: ' . $fh;

}

- We'll add James this time, so that we can see it has been added

close $fh or die 'could not close filehandle: ' . $fh;

- We close again

File Operations - Summary

- We have seen how to read a file into the program with write_file from File ::Slurp and using a filehandle
- We have seen how to write to a file from the program with read_file from File ::Slurp and using a filehandle
- We have seen how to append to a file
- We have touched upon when a filehandle might be better – large amounts of data

File Operations - Workshop

- 09-file_operations/workshop
- bin/01-reading_and_writing_workshop.pl
- using the file given file write a script that will obtain the sequence and the quality scores, generate a unique name for each read from other information, and then output out in the format
- # read_name
- # sequence
- # quality
- into another file in the output directory