H3ABioNet

Pan African Bioinformatics Network for H3Africa

Introduction to BASH

Sumir Panji / Amel Ghouila

Material adapted from: H3ABioNet and WT NGS Bioinformatics Course Africa 2021

BASH

- So far we are using different single commands to do various operations on input data files
- One generally tends to do similar / same operations when new input data files are generated
- Would be nice to organize these different commands to be run once as a script, rather then run each command individually
- BASH (Bourne Again Shell) scripting is suitable for putting these command line commands into a script – mainly command line commands
- BASH is useful, but be aware that they are more modern programming / scripting languages such as Python and R that are suitable for more re-used and complicated tasks as they have numerous inbuilt functions and libraries – mainly used for calling command line programs

Some BASH basics

- One creates a bash script using a text editor such as nano, vim, emacs etc

- A bash script should be saved / ends with a .sh extension

- A bash script should have a header line that provides the path the interpreter to execute the script and let unix know that it is a bash script – pretty much standard on most unix systems: #!/usr/bin/env bash

- Should change the .sh bash script file permissions to be executable e.g. chmod +x bash_script.sh

Some BASH basics

- Let's create a simple bash script to print out "Hello World!" called hello.sh

```
#!/usr/bin/env bash
echo "Hello World"

bash hello.sh
Hello World

./hello.sh
bash: ./hello.sh: Permission denied

chmod +x hello.sh

./hello.sh
Hello World
```

Some BASH basics

- There are a couple of ways to run a bash script:
  - bash script_name.sh
  - ./script_name.sh
  - script_name.sh

- The first option is when the script file does not have executable permissions – we generally do want scripts / program files to be executable

- The second option is when the script file has executable permissions

- The first two ways of running a bash script assume you are in the same directory as the script

- One might want to run the script on multiple files in multiple locations – e.g. script_name.sh

Some BASH basics

- One might want to run the script on multiple files in multiple locations

- Can add the location of the script to your PATH variable so it will be globally accessible while you are logged into your terminal and just type in the script name:

```
pwd
/home/manager/course_data/unix/practical/Notebooks/advanced_bash/scripts

cd ../
/hello.sh
bash: ./hello.sh: No such file or directory

export
PATH=$PATH:/home/manager/course_data/unix/practical/Notebooks/advanced_bash/scripts

hello.sh
Hello World
```

Some BASH basics

- The general format of running a bash script would be script_name.sh input_file

- Will use the input file called test_file

test file line 1
test file line 2
another line
more lines
more lines
penultimate line
last line
pwd

Script name: options.sh
#!/usr/bin/env bash
echo filename is: $1
echo
echo First $2 lines of file $1 are:
head -n $2 $1

Some BASH basics

```
#!/usr/bin/env bash
echo filename is: $1
echo
echo First $2 lines of file $1 are:
head -n $2 $1

./options_example.sh test_file 3
filename is: test_file

First 3 lines of file test_file are:
test file line 1
test file line 2
another line
```

Some BASH basics

```
#!/usr/bin/env bash
echo filename is: $1
# store the file name in a variable called $1 and print out the file name
filename is: test_file

echo
#Print out space for formatting

echo First $2 lines of file $1 are:
# store the number of lines provided by the user in a variable called $2
First 3 lines of file test_file are:

head -n $2 $1
# use the values in the variables $2 and $1 for obtaining the number of lines
entered by the user and print them out
test file line 1
test file line 2
```

Some BASH basics

- The variables $1, $2 are user defined and are not specific to bash, unlike the way they work for awk

- Can make the script more user friendly and easier to read by using good variable names:

```
#!/usr/bin/env bash
filename=$1
number_of_lines=$2
echo filename is: $filename
echo
echo First $number_of_lines lines of file $filename are:
head -n $number_of_lines $filename
```

General scripting good practice

Should use inbuilt error flags within bash to terminate the program incase it encounters an error at any line (by default the script will continue to run till the end) using the set function:

```
#!/usr/bin/env bash
set -eu
```

- Should provide help / error messages that inform the user of the options if run incorrectly
- Should have comments – lines in a script that begin with # are ignored by the interpreter and are used for commenting about code
- Use of good / informative variable names

e.g. './options_example.2.shfilename is:

First lines of file are:
head: option requires an argument -- 'n
Try 'head --help' for more information.

General scripting good practice

```
#!/usr/bin/env bash
set -eu
# check that the correct number of options was given.
# If not, then write a message explaining how to use the
# script, and then exit.
if [ $# -ne 2 ]
then
echo "usage: options_example.3.sh filename number_of_lines"
echo
echo "Prints the filename, and the given first number of lines of the file"
exit
fi
# Use sensibly named variables
filename=$1
number_of_lines=$2
# check if the input file exists
if [ ! -f $filename ]
then
echo "File '$filename' not found! Cannot continue"
exit
fi
# If we are still here, then the input file was found
echo filename is: $filename
echo
echo First $number_of_lines lines of file $filename are:
head -n $number_of_lines $filename
```

General scripting good practice

- An inbuilt variable $# which should be equal to 2 options provided to the script, if not equal to 2 (-ne 2), then execute this block of code that starts from then then and terminates at fi (a variable that closes and if statement) and exits providing usage instructions.

```
if [ $# -ne 2 ]
then
echo "usage: options_example.3.sh filename number_of_lines"
echo
echo "Prints the filename, and the given first number of lines of the file"
exit
fi
```

- A check to see if the filename exists / provided correctly, if not then exit the program

```
if [ ! -f $filename ]
then
echo "File '$filename' not found! Cannot continue"
exit
fi
```

General scripting good practice

Without user options:
'./options_example.2.sh
filename is:

First lines of file are:
head: option requires an argument -- 'n
Try 'head --help' for more information

With user options:
./options_example.3.sh

usage: options_example.3.sh filename number_of_lines

Prints the filename, and the given first number of lines of the file

For loop

- Usually working with multiple files and would like to run the same operations on these files at one go, rather then individually
- The for command is very useful for this
- Basic usage: for files in directory/*; do operation $files; done
- for filename in loop_files/*; do wc $filename; done

```
2  8 28 loop_files/file.1
 5 20 70 loop_files/file.2
 6 24 84 loop_files/file.3
 1  4 14 loop_files/file.4
0 0 0 loop_files/file.5
```

Running bioinformatics programs

- A lot of bioinformatics software programs are run via the command line

- They require input files to do computations and then write the results out to an output file

- A good bioinformatics program / software package usually has:
  - A version number and software license
  - Options and parameters to choose from
  - Provides information on the options and usage of the program
  - Documentation on how to use the software the various options
  - Usually a good user base and support

Running bioinformatics programs

- Usually bioinformatics programs are installed on a unix system and are available system wide – might have to set the variable $PATH to access the program outside of the program directory

- Can access by typing in the tool's name e.g. samtools

Program: samtools (Tools for alignments in the SAM format)
Version: 1.10 (using htslib 1.10.2)
Usage: samtools <command> [options]

- bcftools

Program: bcftools (Tools for variant calling and manipulating VCFs and BCFs)
License: GNU GPLv3+, due to use of the GNU Scientific Library
Version: 1.10.2 (using htslib 1.10.2)
Usage: bcftools [--version|--version-only] [--help] <command> <argument>

Running bioinformatics programs

- The usual syntax for running a tool is:

Tool name – command from the tool – optional parameters for that specific command – input file – output file (or print to screen if no file is specified)

- The optional parameters for the specific command are usually provided in the format of unix style flags with a – (option) e.g. sort –u

- Spend some time going through the documentation for a bionformatics tool to determine what it does, required input and output formats, various options, parameters and its usage syntax as they do differ

Common errors

- Common errors when running bioinformatics tools include:
  - Not providing the correct path to the input file – try running the program in the directory which has the input files, or provide the complete path to the file(s)
  - Not specifying an output file or output directory
  - The input file might not have content, or be correctly formatted for the tool to use
- Most bioinformatics tools provide some error messages e.g. bcftools mpileup myfile

Error: mpileup requires the --fasta-ref option by default; use --no-reference to run without a fasta reference

- Do look at these error messages as they will help you determine what the issue is e.g. bcftools mpileup --no-reference myfile

[E::hts_open_format] Failed to open file "myfile" : No such file or directory
[mpileup] failed to open myfile: No such file or directory

Useful links

- http://people.duke.edu/~ccc14/duke-hts-2018/cliburn/Bash_in_Jupyter.html
- https://data-skills.github.io/unix-and-bash/03-bash-scripts/index.html
- https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008645
- https://scg.dgist.ac.kr/wp-content/uploads/2017/02/20170330.pdf
- https://wikis.utexas.edu/display/bioiteam/Shell+Script
- https://bioinformatics-core-shared-training.github.io/shell-genomics/06-writing-scripts/index.html
- http://williamslab.bscb.cornell.edu/?page_id=235

H3ABioNet

Thank you!!!

Acknowledgements: Amel Ghouila and course material adapted from H3ABioNet WT NGS Bioinformatics Course Africa 2021