LINUX II: Bash Scripting

Animal Evolution and Diversity 23

Joseph Kirangwa
Email: jkirangw@uni-koeln.de
Worm-Laboratory

LINUX II: What will you learn?

- Bash variables
- Bash scripting
- Bash conditional statements (e.g if statement)
- Loops (for, while)
- Bash functions
- grep, awk, sed, tr, cut
- Working with sequence data files
- Practical

LINUX II: Bash script

- Create script `script.sh`

```
#!/bin/bash
echo “hello world” > hello.txt
```

- Make it executable `chmod +x script.sh`
- Run it! `./script.sh`

LINUX II: Variables and command arguments

- create a variable and assign it a value with (do not use spaces around the equals sign!):

```
$ results_dir="results/"

$ echo $results_dir
```

LINUX II: Variables and command arguments

- Quoting and wrapping variable names in braces

```
$ sample="CNTRL01A"

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

LINUX II: Command-line arguments

- The variable $0 stores the name of the script:

```
#!/bin/bash

echo "script name: $0"

echo "first arg: $1"

echo "second arg: $2"

echo "third arg: $3"
```

LINUX II: Command-line arguments

- Running this file prints arguments assigned to `$0`, `$1`, `$2`, and `$3`:

```
$ bash args.sh arg1 arg2 arg3

script name: args.sh

first arg: arg1

second arg: arg2

third arg: arg3
```

LINUX II: Conditionals in a Bash Script: if Statements

```
#!/bin/bash

if [ "$#" -lt 3 ] # are there less than 3 arguments?

then

    echo "error: too few arguments, you provided $#, 3 required"

    echo "usage: script.sh arg1 arg2 arg3"

    exit 1

fi
```

LINUX II: Processing files with Bash using for loops and globbing

Three essential parts to creating a pipeline to process a set of files:

- Selecting which files to apply the commands to
- Looping over the data and applying the commands
- Keeping track of the names of any output files created

LINUX II: Processing files with Bash using for loops

```
#!/bin/bash

sample_info=samples.txt

# create a Bash array from $sample_info
sample_files=($(cat "$sample_info"))

for fastq_file in ${sample_files[@]}
do
    # strip .fastq from each FASTQ file, and add suffix
    # "-stats.txt" to create an output filename for each FASTQ file
    results_file="$(basename $fastq_file .fastq)-stats.txt"

    # run fastq_stat on a file, writing results to the filename we've
    # above
    cat $fastq_file | fastq-scan >stats/$results_file
done
```

LINUX II: Processing files with Bash using for loops and globbing

```
#!/bin/bash


for fastq_file in *.fastq
do
     #Count the number of entries in the fastq file
     echo "$fastq_file: " $(bioawk -c fastx 'END {print NR}' $fastq_file)

done
```

LINUX II: Processing files with Bash using while loop

```
#!/bin/bash

while read fastq_file
do
        echo "$fastq_file"

        # strip .fastq from each FASTQ file, and add suffix
        # "-stats.txt" to create an output filename for each FASTQ file

        results_file="$(basename $fastq_file .fastq)-stats.txt"

        # run fastq_scan on a file, writing results to the filename we've
        # above
        cat $fastq_file | fastq-scan >stats2/$results_file
done<samples.txt
```

LINUX II: Processing files with Bash using a function

```
count_entries() {

        read=$1

        #run bioawk on a file

        echo "$read: " $(bioawk -c fastx 'END {print NR}' $read)

}

#function call
count_entries A006200281_192656_S38_L000_R1_001.fastq
```

LINUX II: Powerful Linux Data Tools

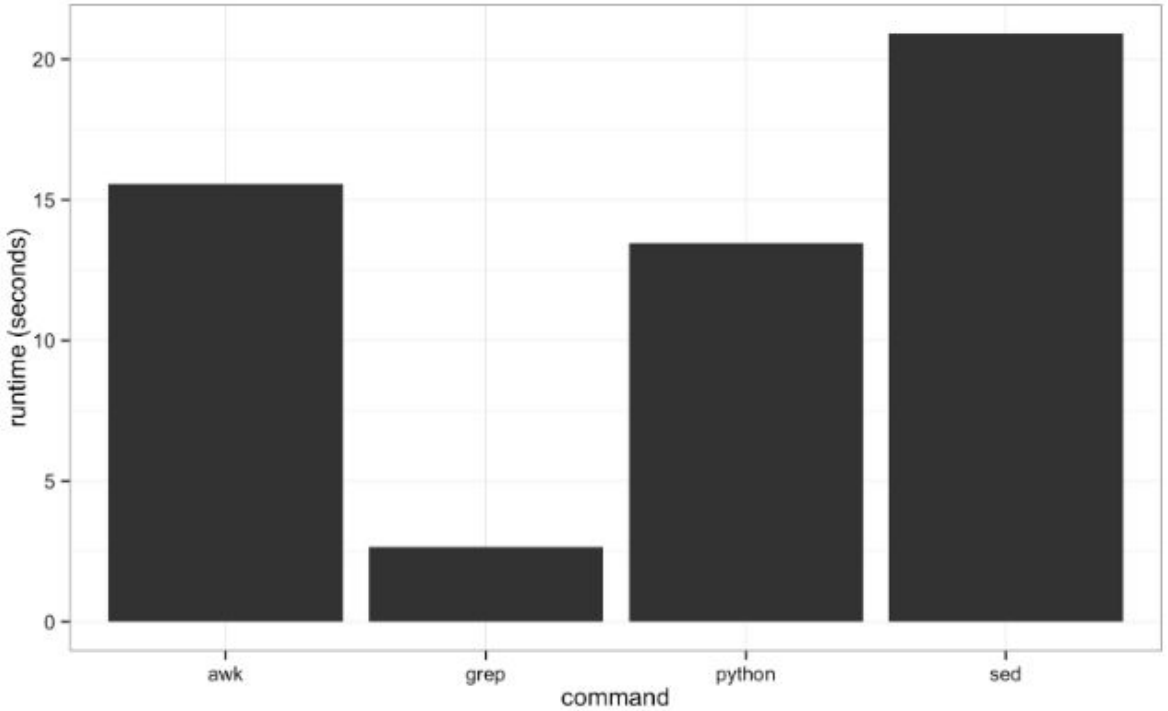

Figure 7-2. Benchmark of the time it takes to search the Maize genome for the exact string "AGATGCATG"

LINUX II: Powerful Linux Data Tools: grep

- **How many small nuclear RNAs are in our Mus_musculus.GRCm38.75_chr1.gtf file?**
- **snRNAs are annotated as gene_biotype "snRNA" in the last column of this GTF file**

```
$  grep -c 'gene_biotype "snRNA"' Mus_musculus.GRCm38.75_chr1.gtf

$  grep -o "Olfr.*" Mus_musculus.GRCm38.75_chr1_genes.txt | head -n 3

$  grep -v "^#" Mus_musculus.GRCm38.75_chr1.gtf | cut -f1,4,5 > test.txt
```

LINUX II: Powerful Linux Data Tools: awk

```
#Pattern matching
$ awk '/pseudogene/' Mus_musculus.GRCm38.75_chr1.gtf #pattern statement

#Print a specific column from the file
$ awk '{ print $2 }' Mus_musculus.GRCm38.75_chr1.gtf  | head -n 10 #action statement

#generate a three-column BED file from
$ awk '!/^#/ { print $1 "\t" $4-1 "\t" $5 }' Mus_musculus.GRCm38.75_chr1.gtf | head -n 3

#Filtering rows based on a condition
$ awk '$3 > 30' example.bed

#Print the entire record with variable $0
$ awk '{ print $0 }' example.bed

#Subtract within a pattern to calculate length of feature
$ awk '$3 - $2 > 18' example.bed

#all lines on chromosome 1 with a length greater than 10
$ awk '$1 ~ /chr1/ && $3 - $2 > 10' example.bed
```

LINUX II: Powerful Linux Data Tools: tr

```
#delete all occurrences of '>'
$ cat Hox_genes.txt | tr -d '>'


#translation of all occurrences of h at the start of a line to H.
$ cat Hox_genes.txt | tr '^h' 'H'

#translate all newlines to spaces.
$ cat Hox_genes.txt | tr '\n' ' '

#set all letters to uppercase by defining two ranges.
$ cat Hox_genes.txt | tr 'a-z' 'A-Z'
```

LINUX II: Powerful Linux Data Tools: sed

```
#delete all occurrences of '>'
$ cat Hox_genes.txt | sed 's/>//'
$ cat Hox_genes.txt | sed '/>/d'


#replace all occurrences of h at the start of a line to H.
$ cat Hox_genes.txt | sed 's/^h/H/'

#Add g for global replacements (all occurrences of the string per line)
$ cat Hox_genes.txt | sed 's/h/H/g'

#set all letters to uppercase.
$ cat Hox_genes.txt | sed 's/.*/\U&/'
```

LINUX II: Sequence data format: The FASTA format

```
>ENSMUSG00000020122|ENSMUST00000138518
CCCTCCTATCATGCTGTCAGTGTATCTCTAAATAGCACTCTCAACCCCCGTGAACTTGGT
TATTAAAAACATGCCCAAAGTCTGGGAGCCAGGGCTGCAGGGAAATACCACAGCCTCAGT
TCATCAAAACAGTTCATTGCCCAAAATGTTCTCAGCTGCAGCTTTCATGAGGTAACTCCA
GGGCCCACCTGTTCTCTGGT
```

LINUX II: Sequence data format: The FASTQ format

The FASTQ format looks like:

```
@DJB775P1:248:D0MDGACXX:7:1202:12362:49613 ❶
TGCTTACTCTGCGTTGATACCACTGCTTAGATCGGAAGAGCACACGTCTGAA ❷
+ ❸
JJJJJIIJJJJJJHIHHHGHFFFFFFCEEEEEDBD?DDDDDDBDDDABDDCA ❹
@DJB775P1:248:D0MDGACXX:7:1202:12782:49716
CTCTGCGTTGATACCACTGCTTACTCTGCGTTGATACCACTGCTTAGATCGG
+
IIIIIIIIIIIIIIIHHHHHHFFFFFFEECCCCBCECCCCCCCCCCCCCCCC
```

❶ The description line, beginning with `@`. This contains the record identifier and other information.

❷ Sequence data, which can be on one or many lines.

❸ The line beginning with `+`, following the sequence line(s) indicates the end of the sequence. In older FASTQ files, it was common to repeat the description line here, but this is redundant and leads to unnecessarily large FASTQ files.

❹ The quality data, which can also be on one or many lines, but *must* be the same length as the sequence. Each numeric base quality is encoded with

Practical: Count FASTA/FASTQ ENTRIES

1. Count the number of reads in A006200281_192656_S38_L000_R1_001.fastq

2. How many sequence entries in ***Panagrolaimus ps1159*** (panagrolaimus_ps1159.PRJEB32708.WBPS17) ?