Bash scripting practical

1. Write a script to read a tab delimited file containing primer names and sequences. The primer sequences contain eight nucleotide index (8nt-index) at position 1 to 8. Remove the 8nt-index from each primer sequence and print out both primer names and edited sequences in FASTA format.
   a. Input: dv1_primer.txt

Script file: practical1.sh

Run the script file:

```
$./practical1.sh dv1_primer.txt
```

CODE DESCRIPTION:

```
#!/bin/bash

while read line || [ -n "$line" ]; #Use while loop to read the input file.  This while loop will keep going until the end of the file OR until the "$line" variable is not empty. This will make sure that the file's last line is read by the while loop.

do
    line_array=( $line )  # Keep data from each line in an array

    name=${line_array[0]}  #The first index of array is primer name. Assign primer name to variable “name”.

    seq=${line_array[1]}  #The second index of array is primer sequence. Assign primer sequence to variable “seq”
    echo -e ">$name \n${seq:8}" # Print the name and sequence on the screen in fasta format. Use the substring function to get the primer sequence from position 9 to the end. The "-e" option will tell the echo command to recognize "\n".

done < $1   # Get a primer file from command line arguments.
```

2. Write a script to read a genome sequence from a FASTA file. Split the genome sequence into each gene using the table of gene positions below (c). Pipe all gene sequences in a FASTA format to an output file.
    a. Input: reference.fasta
    b. Output: dv1_gene.fasta
    c. Gene position

Gene	Start	End
capsid	95	436
prM	437	934
envelope	935	2419
ns1	2420	3475
ns2a	3476	4129
ns2b	4130	4519
ns3	4520	6376
ns4a	6377	6826
ns4b	6827	7573
ns5	7574	10270

!!First, copy the position of the gene to a text file "gene.txt".

Script file: practical2.sh

How to run the script file:

```
$./practical2.sh reference.fasta gene.txt > dv1_gene.fasta
```

CODE DESCRIPTION:

```
#!/bin/bash

ref_file=$1   # The first command line argument is the file
"reference.fa". Assign the reference file to the variable
"ref_file".
```

```
gene_file=$2  # The second argument on the command line is the position of the gene in a text file.  Assign the text file of gene positions to the variable "gene file."
```

```
ref=$(grep -v ">" $ref_file | tr -d "\n")  # Read the genome sequence from a file and store it in the "ref" variable. "grep -v" is used to find lines that don't have ">" in them. Then, pass sequence line to "tr -d" to get rid of the newline.
```

```
while read line || [ -n "$line" ]; #Use while loop to read the input file.  This while loop will keep going until the end of the file OR until the "$line" variable is not empty. This will make sure that the file's last line is read by the while loop.
```

```
Do
    line=$(echo $line | tr -d "\r") # Use tr -d to get rid of the “\r”.

    line_array=($line) # Keep data from each line in an array

    gene=${line_array[0]} #The first index of array is gene name. Assign gene name to variable “gene”.

    start=${line_array[1]} #The second index of array is start position. Assign start position to variable “start”.

    end=${line_array[2]} #The third index of array is end position. Assign end position to variable “end”.
    position=$((start-1)) # The substring function will take the substring after the given position. So, the position of the substring should be one position after the start position.

    length=$((end-start+1)) # Calculate length of gene

    ###let length=(end-start)+1 #You can also use let command to calculate length.

    gene_seq=${ref:position:length} # Using calculated position and length with the substring function to get the gene sequence.

    echo -e ">$gene \n$gene_seq" # Print the gene name and gene sequence on the screen in FASTA format.

done < $gene_file Get a gene position file from command line arguments
```

Use ">" to pass data to a text file so that the result of printing can be kept in a file.

```
./practical2.sh reference.fasta gene.txt > dv1_gene.fasta
```

Group practical

1. Create a folder ‘p1’ , and then move files ‘p1_1.fastq.gz’ and ‘p1_2.fastq.gz’ into the newly created folder.

Script file: movefile.sh

How to run the script file:

```
$./movefile.sh
!!This script needs to be run in the same folder as the
“fastq.gz” files.
```

CODE DESCRIPTION:

```
#!/bin/bash

for file in *.gz # Looping over the names of files that end
in ".gz."

do
     fd=${file%%_*} # Use substring to get rid of anything
after the underscore from the file name.

# Before making a new directory and moving files, check
     if [ -d $fd ]; then # Check if the directory exists.

          # If yes
          if [ -e $file ]; then # Check to see if the file
     you want to move is ready to be moved.

          echo "Moving file $file to folder $PWD/$fd/"
```

```
        mv $file "$PWD/$fd/" # Move the file to the newly created folder

        fi

    # If No
    else
    mkdir "$PWD/$fd" # Make a new folder.

    echo "Moving file $file to folder $PWD/$fd/"
    mv $file "$PWD/$fd/" # Move the file to a newly created folder
    fi
done
```

2. Write a script “`run_analysis.sh`“ to build an automated pipeline to run the following processes:
    1) Run “Trimmometic” program to trim low quality bases
        a. Input: p1_1.fastq.gz, p1_2. fastq.gz in the p1 folder
    2) Align trimmed sequences to a reference genome using minimap2
    3) Convert a SAM file (from step2) to a BAM file, then sort BAM file and filter only paired mapped
    4) Run samtools flagstat

Script file: run_analysis.sh

How to run the script file:

```
$./run_analysis.sh p1 reference.fasta
!! The folder p1 contains the files p1_1.fastq.gz and p1_2.fastq.gz.
```

CODE DESCRIPTION:

```
#!/bin/bash

fd=$1 # The script needs two inputs: the name of the folder in argument 1 and the name of the sequence file in argument 2.

ref_file=$2


files=($(ls $fd/*.gz)) # Use the "ls" command to list all files that end in ".gz" in the input folder and keep the file names in the "files" array.
```

```
file1=${files[0]} # Assign first file in variable "file1"

file2=${files[1]} # Assign second file in variable "file2"


## 1. Run Trimmometic ##

echo "1. Run Trimmometic: $fd"

# Prepare the name of the output file before you run Trimmometic. For each input file, Trimmometic will return two output files.
```

Therefore, four output files will be created. A trim file has reads that pass the quality control for both pairs. If only one read of a pair passes the QC, it will be saved in an unpair file.

String manipulation used here if Find the ".fastq.gz" part and replace it with the name you want.

```
f1_trim=${file1/.fastq.gz/.trim.fastq.gz}

f2_trim=${file2/.fastq.gz/.trim.fastq.gz}

f1_unpair=${file1/.fastq.gz/.unpair.fastq.gz}

f2_unpair=${file2/.fastq.gz/.unpair.fastq.gz}


# Place the file name variable in the Trimmometic command.

trim_cmd="trimmomatic PE -phred33 $file1 $file2 $f1_trim $f1_unpair $f2_trim $f2_unpair LEADING:20 TRAILING:20 SLIDINGWINDOW:5:20 MINLEN:40"


# Show the input file and the output file of this step on the screen.
```

```
echo "Input files: $file1 $file2"
echo -e "Output files: \n$f1_trim \n$f2_trim \n$f1_unpair \n$f2_unpair"
##----Run trimmometic command
$trim_cmd  # Run the trimmometic command

echo -e "\n\n"
## 2. Run Minimap2
echo "2. Run Minimap2: $fd" # Second step, run alignment with Minimap2
out_sam=${file1/_*.fastq.gz/.sam} # Prepare the name of the output file
map_cmd="minimap2 -ax sr -o $out_sam $ref_file $f1_trim $f2_trim" # Put variables to minimap2 command.

# Show the input file and the output file of this step on the screen.
echo -e "Input files: \n$f1_trim \n$f2_trim"
echo -e "Output files: \n$out_sam"
##----Run minimap2 command
$map_cmd  # Run the Minimap2 command

echo -e "\n\n"
## 3. Run samtools
```

```
echo "3. Run samtools: $fd" # Third step, run samtools

echo "-----Convert SAM to BAM---------" # Convert the output file from minimap2 from SAM to BAM.

out_bam=${file1/_*.fastq.gz/.bam} # Prepare the name of the output file


# Show the input file and the output file of this step on the screen.

echo "Input files: $out_sam"

echo "Output files: $out_bam"

bam_cmd="samtools view -Shb -o $out_bam $out_sam" # Put variables to samtools view command.


##---Run command: Convert SAM to BAM

$bam_cmd  # Run the Samtools view command


echo -e "\n"

echo "-----Sort BAM file-------" # Then, sort the BAM file

sorted_bam=${file1/_*.fastq.gz/.sorted.bam} # Prepare the name of the output file


# Show the input file and the output file of this step on the screen.

echo "Input files: $out_bam"

echo "Output files: $sorted_bam"
```

```
sort_cmd="samtools sort -@ 2 -o $sorted_bam $out_bam"
# Put variables to samtools sort command.
##---Run command: Sort BAM file
$sort_cmd # Run the Samtools sort command


echo -e "\n"
echo "-----Filter paired mapped-------" # Filter only read that mapped pairs from the sorted BAM file.


pair_bam=${file1/_*.fastq.gz/.sorted.pair.bam} # Prepare the name of the output file


# Show the input file and the output file of this step on the screen.
echo "Input files: $sorted_bam"
echo "Output files: $pair_bam"
pair_cmd="samtools view -hb -f 2 -o $pair_bam $sorted_bam" # Put variables to samtools view command.


##-----Run filter paired mapped
$pair_cmd # Run the Samtools view command


echo -e "\n\n"
## 4. Run Flagstat
```

```
echo "4. Run Flagstat: $fd" # Run FLAGSTAT to see a summary of the results of the alignment.

# Show the input file of this step on the screen.
echo "Input files: $pair_bam"
flag_cmd="samtools flagstat $pair_bam" # Put variables to flagstat command.
##-----Run FLAGSTAT
$flag_cmd # Run the flagstat command
```

!!Here is the easiest way to create the automated pipeline. To make the pipeline better, each step should have a checkpoint for running errors. I hope this practical gives you some ideas to use bash scripting for your work.