SECTION 1: BASIC LINUX

The purpose of this tutorial is to introduce students to the frequently used tools for NGS analysis as well as giving experience in writing one-liners. Copy the required files to your current directory, change directory (cd) to the `linuxTutorial` folder, and do all the processing inside:

files needed for this exercise session can be found in the following address:

http://cimorgh.ir/workshop/linuxTutorial.zip 

You need to Open Terminal 

To see where you are in the file system:

```
$ pwd
/Users/…/Desktop/Workshop2018/linuxTutorial/
```

List the files in the current directory:

```
$ ls
data
```

I have deliberately chosen awk in the exercises as it is a language in itself and is used more often to manipulate NGS data as compared to the other command line tools such as `grep`, `sed`, `perl` etc.

awk (https://www.tutorialspoint.com/awk/index.htm) is a programming language which allows easy manipulation of structured data and is mostly used for pattern scanning and processing. It searches one or more files to see if they contain lines that match with the specified patterns and then perform associated actions. The basic syntax is:

```
awk [options] file
```

The working of awk is as follows:

- awk reads the input files one line at a time.
- For each line, it matches with given pattern in the given order.
- If no pattern matches, no action will be performed.
- In the above syntax, either search pattern or action are optional, But not both.
- If the search pattern is not given, then awk performs the given actions for each line of the input.
- If the action is not given, print all that lines that matches with the given patterns.
- Empty braces without any action does nothing. It won't perform default printing operation.
- Each statement in Actions should be delimited by semicolon.

Now try different commands from the sheet given below and in the next page:

```
Command1 | Command2 => Pipe the output of command1 as the input of command2
Command > filename => Save the output of command in file with given filename.
cat <file> => Reads the complete file, useful for piping into other commands. You ca
n also give several files as input and it concatenate them into the given order.
grep <word> <file> => Finds the lines which contain word in a given file, the –
v option returns the lines which DON'T contain the word.
cut –f <column number> -
d <delimiter> <file> => Cuts out the given column, you specify several columns or ra
nges of columns by doing –f 3,4 or –f 5-9
wc –l <file> => Counts the number of lines
sort <file> => Sorts the file alphabetically or numerically
uniq <file> => Only outputs unique lines. This needs to be applied to a sorted file!
 The –c option gives you the count for that unique entry.
export PATH=$PATH:/data/
```

UNIX / LINUX CHEAT SHEET

FILE SYSTEM

`ls` — list items in current directory

`ls -l` — list items in current directory and show in long format to see perimissions, size, and modification date

`ls -a` — list all items in current directory, including hidden files

`ls -F` — list all items in current directory and show directories with a slash and executables with a star

`ls dir` — list all items in directory dir

`cd dir` — change directory to dir

`cd ..` — go up one directory

`cd /` — go to the root directory

`cd ~` — go to to your home directory

`cd -` — go to the last directory you were just in

`pwd` — show present working directory

`mkdir dir` — make directory dir

`rm file` — remove file

`rm -r dir` — remove directory dir recursively

`cp file1 file2` — copy file1 to file2

`cp -r dir1 dir2` — copy directory dir1 to dir2 recursively

`mv file1 file2` — move (rename) file1 to file2

`ln -s file link` — create symbolic link to file

`touch file` — create or update file

`cat file` — output the contents of file

`less file` — view file with page navigation

`head file` — output the first 10 lines of file

`tail file` — output the last 10 lines of file

`tail -f file` — output the contents of file as it grows, starting with the last 10 lines

`vim file` — edit file

`alias name 'command'` — create an alias for a command

SYSTEM

`shutdown` — shut down machine

`reboot` — restart machine

`date` — show the current date and time

`whoami` — who you are logged in as

`finger user` — display information about user

`man command` — show the manual for command

`df` — show disk usage

`du` — show directory space usage

`free` — show memory and swap usage

`whereis app` — show possible locations of app

`which app` — show which app will be run by default

SEARCHING

`grep pattern files` — search for pattern in files

`grep -r pattern dir` — search recursively for pattern in dir

`grep -rn pattern dir` — search recursively for pattern in dir and show the line number found

`grep -r pattern dir --include='*.ext` — search recursively for pattern in dir and only search in files with .ext extension

`command | grep pattern` — search for pattern in the output of command

`find file` — find all instances of file in real system

`locate file` — find all instances of file using indexed database built from the updatedb command. Much faster than find

`sed -i 's/day/night/g' file` — find all occurrences of day in a file and replace them with night - s means substitude and g means global - sed also supports regular expressions

PROCESS MANAGEMENT

`ps` — display your currently active processes

`top` — display all running processes

`kill pid` — kill process id pid

`kill -9 pid` — force kill process id pid

NETWORKING

`wget file` — download a file

`curl file` — download a file

`scp user@host:file dir` — secure copy a file from remote server to the dir directory on your machine

`scp file user@host:dir` — secure copy a file from your machine to the dir directory on a remote server

`scp -r user@host:dir dir` — secure copy the directory dir from remote server to the directory dir on your machine

`ssh user@host` — connect to host as user

`ssh -p port user@host` — connect to host on port as user

`ssh-copy-id user@host` — add your key to host for user to enable a keyed or passwordless login

`ping host` — ping host and output results

`whois domain` — get information for domain

`dig domain` — get DNS information for domain

`dig -x host` — reverse lookup host

`lsof -i tcp:1337` — list all processes running on port 1337

SHORTCUTS

`ctrl+a` — move cursor to beginning of line

`ctrl+f` — move cursor to end of line

`alt+f` — move cursor forward 1 word

`alt+b` — move cursor backward 1 word

PERMISSIONS

`ls -l` — list items in current directory and show permissions

`chmod ugo file` — change permissions of file to ugo - u is the user's permissions, g is the group's permissions, and o is everyone else's permissions. The values of u, g, and o can be any number between 0 and 7.

`7` — full permissions

`6` — read and write only

`5` — read and execute only

`4` — read only

`3` — write and execute only

`2` — write only

`1` — execute only

`0` — no permissions

`chmod 600 file` — you can read and write - good for files

`chmod 700 file` — you can read, write, and execute - good for scripts

`chmod 644 file` — you can read and write, and everyone else can only read - good for web pages

`chmod 755 file` — you can read, write, and execute, and everyone else can read and execute - good for programs that you want to share

COMPRESSION

`tar cf file.tar files` — create a tar named file.tar containing files

`tar xf file.tar` — extract the files from file.tar

`tar czf file.tar.gz files` — create a tar with Gzip compression

`tar xzf file.tar.gz` — extract a tar using Gzip

`gzip file` — compresses file and renames it to file.gz

`gzip -d file.gz` — decompresses file.gz back to file

Reference: http://cheatsheetworld.com/programming/unix-linux-cheat-sheet/

EXERCISE 1: EXTRACTING READS FROM A FASTA FILE BASED ON SUPPLIED IDS

Tips:

Say you have `data.tsv` with the following contents:

```
$ cat data/test.tsv
blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2 ACTTTATATATT
blah_C3 ACTTATATATATATA
blah_C4 ACTTATATATATATA
blah_C5 ACTTTATATATT
```

By default `awk` prints every line from the file.

```
$ awk '{print;}' data/test.tsv
blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2 ACTTTATATATT
blah_C3 ACTTATATATATATA
blah_C4 ACTTATATATATATA
blah_C5 ACTTTATATATT
```

We print the line which matches the pattern `blah_C3`

```
$ awk '/blah_C3/' data/test.tsv
blah_C3 ACTTATATATATATA
```

`awk` has number of built-in variables. For each record *i.e* line, it splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 5 words, it will be stored in `$1, $2, $3, $4` and `$5`. `$0` represents the whole line. `NF` is a built-in variable which represents the total number of fields in a record.

```
$ awk '{print $1","$2;}' data/test.tsv
blah_C1,ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2,ACTTTATATATT
blah_C3,ACTTATATATATATA
blah_C4,ACTTATATATATATA
blah_C5,ACTTTATATATT

$ awk '{print $1","$NF;}' data/test.tsv
blah_C1,ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2,ACTTTATATATT
blah_C3,ACTTATATATATATA
blah_C4,ACTTATATATATATA
blah_C5,ACTTTATATATT
```

`awk` has two important patterns which are specified by the keyword called `BEGIN` and `END`. The syntax is as follows:

```
BEGIN { Actions before reading the file}
{Actions for everyline in the file}
END { Actions after reading the file }
```

For example,

```
$ awk 'BEGIN{print "Header,Sequence"}{print $1","$2;}END{print "-------
"}' data/test.tsv
Header,Sequence
blah_C1,ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2,ACTTTATATATT
blah_C3,ACTTATATATATATA
blah_C4,ACTTATATATATATA
blah_C5,ACTTTATATATT
-------
```

We can also use the concept of a conditional operator in `print` statement of the form `print CONDITION ? PRINT_IF_TRUE_TEXT : PRINT_IF_FALSE_TEXT`. For example, in the code below, we identify sequences with `lengths > 14`:

```
$ awk '{print (length($2)>14) ? $0">14" : $0"<=14";}' data/test.tsv
blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG>14
blah_C2 ACTTTATATATT<=14
blah_C3 ACTTATATATATATA>14
blah_C4 ACTTATATATATATA>14
blah_C5 ACTTTATATATT<=14
```

We can also use `1` after the last block `{}` to print everything (`1` is a shorthand notation for `{print $0}` which becomes `{print}` as without any argument print will print `$0` by default), and within this block, we can change `$0`, for example to assign the first field to `$0` for third line (`NR==3`), we can use:

```
$ awk 'NR==3{$0=$1}1' data/test.tsv
blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2 ACTTTATATATT
blah_C3
blah_C4 ACTTATATATATATA
blah_C5 ACTTTATATATT
```

You can have as many blocks as you want and they will be executed on each line in the order they appear, for example, if we want to print `$1` three times (here we are using `printf` instead of `print` as the former doesn't put end-of-line character),

```
$ awk '{printf $1"\t"}{printf $1"\t"}{print $1}' data/test.tsv
blah_C1 blah_C1 blah_C1
blah_C2 blah_C2 blah_C2
blah_C3 blah_C3 blah_C3
blah_C4 blah_C4 blah_C4
blah_C5 blah_C5 blah_C5
```

Given all that you have learned so far, we are going to extract reads from a FASTA file based on IDs supplied in a file. Say, we are given a FASTA file with following contents:

```
$ cat data/test.fa
>blah_C1
ACTGTCTGTC
ACTGTGTTGTG
ATGTTGTGTGTG
>blah_C2
ACTTTATATATT
>blah_C3
ACTTATATATATATA
>blah_C4
ACTTATATATATATA
>blah_C5
ACTTTATATATT
```

and an IDs file:

```
$ cat data/IDs.txt
blah_C4
blah_C5
```

Result:

After looking at the file, it is immediately clear that the sequences may span multiple lines (for example, for blah_C1). If we want to match an ID, we can first linearize the file by using the conditional operator as discussed above to have the delimited information of each sequence in one line, and then make logic to perform further functionality on each line later. Our logic is that for lines that contain header information /^>/ we can do something differently, and for other lines we use printf to remove new line character:

```
$ awk '{printf /^>/ ? $0 : $0}' data/test.fa
>blah_C1ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG>blah_C2ACTTTATATATT>blah_C3ACTTATATATATATA
>blah_C4ACTTATATATATATA>blah_C5ACTTTATATATT
```

We can then put each sequence on a separate line and also put a tab character ("\t") between the header and the sequence:

```
$ awk '{printf /^>/ ? "\n"$0 : $0}' data/test.fa
>blah_C1ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
>blah_C2ACTTTATATATT
>blah_C3ACTTATATATATATA
>blah_C4ACTTATATATATATA
>blah_C5ACTTTATATATT

$ awk '{printf /^>/ ? "\n"$0"\t" : $0}' data/test.fa
>blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
>blah_C2 ACTTTATATATT
>blah_C3 ACTTATATATATATA
>blah_C4 ACTTATATATATATA
>blah_C5 ACTTTATATATT
```

We can then use NR==1 block to stop printing a new line character before the first header (as you can see there is an empty space) and use next to ignore the later block:

```
$ awk 'NR==1{printf $0"\t";next}{printf /^>/ ? "\n"$0"\t" : $0}' data/test.fa
>blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
>blah_C2 ACTTTATATATT
>blah_C3 ACTTATATATATATA
>blah_C4 ACTTATATATATATA
>blah_C5 ACTTTATATATT
```

We can then pipe this stream to another awk statement using "\t" as a delimiter (which you can specify using -F) and use gsub to remove > from the start of each line since our IDs file doesn't contain that character:

```
$ awk 'NR==1{printf $0"\t";next}{printf /^>/ ? "\n"$0"\t" : $0}' data/test.fa |  awk
 -F"\t" '{gsub("^>","",$0);print $0}'
blah_C1 ACTGTCTGTCACTGTGTTGTGATGTTGTGTGTG
blah_C2 ACTTTATATATT
blah_C3 ACTTATATATATATA
blah_C4 ACTTATATATATATA
blah_C5 ACTTTATATATT
```

Now we load the IDs.txt file in the BEGIN block, store the IDs in the memory, and in the stream if the first field ($1) matches the ID stored in the memory, we output the formatted record:

```
$ awk 'NR==1{printf $0"\t";next}{printf /^>/ ? "\n"$0"\t" : $0}' data/test.fa | awk
-
F"\t" 'BEGIN{while((getline k < "data/IDs.txt")>0)i[k]=1}{gsub("^>","",$0); if(i[$1]
){print ">"$1"\n"$2}}'
>blah_C4
ACTTATATATATA
>blah_C5
ACTTTATATATT
```

EXERCISE 2: PREPARE FOR THE COURSE

Use test.tsv and miller.vcf

1. Check the file content by CAT command

2. Check the file content by AWK command

3. Print the lines contain “blah_C3” by AWK command

4. Convert file to Comma separated by AWK command and choose column 1 and 2

5. Adding header and footer (Header, Sequence) to the file

6. Convert test.fa to a tab separated file

7. Remove the extra “>” character from the results of the previous command

8. From the miller.vcf file, count the number of lines with “0/1” with cat command