Bioinformatics shell scripting

Because bioinformatics work involves a lot of tedious data file processing, you will want to pipeline things together to process many files. It's important to do this in a reproducible and robust way.

- ***We will learn how to write re-runnable bash scripts and automate file processing tasks with find and zargs***
- ***We are going to do all of these steps on our local computers at the terminal. Download the BDS chapter 12 data to your desktop, open a terminal window, and change directory to the chapt 12 data.***

Basic bash scripting

Most bash scripts are just commands organized into a re-runnable script with some added bells and whistles--do the files exist? Any errors?

Writing and running robust bash scripts

Bash scripts by convention have the .sh extension. They are created in text editors at the command line. The header of a bash script looks like this:

```
#!/bin/bash #This is the shebang! line that indicates the path to the
interpreter used to execute the script
set -e #This terminates the script if any command exited with a nonzero exit
status
set -u #prevents the script from running if we have forgotten to set a variable
set -o pipefail #Another step to ensure that the script will fail if *any*
program included in the pipe fail
```

To run a bash script, you need to execute it with the following command

```
$   chmod u+x script.sh #This makes the script executable
$   ./script.sh #This runs the script
```

Variables and command arguments

You can store settings as variables -- for example, which directories to store results in, parameter values for commands, input files, etc -- and rather than having to change hardcoded values in your scripts, using variables to store settings means you only have to change one value -- the variable

Open a terminal window, change directories to the Desktop, and type in below commands.

```
$   sample="CNTRL01A"
$   mkdir "${sample}_aln/"
```

What happened?

How does bash handle command line arguments?

Now use your terminal editor to make a simple bash script (called args.sh) with the following lines:

```
#!/bin/bash
echo "script name: $0"
echo "first arg: $1"
echo "second arg: $2"
echo "third arg: $3"
```

At command line, run the following:

```
$ bash args.sh arg1 arg2 arg3
```

Now let's add a conditional stating that we want the script to fail if we don't input enough arguments at the command line. Open the bash script and add in the following:

```
#!/bin/bash

if [ "$#" -lt 3] #less than 3 arguments?
then
    echo "error: too few arguments, you provided $#, 3 required"
    echo "usage: script.sh arg1 arg2 arg3"
    exit 1
fi

echo "script name: $0"
echo "first arg: $1"
echo "second arg: $2"
echo "third arg: $3"
```

Conditionals in a bash script: if statements

The basic syntax of a bash if conditional statement looks like this:

```
if [command]
then
    [if statements]
else
    [else statements]
fi
```

- Where [command] is a placeholder for any command, set of commands, pipeline, or test condition.
- If the exit status of the command is 0 (i.e., it worked!) then execution continues to the next block
- [if statements] is the placeholder for all statements executed
- [else statements] is a placeholder for all statements executed if [commands] is false (1)

Below is a general set of commands using an if condition

```
#!/bin/bash
if grep "pattern" file1.txt > /dev/null &&
   grep "pattern" file2.txt > /dev/null
then
    echo "found 'pattern' in 'file1.txt' and in 'file2.txt'"
fi
```

Processing files with bash using for loops and globbing

Anatomy of a realistic shell script

```
#!/bin/bash
set -e
set -u
set -o pipefail

#specify the input samples file, where the third column is the path to each
sample fast     #file
sample_info=samples.txt

#our reference
reference=zmays_AGPv3.20.fa

#Create a bash array from the first column, which are simple names. Because
there are   #duplicate sample names (one for each read pair) we call uniq
sample_names=($(cut -f 1 "$sample_info" | uniq))

for sample in ${sample_names[0]}
do
    #create an output file from the sample name
    results_file="${sample}.sam"
    bwa mem $reference ${sample}_R1.fastq ${sample}_R2.fastq > $results_file

done
```

1. First, we load our sample names into a Bash array with sample_info=samples.txt
2. Next, we tell Bash which file to use as our reference for mapping reads
3. We then tell Bash we want to use only the 1st column for our sample_names using the cut command. We also use the uniq command since each file name is represented twice.
4. Then we run a for each do loop -- saying for each *sample* in the array sample_names, create an output file *using* the sample name (i.e. results_file="${sample}.sam")
5. Finally, we have the call for the software we want to run.
6. Effectively, we are storing the name of each file into a variable array, storing the name of the output file based on this name (and giving it the extension .sam), and then running the program for each sample in the array.