Shell scripts

- Used to collect multiple commands into a single program
- Run the same commands again or on other data
- Also used to document the steps and describe the thought process! Lab book for analysis.
- Finding the right level of detail is challenging: not too much, not too little.

Bash may be your first programming language

- The bash terminal is a programming language
- It is used to launch and manage processes
- Shell meta characters allow pattern matching:
  - `?`: any one character
  - `*`: any characters
- Shell programming constructs: loops and conditionals
- “tab completion” – this is how you avoid typos

The shell is not a good algorithm development language

- The shell syntax is “ancient” – from the beginning of computing
- Excellent at launching programs and simple interactions
- Terrible at algorithms
- Most languages: Perl, Python, Ruby exist to replace the shell
Build your script one step at a time

```bash
# Comments start with the # symbol.
# Make comments short, concise, full sentences.
# We need comments because we may forget
# what our code does.
# This code demonstrates helpful shell constructs.
# A shell script contains commands you might want
# to run again later.
# Get the dataset from SRA.
fastq-dump -X 150000 --split-files SRR519926
# Run fastqc reports on the SRA run
fastqc SRR519926_1.fastq
```

Run your shell script

```bash
ialbert@grit ~/appbio/lec10
$ bash ebola-qc.sh
```

Refine your script to isolate variable and static parts

```bash
# It is not good to repeat the same
# information because we need to remember
# to change them all, that can be error prone.
# We can create variables to show what
# parts are subject to change.
RUN=SRR519926
# Remind the user what is going on.
echo "Dumping fastq for $RUN then running fastqc on it"
# Get the dataset from SRA.
fastq-dump -X 150000 --split-files $RUN
# Run fastqc reports on the data.
fastqc $RUN_1.fastq
```

Parametrize the script – move variable parts into parameters

```bash
# It is not good to repeat the same
# information because we need to remember
# to change them all, that can be error prone.
# We can create variables to show what
# parts are subject to change.
RUN=SRR519926
# Get the dataset from SRA.
fastq-dump -X 150000 --split-files $RUN
# Run fastqc reports on the data.
fastqc ${RUN}_1.fastq
```
Parameterize the script even more – get values from command line

```bash
# The first script parameter will be the run id.
# For example SRR519926.
RUN=$1

# Remind the user what is going on.
echo "Dumping fastq for $RUN then running fastqc on it"

# Get the dataset from SRA.
fastq-dump -X 15000 --split-files $RUN

# Run fastqc reports on the data.
fastqc $RUN_1.fastq
```

The script can now take a parameter from command line

```
jalbert@grit ~/appbio/lec10
$ bash ebola-qc.sh SRR519926
```

Read a file in bash

```
# A few caveats: Bash Scripting & Read File line by line

INPUT=$1
while read -r line;
do
  echo "This is a line : $line"
done < $INPUT
```
Bash has lots of features

We will slowly introduce some features along the way. It is not a particularly good programming language. Don’t try to write overly complex programs. There are better solutions for that.

Other advanced features: build functions

It uses functions to isolate different tasks, then it is easier to create custom tasks by chaining together various building blocks.

The art of data analysis: creating the right building blocks.

Homework 10

Write a script that takes two parameters

1. a Bioproject ID
2. an average read quality value

The script should do the following:

1. Download 25,000 reads from all (or up to 10) experiments from the project.
2. Generate fastqc reports on all these files.
3. Run a quality control on each of the files to filter for the average quality value.

Submit only the printout of the script.

Imagine that this script will be used by someone else and they need to understand who wrote it and what it does, what the inputs and outputs are.