BASh based pipeline for variant calling and annotation

This task was carried out by Joseph Osifeso and is dedicated to my friend Olumide (Lincoln).

Introduction

The objective of this project is to create a pipeline that makes it simple and easy to run routine variant callings anywhere anytime as long as you have a starting fastq file. Variant calling is usually done to see how the population changed over time relative to the original population. Therefore, in this project I set out to automate the process of calling the variants when the sequenced reads and reference genome are supplied. To achieve this, I have used for loops in the script to iterate the commands over multiple input files. In these for loops, the filename has been defined as a variable in the for statement, which will enable anyone to run the loop on multiple files. This script can be used to perform a large number of variants calling on one or many files. This saves you the effort of having to type each of those commands over for each of your data files and makes your work less error-prone and more reproducible.

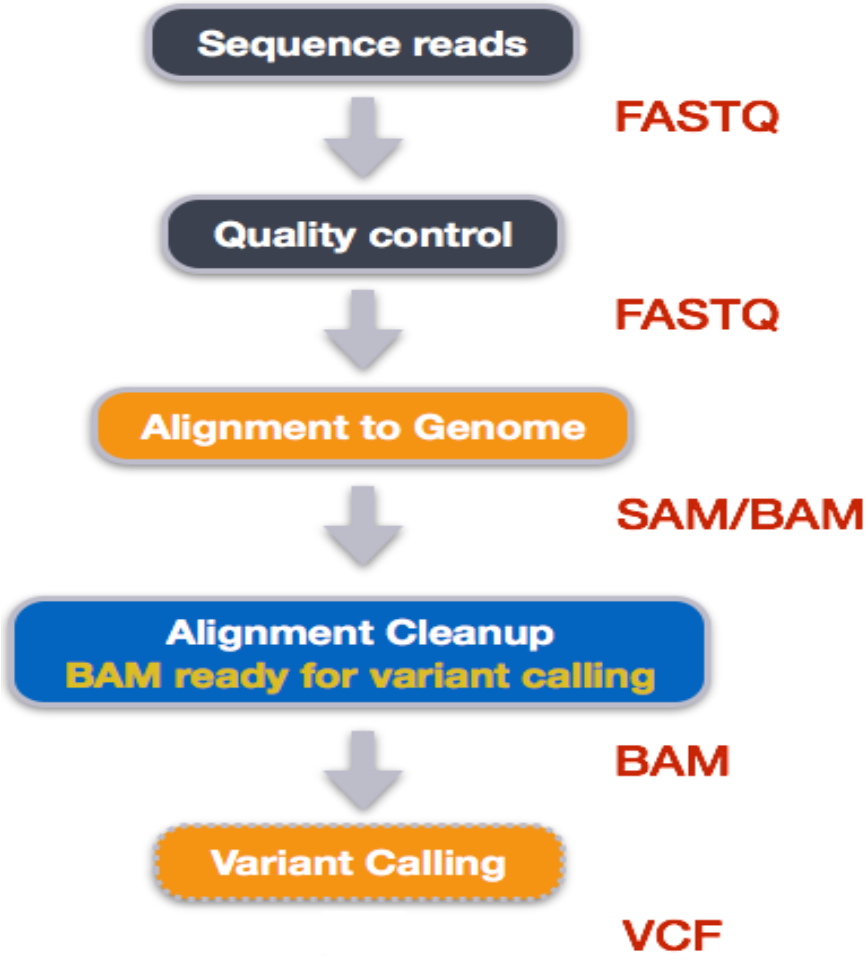

Figure 1. *Diagram of the workflow in the project.*

Methodology

Throughout the workflow directories to store different results were created.

Software packages used

1. FastQC
2. Fastp
3. Burrows Wheeler Aligner (BWA)
4. Samtools
5. Bcftools

Datasets used to test the pipeline

wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR124/000/SRR12404800/SRR12404800_1.fastq.gz -O SRR12404800_r1.fastq.gz

wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR124/000/SRR12404800/SRR12404800_2.fastq.gz -O SRR12404800_r2.fastq.gz

wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR124/000/SRR12404800/SRR12404801_1.fastq.gz -O SRR12404801_r1.fastq.gz

wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR124/000/SRR12404800/SRR12404801_2.fastq.gz -O SRR12404801_r2.fastq.gz

Reference

wget https://hgdownload.soe.ucsc.edu/goldenPath/mm10/chromosomes/chr1.fa.gz

pls note: To use this script you will have to save your dataset as ***.r1.fastq.gz and .r2.fastq.gz*** file. Also, you save your reference genome as ***.fa.gz*** file.

First of all, I enquired using an if statement whether the necessary packages are installed and a 'yes or no' response determines if the script sudo installs the packages or skips the process entirely. Then it requests the path to the directory of your datasets and reference genome and copies them to the directories where they would be called and proceeds to running the packages. It also creates directories to store the results that will be generated as part of the workflow.

Analysis

First, it uses a for loop performed fastqc to clean up the dataset. Thereafter, it unzips the dataset also using a for loop and then runs Fastp to trim off adapters and bad reads.

The fastqc command:

```
echo "Running FastQC ..."


for file in *.fastq.gz
        do
        fastqc $file
```

```
done
```

The unzipping command:

```
echo "Unzipping..."

for filename in *.gz
    do
    gunzip $filename
    done
echo "Done"
```

The fastp command:

```
for f1 in
*1.fastq

do
    f2=${f1%%1.fastq}"2.fastq"
  fastp \
    -i "$PWD/${f1}" \
    -I "$PWD/${f2}" \
    -o "qc_reads/$(basename ${f1%_*})_r1.trimmed.fastq" \
    -O "qc_reads/$(basename ${f2%_*})_r2.trimmed.fastq" \
    --html "qc_reads/${f1%_*}_fastp.html"
done
echo "Done"
```

Variant calling

Prior to performing the alignment, the program unzips the reference genome and then it performs read alignment (mapping) to determine where in the genome the reads originated from. I used the Burrows Wheeler Aligner (BWA), which is a software package for mapping low-divergent sequences against a large reference genome.

The process of alignment involves two steps:

1. Indexing the reference genome
2. Aligning the reads to the reference genome using bwa-mem command.

The bwa index command:

```
echo "Indexing the reference
genome..."

bwa index $ref_genome
echo "Done"
```

The bwa-mem algorithm starts by aligning the reads from one of the samples in our dataset until it iterates the whole process on all of the supplied files. The resulting sam file is stored in the sam directory in result folder then from there it is compressed into a binary version (bam file) which is a smaller file and allows for indexing.

The variant calling command:

```
for file1 in
~/workshop/data/trimmed_fastq
_file/*_r1.trimmed.fastq
                                    do
                                    echo "working with file $file1"

                                    base=$(basename $file1 _r1.trimmed.fastq)
                                    echo "base name is $base"

                                file1=~/workshop/data/trimmed_fastq_file/${base}_r1.trimmed.fastq

                                file2=~/workshop/data/trimmed_fastq_file/${base}_r2.trimmed.fastq
                                    sam=~/workshop/results/sam/${base}.aligned.sam
                                    bam=~/workshop/results/bam/${base}.aligned.bam
                                    sorted_bam=~/workshop/results/bam/${base}.aligned.sorted.bam
                                    raw_bcf=~/workshop/results/bcf/${base}_raw.bcf
                                    variants=~/workshop/results/vcf/${base}_variants.vcf

                                final_variants=~/workshop/results/vcf/${base}_final_variants.vcf

                                    echo "Aligning..."
                                    bwa mem $ref_genome $file1 $file2 > $sam
                                    echo "Done"
                                    echo "Converting to bam ..."
                                    samtools view -S -b $sam > $bam
                                    echo "Done"
                                    echo "Sorting bam ..."
                                    samtools sort -o $sorted_bam $bam
                                    echo "Done"
                                    echo "Indexing sorted files ..."
                                    samtools index $sorted_bam
                                    bcftools mpileup -O b -o $raw_bcf -f $ref_genome $sorted_bam
                                    bcftools call --ploidy 1 -m -v -o $variants $raw_bcf
                                    vcfutils.pl varFilter $variants > $final_variants

                                    done
                                    echo "Done"
```

The bam file is then sorted while the variant calling occurs on the sorted files. I used the bcftools *mpileup* command to count the read coverage while using the bcftools *call* command to identify the variants.

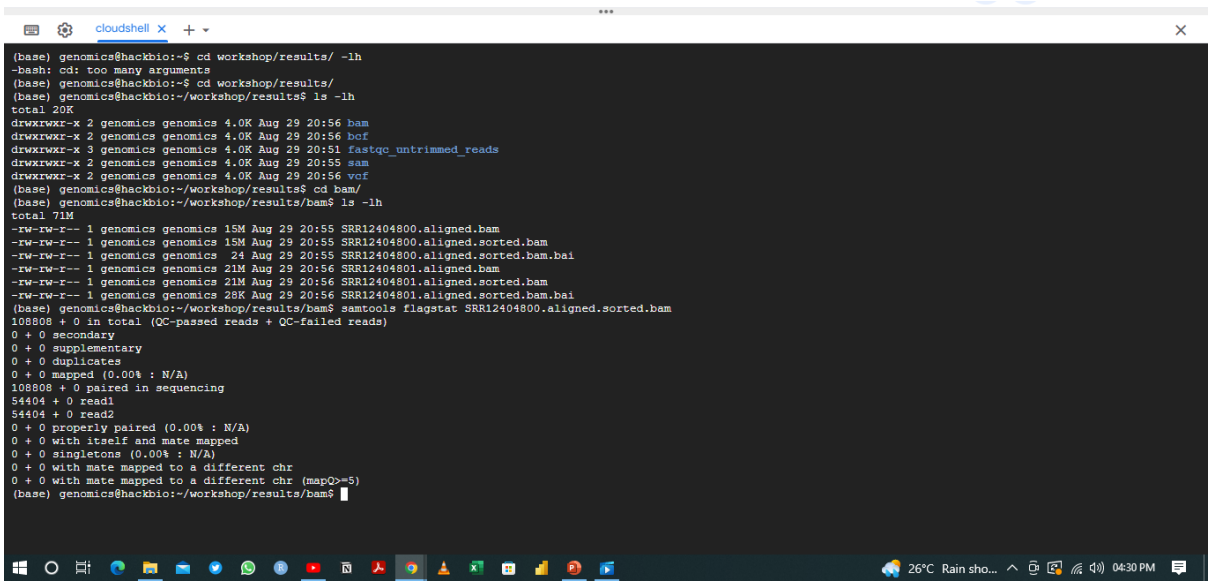

Figure 2: Result Directories *The diagram shows the different directories in the result folder where the results of the variant call workflow are stored.*

Conclusion

In conclusion, the project really enabled me to understand how to effectively use a for loop to automate the entire variant calling procedure. Also, I learned how to use an if statement to install packages, to manage directory structures to store the results of the different analyses in the pipeline. I was able to understand the beauty of research and how to ask questions not just questions but the right questions.

Code for the above pipeline

https://github.com/Jossyfeson/JEDI/blob/main/HackBio%20stage%20three%20(Project)

Video Presentation

https://drive.google.com/file/d/1WVZ7Jpsj7JNhnyQxulA9b-_NbQd_hdQb/view?usp=sharing