MA5112 Variant Calling
Assessment III

Barry Digby

Abstract

A clinical colleague has asked you to perform germline variant calling on normal biopsies (only) from three head and neck squamous cell carcinoma (HNSCC) cancer patient samples. She has provided you with fastq files generated from 100bp PE libraries on an Illumina HiSeq machine. Your report should also detail any variants shared amongst the patients (if detected).

This report provides a comprehensive guideline using both samtools and Picard/GATK to generate VCF files containing patient sample variants.

1. Data Assembly

1.1. Raw Reads

The 100bp paired end (PE) reads generated by the Illumina HiSeq Machine were supplied under /data4/nextgen2015/pilib/MA5112/_Assign3 on the NUIG syd cluster. In the directory containing the 6 fasta files, the following command was used to transfer them to my own directory:

```
1 cp *fastq /data4/nextgen2015/users/11384191/2018/assessment3/data
```

It is useful to run *du -sh ** to check the files size before and after copying them to your own directory, to ensure it was successful.

1.2. Genome Data

We were asked to align the reads to the hg19 human genome. This required downloading the genome to our directory for further downstream alignment using BWA–MEM.

Source notes on the chromFa.tar.gz file we downloaded: *The assembly sequence in one file per chromosome. Repeats from RepeatMasker and Tandem Repeats Finder (with period of 12 or less) are shown in lower case; non-repeating sequence is shown in upper case.*

Using the following commands, we can:

1. Download the Genome from UCSC Genome Browser
2. Unpack each chromosome individually
3. Concatenate each chromosome into one file (hg19.fa)
4. Remove the individual chromosome files used to generate the hg19.fa file

```
wget
 ↪ http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz
tar zxvf chromFa.tar.gz
cat *.fa > hg19.fa
rm chr*.fa
```

Now we have the hg19 human genome in one fasta file to work with for the project. A quick check on the file size confirmed the concatenation was successful:

```
ls -l --b=M hg19.fa | cut -d " " -f5
```

The output from this command was 3052Mb, or 3Gb. (note to view the output in Gb instead of Mb, '--b=Gb ' should be used).

2. Quality control tool for high throughput sequence data

2.1. FastQC

Before beginning any analysis on the reads, we must first run QC. FastQC generates a set of analysis which tells us if there is anything wrong with the reads before continuing with the analysis. Use the following bash script to generate QC files on the data:

```
#!/bin/bash

#Your job name
$ -N Mr.Meeseeks

#The job should be placed into the queue 'all.q'
$ -q all.q

#Running in the current directory
$ -cwd

#Export some necessary environment variables
$ -v PATH
$ -v LD_LIBRARY_PATH
$ -v PYTHONPATH
$ -S /bin/bash

#Finally, put your commanad here:


for f in *.fastq;
do
          fastqc $f;

done
```

Place this bash script in the same folder as the raw read files, and submit to the cluster:

```
qsub fastqc.sh
```

Unzip each .zip file created for each read. We now have 8 fastqc reports. After evaluating each read individually (only one read was of concern) we can concatenate the reports using MultiQC. Source notes on MultiQC: *"Aggregate results from bioinformatics analyses across many samples into a single report".*

Run the following commands from the directory containing the fastqc files generated from FastQC:

```
module load git/2.9.5
module load zlib/1.2.8

git clone https://github.com/ewels/MultiQC.git

multiqc .
```

The resulting multiqc_report.html file was downloaded to my local machine using the sftp command.

2.2. Raw Reads Analysis

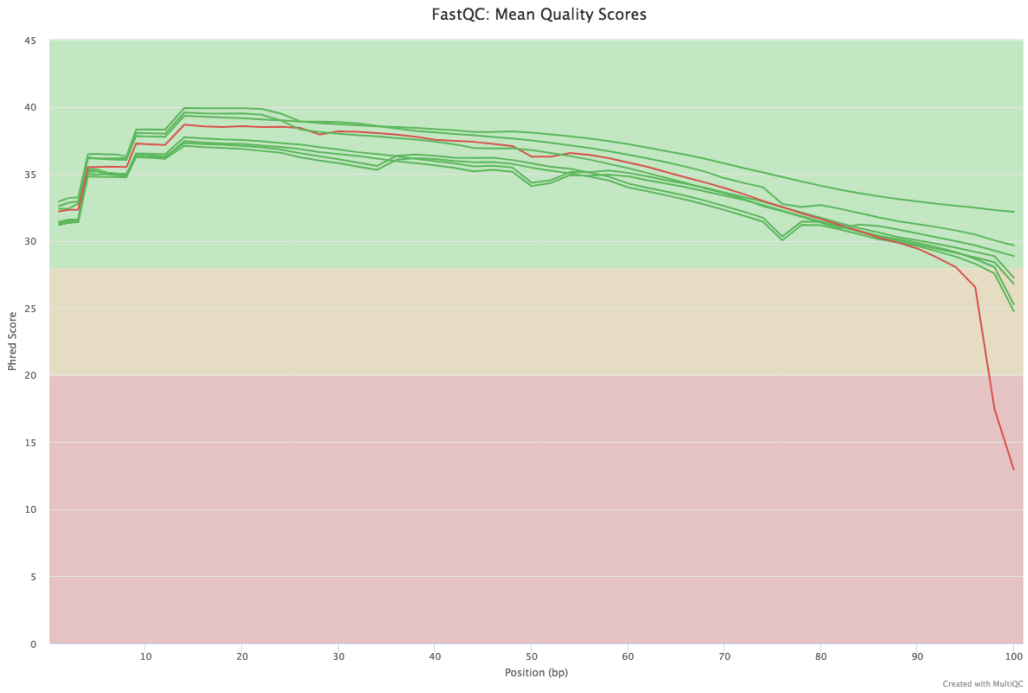


Figure 1: Raw Reads MultiQC Report

We can see from **Figure 1** that only one read, *HN72_AC_P2.fastq*, drops into a red Phred score (<20) at the 94th base of the read. Several reads enter the orange Phred score (<30) at the 98th base of the reads.

- Phred scores of 20 indicate 99% basecall accuracy.
- Phred scores of 30 indicate 99.9% basecall accuracy.
- Phred scores of 40, where most of our reads lie, indicate 99.99% basecall accuracy.

Despite one read "failing" the mean quality score, the reads are of excellent quality. If I wanted to ensure 99.99% accuracy across all reads I would trim each read by setting a quality score threshold of <30.

2.3. FastQC Summary Notes

Kmer Content Fail:

- HN60_s2_normal.BD1LYPACXX.lane_7_P1_I16.hg19.sequence.fastq
- HN72_s2_normal.AC254KACXX.lane_2_P1_I15.hg19.sequence.fastq
- HN72_s2_normal.AC254KACXX.lane_2_P2_I15.hg19.sequence.fastq
- HN72_s2_normal.AH0LENADXX.lane_2_P1_I15.hg19.sequence.fastq
- HN72_s2_normal.AH0LENADXX.lane_2_P2_I15.hg19.sequence.fastq

*This module will issue a warning if any k-mer is imbalanced with a binomial p-value $<10^{-5}$.

Libraries which derive from random priming will nearly always show Kmer bias at the start of the library due to an incomplete sampling of the possible random primers.

Per Tile Sequence Quality:

- HN72_s2_normal.AC254KACXX.lane_2_P1_I15.hg19.sequence.fastq
- HN72_s2_normal.AC254KACXX.lane_2_P2_I15.hg19.sequence.fastq

"This module will issue a warning if any tile shows a mean Phred score more than 5 less than the mean for that base across all tiles."

Whilst warnings in this module can be triggered by individual specific events we have also observed that greater variation in the phred scores attributed to tiles can also appear when a flowcell is generally overloaded. In this case events appear all over the flowcell rather than being confined to a specific area or range of cycles. We would generally ignore errors which mildly affected a small number of tiles for only 1 or 2 cycles, but would pursue larger effects which showed high deviation in scores, or which persisted for several cycles.

No reads contained any Adapter Contamination.

I was unsure if trimming the reads was necessary. Multiple forum boards

seemed to echo the fact that you don't need to do quality trimming with BWA–MEM. This is because BWA–MEM largely does local alignment. If a tail cannot be mapped well, it will be soft clipped.

3. Trimming Reads

3.1. Sickle

To trim the reads, I used Sickle. Source notes from Sickle: Sickle is a tool that uses sliding windows along with quality and length thresholds to determine when quality is sufficiently low to trim the 3'-end of reads and also determines when the quality is sufficiently high enough to trim the 5'-end of reads. It will also discard reads based upon the length threshold.

The drop off in quality at the 3' end is what we observed in some of our samples, thus using the –q option for Sickle **we can set the quality score threshold to 30 instead of the default setting 20**. Here are some more of the arguments used for running Sickle with paired end reads:

- qual–type: required argument, 't'
- pe-file1, required argument, 'f'
- pe-file2, required argument, 'r'
- output-pe1, required argument, 'o'
- output-pe2, required argument, 'p'
- output-single, required argument, 's'
- qual-threshold, required argument, 'q'

Before running sickle, I changed the names of the input files to HN51_P1.fastq HN51_P2.fastq etc; to help GNU parallel identify each read 1, read 2. Run the following command in your directory containing the raw reads:

```
|------ SampleDir
|  |--genome
|  |  |--hg19.fa
|  |
|  |
|  |--reads
|     |--HN51_P1.fastq
|     |--HN51_P2.fastq
|     |--HN60_P1.fastq
|     |--HN60_P2.fastq
|     |--HN72_AC_P1.fastq
|     |--HN72_AC_P2.fastq
|     |--HN72_AH_P1.fastq
|     |--HN72_AH_P2.fastq
```

```
find ./ -name "*.fastq" | grep -v _P2.fastq | sed
↪  's/_P1.fastq//' | parallel /tools/sickle-1.33/sickle pe -f
↪  {}_P1.fastq -r {}_P2.fastq -t sanger -o '{}'_R1.fastq -p
↪  '{}'_R2.fastq -s '{}'_s.fastq -q 30
```

```
|------ SampleDir
|  |--genome
|  |  |--hg19.fa
|  |
|  |
|  |--reads
|     |--HN51_P1.fastq
|     |--HN51_P2.fastq
|     |--HN51_R1.fastq
|     |--HN51_R2.fastq
|     |--HN51_s.fastq
|     |--HN60_P1.fastq
|     |--HN60_P2.fastq
|     |--HN60_R1.fastq
|     |--HN60_R2.fastq
```

```
|     |--HN60_s.fastq
|     |--HN72_AC_P1.fastq
|     |--HN72_AC_P2.fastq
|     |--HN72_AC_R1.fastq
|     |--HN72_AC_R2.fastq
|     |--HN72_AC_s.fastq
|     |--HN72_AH_P1.fastq
|     |--HN72_AH_P2.fastq
|     |--HN72_AH_R1.fastq
|     |--HN72_AH_R2.fastq
|     |--HN72_AH_s.fastq
```

After running the command delete the input reads and the _s reads.

3.2. Sickle Output

```
FastQ paired records kept: 9115588 (4557794 pairs)
FastQ single records kept: 174684 (from PE1: 117874, from PE2: 56810)
FastQ paired records discarded: 119532 (59766 pairs)
FastQ single records discarded: 174684 (from PE1: 56810, from PE2: 117874)


FastQ paired records kept: 163491602 (81745801 pairs)
FastQ single records kept: 6100578 (from PE1: 4242124, from PE2: 1858454)
FastQ paired records discarded: 4291542 (2145771 pairs)
FastQ single records discarded: 6100578 (from PE1: 1858454, from PE2: 4242124)


FastQ paired records kept: 213453162 (106726581 pairs)
FastQ single records kept: 19577956 (from PE1: 8767228, from PE2: 10810728)
FastQ paired records discarded: 9638888 (4819444 pairs)
FastQ single records discarded: 19577956 (from PE1: 10810728, from PE2: 8767228)


FastQ paired records kept: 217277766 (108638883 pairs)
FastQ single records kept: 17715687 (from PE1: 7973061, from PE2: 9742626)
FastQ paired records discarded: 8251872 (4125936 pairs)
FastQ single records discarded: 17715687 (from PE1: 9742626, from PE2: 7973061)
```

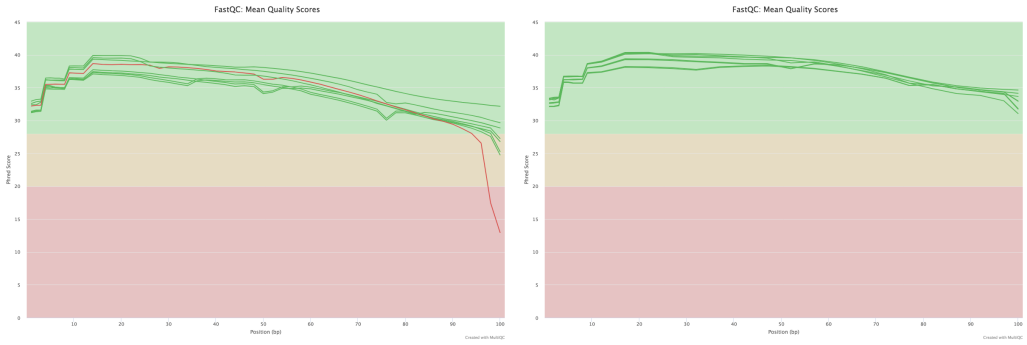
Figure 2: Sickle information

After running sickle the input reads, annotated P1, P2, can be removed. The output of sickle used the annotation R1, R2 are used for downstream analysis.
Inspecting the files from the HN72_AH_R1 / HN72AH_R2 / HN72_AH_s (merged sample) using:

```
ls -l --b=M HN72_AH0LE_merged.fastq | cut -d " " -f5
```

We see that the merged file is ∼30MB whilst the forward and reverse reads are ∼100MB. This would suggest too many reads were removed from the single output file (merged), and it did not concatenate the Forward and Reverse reads as I had hoped. I will not use the merged single files going forward.

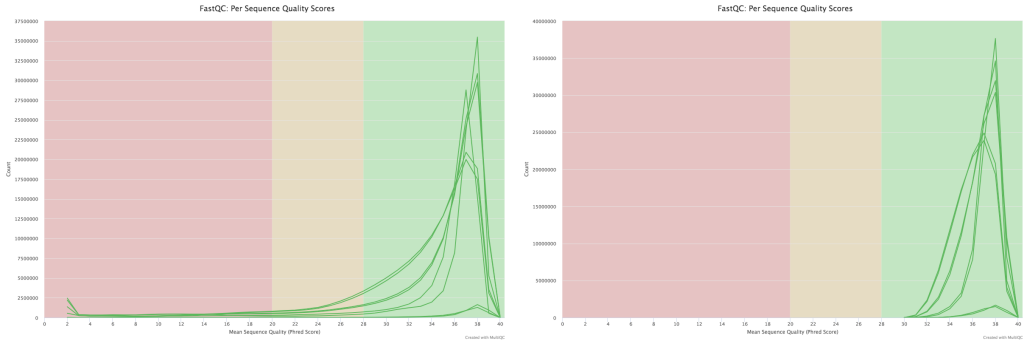
3.3. Comparing Raw Reads vs. Trimmed Reads using QC



(a) Before (b) After

Figure 3: FastQC results: Raw reads compared against Reads after Trimming using Sickle

We can see from **Figure 3** that trimming was successful, all of our reads are now 99.99% Phred score.



(a) Before (b) After

Figure 4: FastQC results: Raw reads compared against Reads after Trimming using Sickle

Figure 4 confirms we have isolated only the highest quality reads for further downstream analysis.

4. Samtools Variant Calling

Now that we have trimmed reads to work with for analysis, the first variant calling step of this report details how to use samtools to generate VCF files containing variant information on the patient samples. The process involves aligning our reads to the hg19 genome, followed by downstream samtools analysis of the bam files generated via BWA–MEM.

4.1. Indexing Genome

Before aligning our reads to the genome, it must first be indexed using BWA. The command to align our reads to the hg19.fa genome is included in the bash script for BWA–MEM Alignment.

4.2. BWA–MEM Alignment

Burrows–Wheeler Aligner is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. It consists of three algorithms: BWA–backtrack, BWA–SW and BWA–MEM. BWA–MEM is the preferred choice for 70bp or longer Illumina, 454, Ion Torrent and Sanger reads, assembly contigs and BAC sequences.

- -t Number of Threads
- -M Required for Picard compatability
- -R Complete read group header line. The read group ID will be attached to every read in the output.

The following bash script was used to firstly index the genome using BWA and Samtools Faidx, then run BWA – MEM on each sample (Both reads – R1, R2) producing a .bam file. The script includes a pipe to Samtools, converting the .sam file to .bam 'Sb'.

My directory was set up as follows for the pipeline:

```
|------ SampleDir
|  |--genome
|  |  |--hg19.fa
|  |
|  |
|  |--reads
|     |--HN51_R1.fastq
|     |--HN51_R2.fastq
|     |--HN60_R1.fastq
|     |--HN60_R2.fastq
|     |--HN72_AC_R1.fastq
|     |--HN72_AC_R2.fastq
|     |--HN72_AH_R1.fastq
|     |--HN72_AH_R2.fastq
```

```
#!/bin/bash

# Your job name
#$ -N dream1

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

#Provide path to genome
```

```
19
20 genome=../genome/hg19.fa
21
22 #Index using BWA, Samtools
23
24 bwa index $genome
25 samtools faidx $genome
26
27 #BWA mem
28
29 bwa mem -t 12 -M
   ↪ -R"@RG\tID:HN51_lane3\tSM:HN51\tLB:HN51\tPU:BD1LYPACXX.HN51\tPL:illumina"
   ↪ $genome HN51_R1.fastq HN51_R2.fastq | samtools view -Sb - >
   ↪ HN51.bam
30
31 bwa mem -t 12 -M
   ↪ -R"@RG\tID:HN60_lane7\tSM:HN60\tLB:HN60\tPU:BD1LYPACXX.HN60\tPL:illumina"
   ↪ $genome HN60_R1.fastq HN60_R2.fastq | samtools view -Sb - >
   ↪ HN60.bam
32
33 bwa mem -t 12 -M
   ↪ -R"@RG\tID:HN72_AC_lane_2\tSM:HN72_AC\tLB:HN72\tPU:AC254KACXX.HN72_AC\tPL:illu
   ↪ $genome HN72_AC_R1.fastq HN72_AC_R2.fastq | samtools view
   ↪ -Sb - > HN72_AC.bam
34
35 bwa mem -t 12 -M
   ↪ -R"@RG\tID:HN72_AH_lane_2\tSM:HN72_AH\tLB:HN72\tPU:AHOLENADXX.HN72_AH\tPL:illu
   ↪ $genome HN72_AH_R1.fastq HN72_AH_R2.fastq | samtools view
   ↪ -Sb - > HN72_AH.bam
```

```
|------ SampleDir
```

```
|  |--genome
|     |--hg19.fa
|     |--hg19.fa.amb
|     |--hg19.fa.ann
|     |--hg19.fa.bwt
|     |--hg19.fa.pac
|     |--hg19.fa.sa
|     |--hg19.fa.fai
|
|  |--reads
|     |--HN51.bam
|     |--HN51_R1.fastq
|     |--HN51_R2.fastq
|     |--HN60.bam
|     |--HN60_R1.fastq
|     |--HN60_R2.fastq
|     |--HN72_AC.bam
|     |--HN72_AC_R1.fastq
|     |--HN72_AC_R2.fastq
|     |--HN72_AH.bam
|     |--HN72_AH_R1.fastq
|     |--HN72_AH_R2.fastq
```

4.3. Samtools Pipeline

After creating the .bam files we can use samtools to run Fixmate, Remove Duplicates, Sort and generate a VCF file. Detailed description of workflow:

- Samtools Faidx: Index reference sequence in the FASTA format or extract subsequence from indexed reference sequence.
- Samtools view: This command can be used to produce a SAM file. This is useful to check the headers and confirm the Read Groups were assigned properly.
- Samtools Fixmate: BWA can sometimes leave unusual FLAG information on SAM records so it is helpful to first clean up flags and fix readpairing.

- Samtools rmdup: Remove potential PCR duplicates. If multiple read pairs have identical external coordinates, only retain the pair with highest mapping quality.
- Samtools Flagstat: Generate a text file containing basic informations on reads.
- Samtools Index: Index sorted alignment for fast random access.
- Samtools Sort: Sort alignments by leftmost coordinates. (Creates multiple temporary files during execution).
- Samtools mpileup: Generate BCF or pileup for one or multiple BAM files. Alignment records are grouped by sample identifiers in @RG header lines.

```
#!/bin/bash

# Your job name
#$ -N Hope

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

#Commands

```

```
genome=../genome/hg19.fa

for b in *.bam;

do

        sam=${b/.bam/.sam}
        fixed=${b/.bam/.fixed.bam}
        rmdups=${fixed/.fixed.bam/.rmd.bam}
        sorted=${rmdups/.rmd.bam/.rmd.sorted.bam}
        mapstats=${b/.bam/.mapstat.txt}
        rawbcf=${sorted/.rmd.sorted.bam/.var.raw.bcf}
        filtered=${rawbcf/.var.raw.bcf/.var.flt.vcf}

        samtools view -h $b > $sam
        samtools fixmate $b $fixed
        samtools rmdup $fixed $rmdups
        samtools sort $rmdups ${sorted//.bam}
        samtools index $sorted
        samtools flagstat $sorted > $mapstats
        samtools mpileup -ugf $genome $sorted | bcftools view
        ↪  -bvcg -> $rawbcf
        bcftools view $rawbcf | vcfutils.pl varFilter -d 60 >
        ↪  $filtered

done
```

4.4. Bash script requirements

Running Samtools/0.1.18 on our files means that you have to use bcftools provided by samtools. DO NOT load bcftools/1.5 on the cluster. Another gotcha is to unload Perl. Trying to test the code, I kept recieving an error at the vcfutils.pl stage : Use of uninitialized value in concatenation (.) or string

at /cm/shared/apps/perl/5.16.2/lib/5.16.2/warnings.pm line 390. Unloading the module perl and running vcfutils.pl, it was able to access vcfutils.

*Tip to user: whilst the script is running, use the command du -sh * This will list the files in the directory, and the size of each file. A very useful command to check your script is running smoothly and the files are the size they should be.*

After this bash script has completed, we have VCF files generated using samtools.

---

5. Picard/GATK

The first part of our report detailed aligning the reads to the reference genome, and using samtools to generate VCF files containing variants. The second part of the report will detail how to use Picard/GATK to generate VCF files.

5.1. BWA–MEM

This time using BWA-MEM, extra care was taken to fill the Read Group string correctly, as it is used downstream by both Picard and GATK. For this reason I did not make use of variables and a for loop, I wanted to be certain that the RG field was correct for each sample. The following bash script was used to align the trimmed reads to the hg19 genome, and the generated sam file is passed as an argument to Picards SortSam to produce a bam file.

```
|--Alignment
|       |--HN51_R1.fastq
|       |--HN51_R2.fastq
|       |--HN60_R1.fastq
|       |--HN60_R2.fastq
|       |--HN72_AC_R1.fastq
|       |--HN72_AC_R2.fastq
|       |--HN72_AH_R1.fastq
```

```
|       |--HN72_AH_R2.fastq
```

```
#!/bin/bash

# Your job name
#$ -N gatkalign

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

#Provide path to genome

genome=../../genome/hg19.fa

#path to tools

picard=/home/pilib/picard-2.8.2.jar

#BWA mem

java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $picard
 ↪  CreateSequenceDictionary $genome O=hg19.dict
```

```
29
30 bwa mem -t 12 -M  \
31 -R"@RG\tID:HN51_lane3\tSM:HN51\tLB:HN51\tPU:BD1LYPACXX.HN51\tPL:illumina"
    ↪  \
32 $genome HN51_R1.fastq HN51_R2.fastq > HN51.sam
33
34 java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $picard SortSam
    ↪  INPUT=HN51.sam OUTPUT=HN51.bam SORT_ORDER=coordinate
35
36
37 bwa mem -t 12 -M \
38 -R"@RG\tID:HN60_lane7\tSM:HN60\tLB:HN60\tPU:BD1LYPACXX.HN60\tPL:illumina"
    ↪  \
39 $genome HN60_R1.fastq HN60_R2.fastq > HN60.sam
40
41 java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $picard SortSam
    ↪  INPUT=HN60.sam OUTPUT=HN60.bam SORT_ORDER=coordinate
42
43
44 bwa mem -t 12 -M \
45 -R"@RG\tID:HN72_AC_lane_2\tSM:HN72_AC\tLB:HN72_AC\tPU:AC254KACXX.HN72_AC\tPL:illum
    ↪  \
46 $genome HN72_AC_R1.fastq HN72_AC_R2.fastq > HN72_AC.sam
47
48 java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $picard SortSam
    ↪  INPUT=HN72_AC.sam OUTPUT=HN72_AC.bam SORT_ORDER=coordinate
49
50
51 bwa mem -t 12 -M \
52 -R"@RG\tID:HN72_AH_lane_2\tSM:HN72_AH\tLB:HN72_AH\tPU:AHOLENADXX.HN72_AH\tPL:illum
    ↪  \
```

```
$genome HN72_AH_R1.fastq HN72_AH_R2.fastq > HN72_AH.sam

java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $picard SortSam
  ↪  INPUT=HN72_AH.sam OUTPUT=HN72_AH.bam SORT_ORDER=coordinate
```

Note on Bash Script: I was unable to figure out a way to run this in a clean, single for loop.

5.2. MarkDuplicates

In this step Picard will mark duplicates in our bam files. The MarkDuplicates step can also be used to merge the HN72 samples which were run on different chips. For this step I experimented with using arrays in my for loop. Firstly, create a text file containing the sorted bam files, one sample per line as such:

```
bamlist.txt:
HN51
HN60
HN72_AC
HN72_AH
```

Using this text file, we can create an array in the bash script to feed into the INPUT and OUTPUT arguments for MarkDuplicates. The bash script contains comments and is self explanatory:

```
#!/bin/bash

# Your job name
#$ -N Markmerge

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd
```

```
11
12 # Export some necessary environment variables
13 #$ -v PATH
14 #$ -v LD_LIBRARY_PATH
15 #$ -v PYTHONPATH
16 #$ -S /bin/bash
17
18 #Set Picard variable
19 picard=/home/pilib/picard-2.8.2.jar
20
21 #Create an array using the bamlist.txt file containing each
   ↪  Sample (per line):
22 oldIFS="$IFS"
23 IFS=$'\n' array=($(<bamlist.txt))
24 IFS="$oldIFS"
25
26 #Assign INPUT argument for each Sample Group to corresponding
   ↪  element in array:
27 HN51="INPUT=${array[0]}"
28 HN60="INPUT=${array[1]}"
29 HN72="INPUT=${array[2]} INPUT=${array[3]}"
30
31 #Test the array:
32 echo "This is for INPUT argument"
33 echo $HN51
34 echo $HN60
35 echo $HN72
36
37 #Assign Sample Group names for Output file:
38 base_HN51="${array[0]:0:4}"
39 base_HN60="${array[1]:0:4}"
```

```
base_HN72="${array[2]:0:4}"

#Test the Basenames:
echo "This is for OUTPUT base names"
echo $base_HN51
echo $base_HN60
echo $base_HN72

#Assign INPUT, BASENAME to new arrays (each 3 elements) so
 ↪  for loop can use both simultaneously:
declare -a inputarray=("$HN51" "$HN60" "$HN72")
declare -a outputarray=("$base_HN51" "$base_HN60" "$base_HN72")

#Let it rip:
for i in ${!inputarray[*]};
do
          java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar
           ↪  $picard \
        MarkDuplicates \
        "${inputarray[$i]}" \
        OUTPUT="${outputarray[$i]}"_dedup.bam \
        METRICS_FILE="${outputarray[$i]}"_dup_metrics.txt
        done
```

5.3. Base Recalibration

The GATK website provides a link to their online bundle, containing known indel and snp sites for human genomes. For some reason when trying to access the bundle I can only view a directory containing hg38 data. To find the indel and snp sites for hg19, a github page provides a link.

On this page, in the second block of code you can extract ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/. You should now have access to the broad institutes bundle, including the hg19 data. Download the

Mills_and_1000G_gold_standard.indels.hg19.sites.vcf & dbsnp_138.hg19.vcf (download corresponding index files).
BaseRecalibration corrects for systematic bias that affect the assignment of base quality scores by the sequencer. The first pass consists of calculating error empirically and finding patterns in how error varies with basecall features over all bases. The relevant observations are written to a recalibration table. The second pass (ApplyBQSR) consists of applying numerical corrections to each individual basecall based on the patterns identified in the first step (recorded in the recalibration table) and write out the recalibrated data to a new BAM file.

```
#!/bin/bash

# Your job name
#$ -N BDSM

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

#path to genome, gatk

genome=../../genome/hg19.fa
gatk=/home/pilib/gatk/gatk-4.0.2.1/gatk-package-4.0.2.1-local.jar

```

```
#BaseRecalibration:

for b in *_dedup.bam; do

        base=$(basename $b _dedup.bam)
        echo "base name is $base"

                java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar
                 ↪  $gatk \
                BaseRecalibrator -I $b -R $genome \
 --known-sites
  ↪  Mills_and_1000G_gold_standard.indels.hg19.sites.vcf \
                --known-sites dbsnp_138.hg19.vcf \
                -O "${base}"_recal.table
        done


#Create an array using the bqsrlist.txt file:
oldIFS="$IFS"
IFS=$'\n' array=($(<bqsrlist.txt))
IFS="$oldIFS"

#Assign input args:
HN51=${array[0]}
HN60=${array[1]}
HN72=${array[2]}

#Assign bqsr-recal-file input args:
HN51_recal=${array[3]}
HN60_recal=${array[4]}
HN72_recal=${array[5]}
```

```
52
53 #Assign output names
54 HN51_base=${array[6]}
55 HN60_base=${array[7]}
56 HN72_base=${array[8]}
57
58 #Make new arrays
59 declare -a inputarray=("$HN51" "$HN60" "$HN72")
60 declare -a bqsrarray=("$HN51_recal" "$HN60_recal"
   ↪  "$HN72_recal")
61 declare -a basename=("$HN51_base" "$HN60_base" "$HN72_base")
62
63 #Run ApplyBQSR:
64
65 for i in ${!inputarray[*]}; do
66
67                 java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar
                    ↪  $gatk ApplyBQSR \
68                 -R $genome \
69                 -I "${inputarray[$i]}" \
70                 --bqsr-recal-file "${bqsrarray[$i]}" \
71                 -O "${basename[$i]}"_recal.bam
72 done
```

Output:

```
HN51_recal.table
HN51_recal.bai
HN51_recal.bam
etc...
```

5.4. BuildBamIndex

Self explanatory

```
#!/bin/bash

# Your job name
#$ -N BBI

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

picard=/home/pilib/picard-2.8.2.jar

for i in *.bam; do

java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $picard
 ↪  BuildBamIndex INPUT=$i

done
```

5.5. Haplotype Caller

For this step, I decided to analyze the exon regions only. I reasoned that in a professional setting, if provided with exome data then we would only be interested in calling variants in the exon regions. Trying to call variants in intron regions will 1, waste a lot of time and 2, cause false positives.

The code below streams in the 'Known gene regions' of hg19 from UCSC browser, isolates the columns 'chromosome' 'chromStart' & 'chromEnd', and writes them to a file. The file it writes to - hg19generegion.txt, is comma separated. Use the sed command to change it to a tab separated file. finally, the tab separated file is converted to a bed file (compatible with samtools mpileup and GATK HaplotypeCaller).

```
curl  -s
↪ "http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/knownGene.txt.gz"
↪ | gunzip -c | awk '{n=int($8);
↪ split($9,S,/,/);split($10,E,/,/); for(i=1;i<=n;++i)
↪ {printf("%s,%s,%s\n",$2,S[i],E[i]) >
↪ ("hg19generegion.txt");} }'

sed 's|,|\t|g' hg19generegion.txt > hg19regionTSV.txt

cut -f 1,2,3 hg19regionTSV.txt > hg19exonregions.bed
```

After some further reading on the -l option to specify regions of interest, I realized I should have used the same option for BQSR. Not enough time to back track and recalibrate.

The following bash script was my attempt to use the 'scatter-gather' method mentioned on forum boards online.

- The first step of the script takes the hg19exonregions.bed file we created in the previous script, and splits it per chromosome (for i in 1..22), creating 22 new bed files per chromosome.
- These individual chromosome.bed files are then fed into HaplotypeCaller in a for loop using an array.
- In theory, I should have 22 g.vcf files for HN51, HN60 & HN72, which I can recombine using 'GatherVCFs'.

```
#!\bin\bash

# Your job name
#$ -N WOW

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

genome=../../genome/hg19.fa
gatk=/home/pilib/gatk/gatk-4.0.2.1/gatk-package-4.0.2.1-local.jar


#Split the know genes .bed file into individual .bed files,
 ↪  per chromosome.

        for i in {1..22}; do

                grep -w chr$i hg19exonregions.bed > chr$i.bed
        done

#Excluded are X, Y chromosomes.

```

```
#Make an array containing the individual chromosomes
oldIFS="$IFS"
IFS=$'\n' array=($(<chrlist.txt))
IFS="$oldIFS"

oldIFS="$IFS"
IFS=$'\n' basearray=($(<basearray.txt))
IFS="$oldIFS"

#My attempt at scatter (perhaps not gather):

for i in ${!array[*]}; do

        java -jar $gatk HaplotypeCaller \
        -R $genome \
        -I HN51_recal.bam \
        -O ./HN51gvcf/HN51_"${basearray[$i]}".g.vcf \
        -ERC GVCF \
        -L "${array[$i]}" \
        #-bamout HN51_"${basearray[$i]}".bam
done

#sample by sample, I can't figure out a way to use one for loop for all 3 samples.

for x in ${!array[*]}; do

        java -jar $gatk HaplotypeCaller \
        -R $genome \
        -I HN60_recal.bam \
        -O ./HN60gvcf/HN60_"${basearray[$x]}".g.vcf \
```

```
        -ERC GVCF \
        -L "${array[$x]}" \
        #-bamout HN60_"${basearray[$x]}".bam
done


########


for z in ${!array[*]}; do

        java -jar $gatk HaplotypeCaller \
        -R $genome \
        -I HN72_recal.bam \
        -O ./HN72gvcf/HN72_"${basearray[$z]}".g.vcf \
        -ERC GVCF \
        -L "${array[$z]}" \
        #-bamout HN72_"${basearray[$z]}".bam
done
```

```
Eg of output directory (HN51gvcf):
HN51_chr1.g.vcf
HN51_chr1.g.vcf.idx
HN51_chr2.g.vcf
HN51_chr2.g.vcf.idx
HN51_chr3.g.vcf
HN51_chr3.g.vcf.idx
HN51_chr4.g.vcf
HN51_chr4.g.vcf.idx
etc...
```

5.6. Combine GVCFs

The second step in our 'scatter gather' method is to consolidate each per chromosome file, per sample. To execute this I generated 3 .list files *(.txt files will not work! You will get the following error message: A USER ERROR has occurred: Cannot read HN51combine.txt because no suitable codecs found.* ***i.e The Locus Walker cannot work with a .txt file****)*. The .list files contain the path to each samples scattered gvcf files. We can then use a for loop to iterate over each .list file:

```
#!/bin/bash

# Your job name
#$ -N combinegvcfs

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

#path to genome, GATK:
gatk=/home/pilib/gatk/gatk-4.0.2.1/gatk-package-4.0.2.1-local.jar
genome=../../genome/hg19.fa

#Create .list file with path to each scattered GVCFs DIR:

```

```
find HN51gvcf -name "*g.vcf" > HN51combine.list

find HN60gvcf -name "*g.vcf" > HN60combine.list

find HN72gvcf -name "*g.vcf" > HN72combine.list

#Create for loop iterating over each .txt file:

for t in *combine.list; do

        base=$(basename $t combine.list)

        java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $gatk
        ↪  CombineGVCFs \
        -R $genome \
        --variant $t \
        -O "${base}".cohort.g.vcf

done
```

```
Output:
HN51.cohort.g.vcf
HN51.cohort.g.vcf.idx
HN60.cohort.g.vcf
HN60.cohort.g.vcf.idx
HN72.cohort.g.vcf
HN72.cohort.g.vcf.idx
```

5.7. Genotype GVCFs

Perform joint genotyping on one or more samples pre-called with HaplotypeCaller. This was performed with a for loop:

```
#!/bin/bash

# Your job name
#$ -N Genogvcfs

# The job should be placed into the queue 'all.q'
#$ -q all.q

# Running in the current directory
#$ -cwd

# Export some necessary environment variables
#$ -v PATH
#$ -v LD_LIBRARY_PATH
#$ -v PYTHONPATH
#$ -S /bin/bash

#path to genome, GATK:
gatk=/home/pilib/gatk/gatk-4.0.2.1/gatk-package-4.0.2.1-local.jar
genome=../../genome/hg19.fa

#Use for loop:

for i in *.cohort.g.vcf; do

        base=$(basename $i .cohort.g.vcf)

        java -Xms10g -Xmx20g -Djava.io.tmpdir=tmp -jar $gatk
        ↪  GenotypeGVCFs \
        -R $genome \
        -V $i \
```

```
31          -O "${base}".vcf
32
33  done
```

```
Output: (.idx files also generated)
HN51.vcf
HN60.vcf
HN72.vcf
```

This concludes the reports step-by-step generation of VCF files using Samtools and Picard/GATK.

6. Analysis

A 'results' directory was created at ../11384191/2018/assessment3/aNewHope/results/:

```
results/
|-- gatk
|   |-- HN51.vcf
|   |-- HN51.vcf.idx
|   |-- HN51_recal.bai
|   |-- HN51_recal.bam
|   |-- HN60.vcf
|   |-- HN60.vcf.idx
|   |-- HN60_recal.bai
|   |-- HN60_recal.bam
|   |-- HN72.vcf
|   |-- HN72.vcf.idx
|   |-- HN72_RG.bai
|   `-- HN72_RG.bam
`-- samtools
    |-- HN51.rmd.sorted.bam
    |-- HN51.rmd.sorted.bam.bai
    |-- HN51.var.flt.vcf
    |-- HN51.var.flt.vcf.idx
    |-- HN60.rmd.sorted.bam
    |-- HN60.rmd.sorted.bam.bai
    |-- HN60.var.flt.vcf
    |-- HN60.var.flt.vcf.idx
```

```
    |-- HN72.rmd.sorted.bam
    |-- HN72.rmd.sorted.bam.bai
    |-- HN72.var.flt.vcf
    |-- HN72.var.flt.vcf.idx
```

6.1. Merging VCFs

The version of GATK we are using does not have 'Combine VCFs'. use the following commands to generate a merged VCF file, which we can use to draw statistics from. The -p option specifies a vcf input file (gff is default).

```
bgzip *.vcf
tabix -p vcf *.vcf.gz

vcf-merge *.vcf.gz >| merged1.vcf.gz (GATK)
vcf-merge *.vcf.gz >| merged2.vcf.gz (Samtools)
```

I ran into some trouble trying to call 'vcf-compare' on these files, it complains the files were not gunzipped (dont trust every forum board online..) I had to use the following commands to backtrack:

```
mv merged1.vcf.gz merged.vcf

$ module load bcftools

bcftools -Oz -o compressedGATK.vcf.gz merged.vcf

bcftools index compressedGATK.vcf.gz

(carry out same commands on merged Samtools file)

vcf-compare compressedSAM.vcf.gz compressedGATK.vcf.gz | grep
    "^VN" > comparison.cmp
```

The comparison.cmp file shows some percentage statistics:

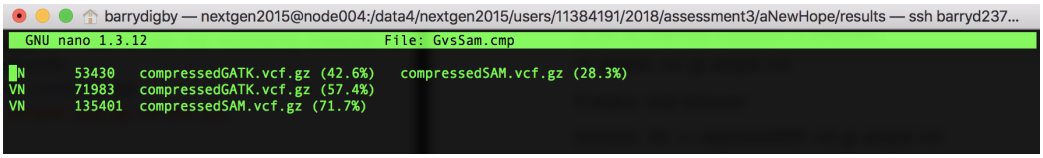


Figure 5: output from vcf-compare

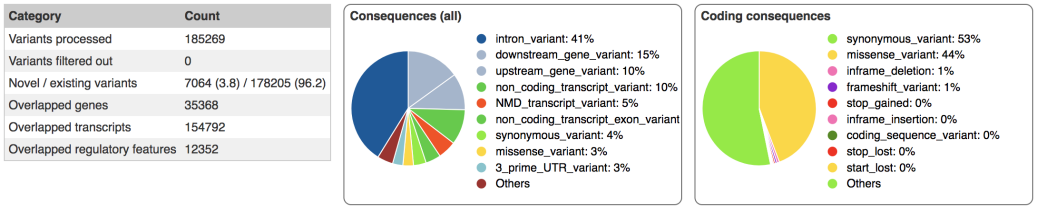
I could not make sense of this so decided to download each samples (gatk and samtools) summary text file from VEP. From here, I isolated the "gene name" column for each sample:

```
cut -f HN51gatk.txt > HN51G.txt
```

Comparing each samples genes against eachother, I was able to generate venn diagrams.

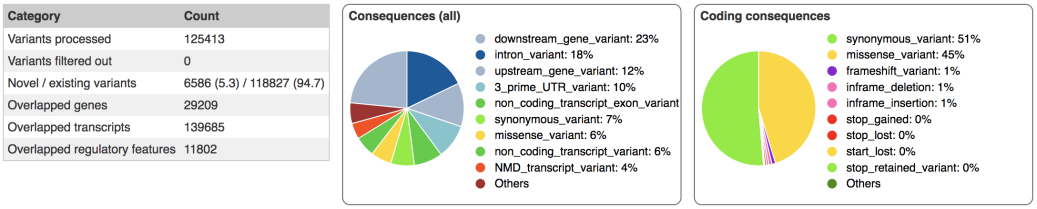
7. Variant Calling Comparison

7.1. Merged VCFs



Category	Count
Variants processed	185269
Variants filtered out	0
Novel / existing variants	7064 (3.8) / 178205 (96.2)
Overlapped genes	35368
Overlapped transcripts	154792
Overlapped regulatory features	12352

Figure 6: Samtools



Category	Count
Variants processed	125413
Variants filtered out	0
Novel / existing variants	6586 (5.3) / 118827 (94.7)
Overlapped genes	29209
Overlapped transcripts	139685
Overlapped regulatory features	11802

Figure 7: GATK

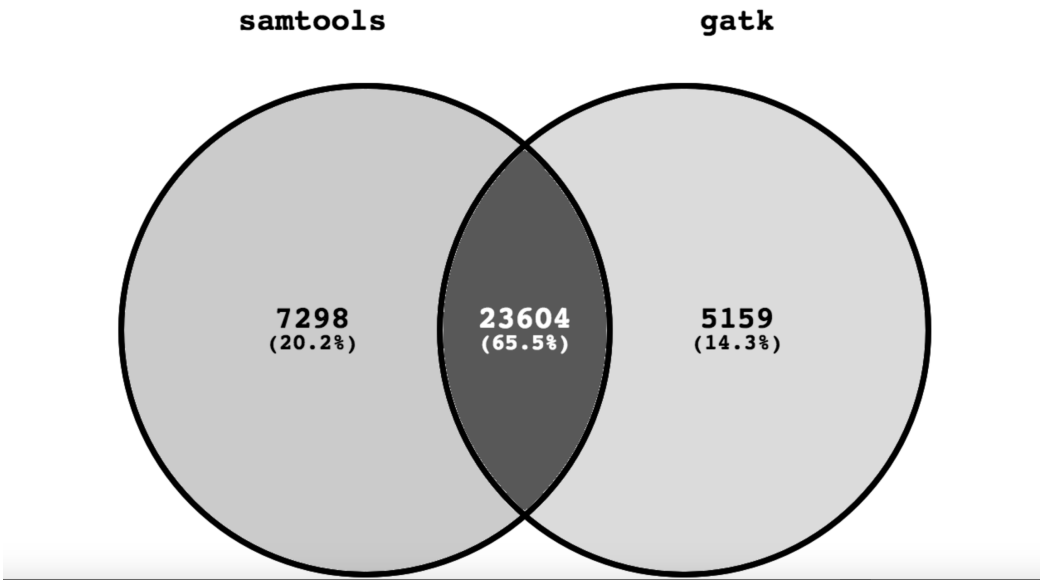
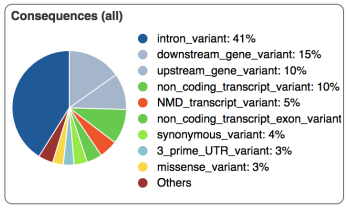


Figure 8: Samtools vs GATK merged samples

7.2. HN51 Sample

Category	Count
Variants processed	107513
Variants filtered out	0
Novel / existing variants	3844 (3.6) / 103669 (96.4)
Overlapped genes	29259
Overlapped transcripts	131818
Overlapped regulatory features	8170



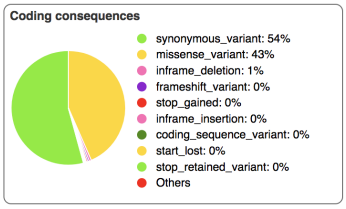
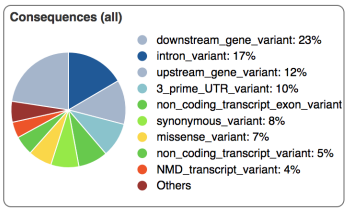


Figure 9: Samtools

Category	Count
Variants processed	58492
Variants filtered out	0
Novel / existing variants	3093 (5.3) / 55399 (94.7)
Overlapped genes	22563
Overlapped transcripts	108789
Overlapped regulatory features	6317



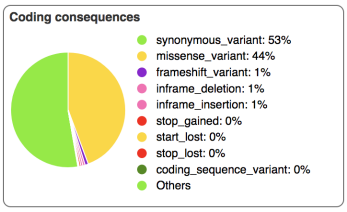


Figure 10: GATK

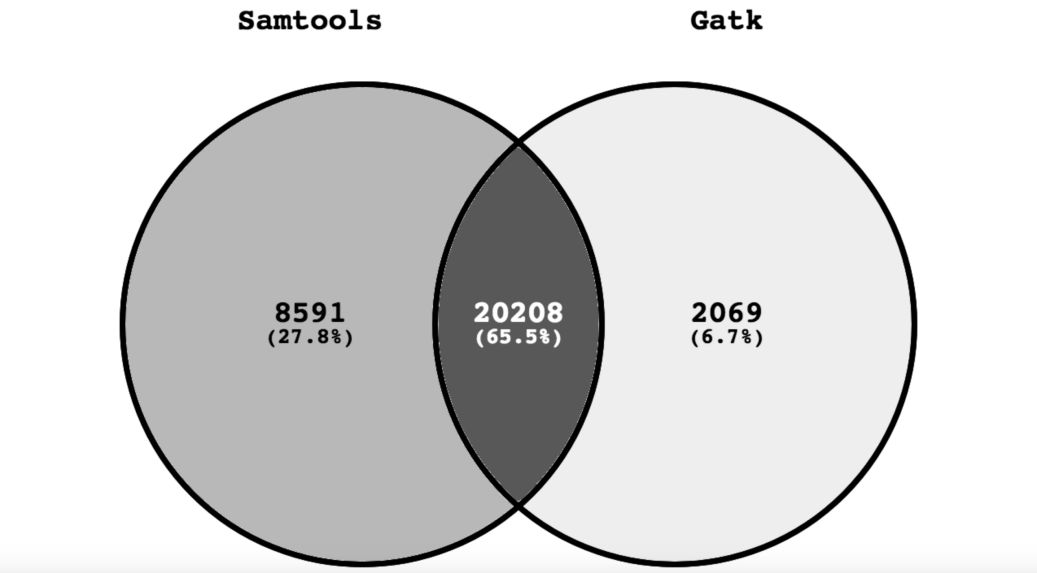


Figure 11: Samtools vs GATK HN51 sample

7.3. HN60 Sample

Category	Count
Variants processed	122225
Variants filtered out	0
Novel / existing variants	4141 (3.4) / 118084 (96.6)
Overlapped genes	31528
Overlapped transcripts	141126
Overlapped regulatory features	8650

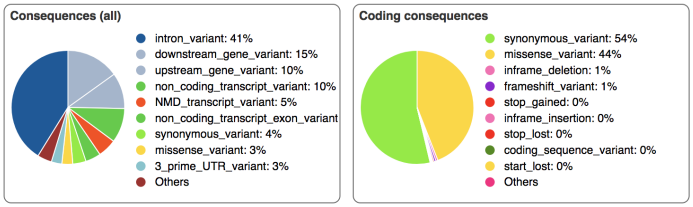


Figure 12: Samtools

Category	Count
Variants processed	67589
Variants filtered out	0
Novel / existing variants	3415 (5.1) / 64174 (94.9)
Overlapped genes	24149
Overlapped transcripts	116955
Overlapped regulatory features	7030

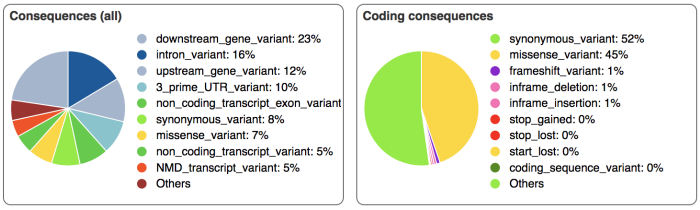


Figure 13: GATK

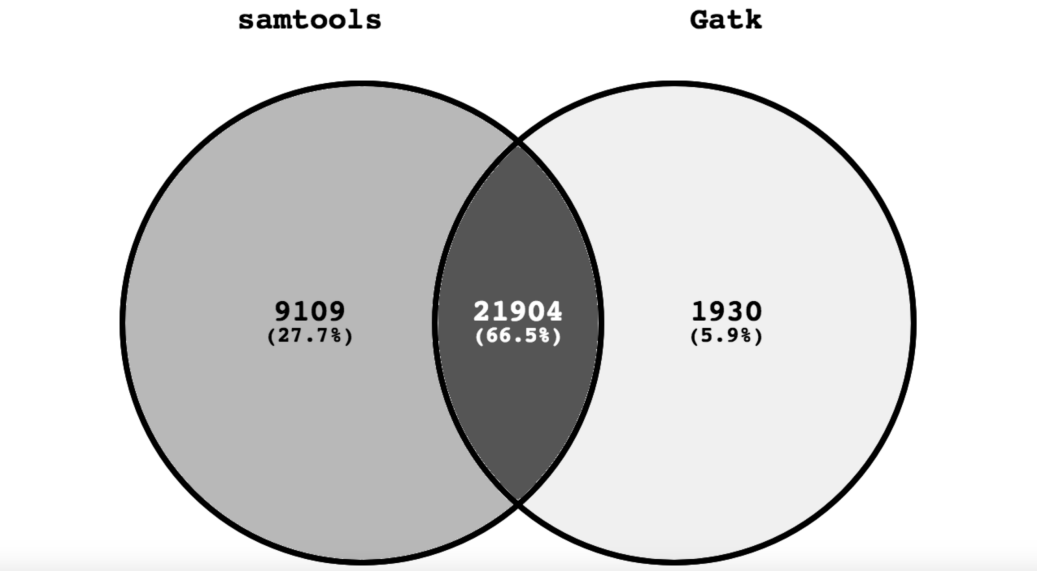


Figure 14: Samtools vs GATK HN60 samples

7.4. HN72 Sample

Category	Count
Variants processed	21187
Variants filtered out	0
Novel / existing variants	1488 (7.0) / 19699 (93.0)
Overlapped genes	8052
Overlapped transcripts	26801
Overlapped regulatory features	1526

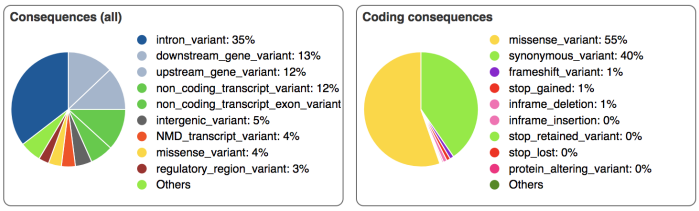


Figure 15: Samtools

Category	Count
Variants processed	72492
Variants filtered out	0
Novel / existing variants	3502 (4.8) / 68990 (95.2)
Overlapped genes	24209
Overlapped transcripts	113849
Overlapped regulatory features	7470

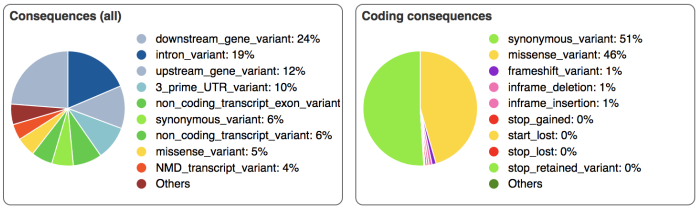


Figure 16: GATK

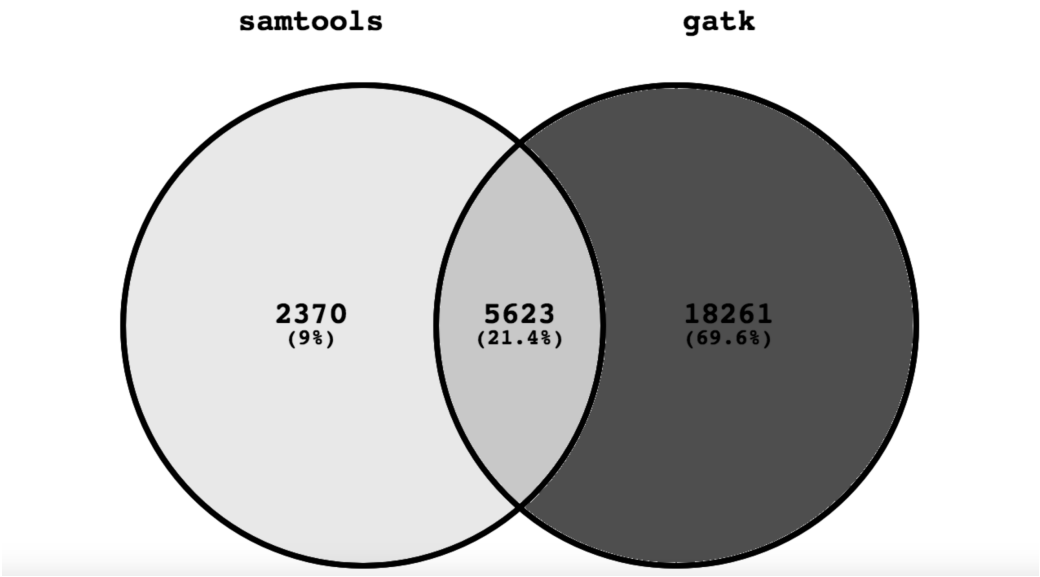


Figure 17: Samtools vs GATK HN72 samples

8. Top 5 HNSCC genes

8.1. TP53

Having identified the top 5 genes involved with HNSCC, I used IGV to check if samtools or GATK were able to call known variants by cross referencing with snps on genecards.org. For the TP53 gene, we can see that the snp rs1042522 is at position chr17:7,579,472. Per 'MedGen', this snp is classed as 'Hereditary cancer-predisposing syndrome'.

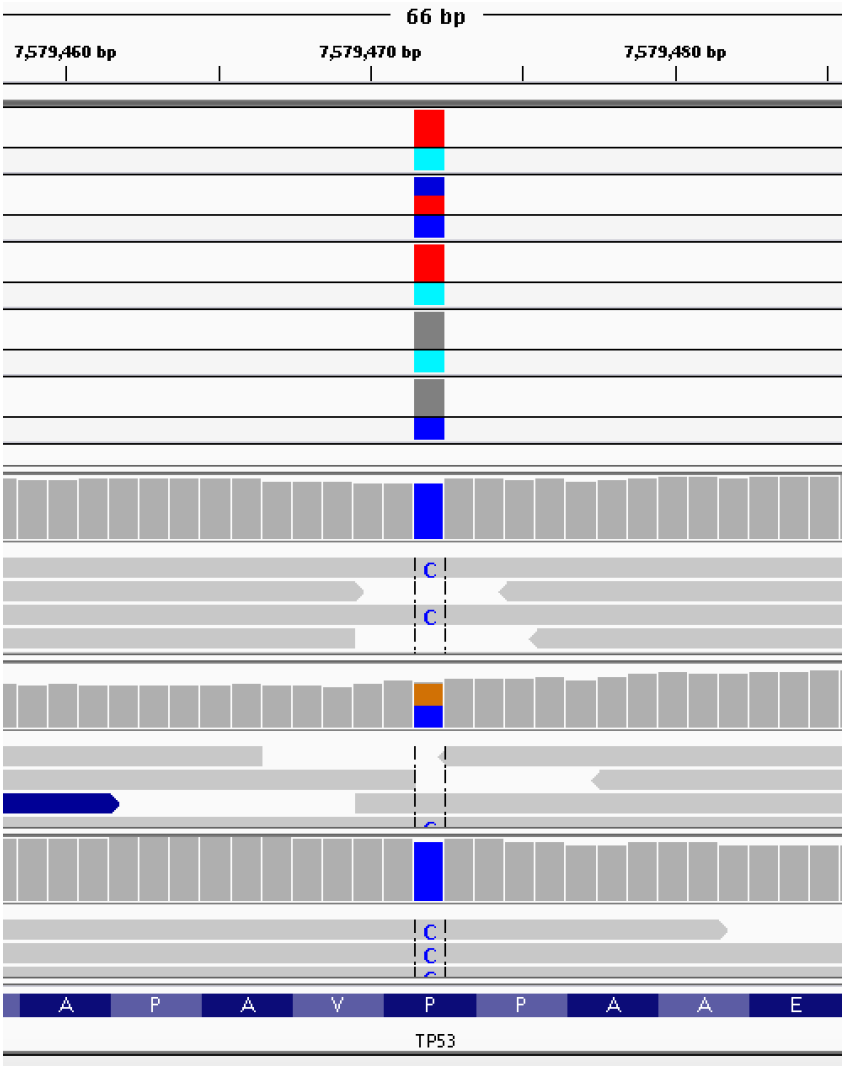


Figure 18: TP53 variants

8.2. TTN

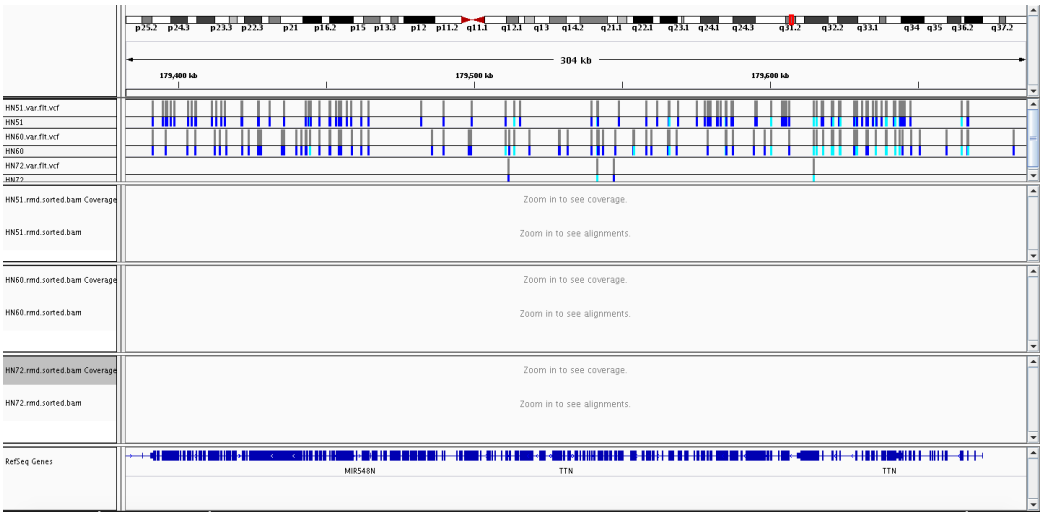


Figure 19: TTN variants

No variants listed in genecards.org matched to the variants observed in IGV.

8.3. FAT1

No snps listed on genecards.org were of any clinical significance.

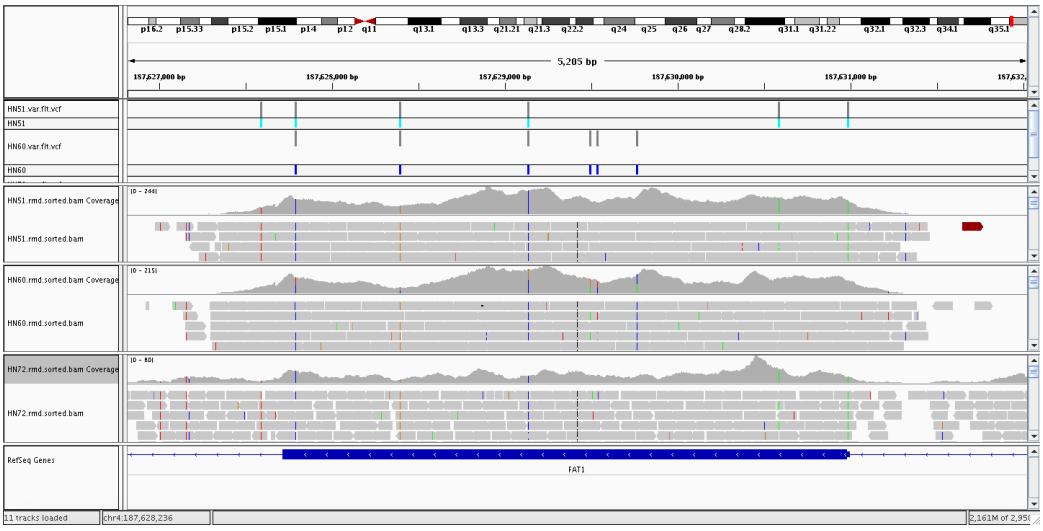


Figure 20: FAT1 variants

8.4. MUC16

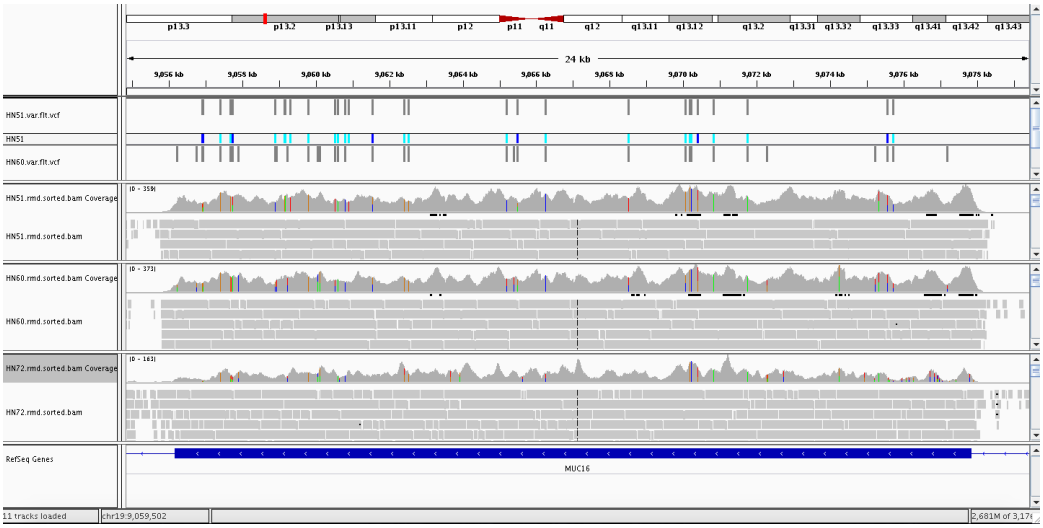


Figure 21: MUC16 variants

Again, none of the top snps with any clinical relevance / validation were called in our samples.

8.5. CDKN2A

The CDKN2A gene displayed only one variant (common across samples) snp rs11515. According to literature this snp is not associated with HNNSC but may be involved in breast cancers. Another paper refuted its role in cancer development, hypothesizing at most it may be involved in cancer risk amongst Asian populations.

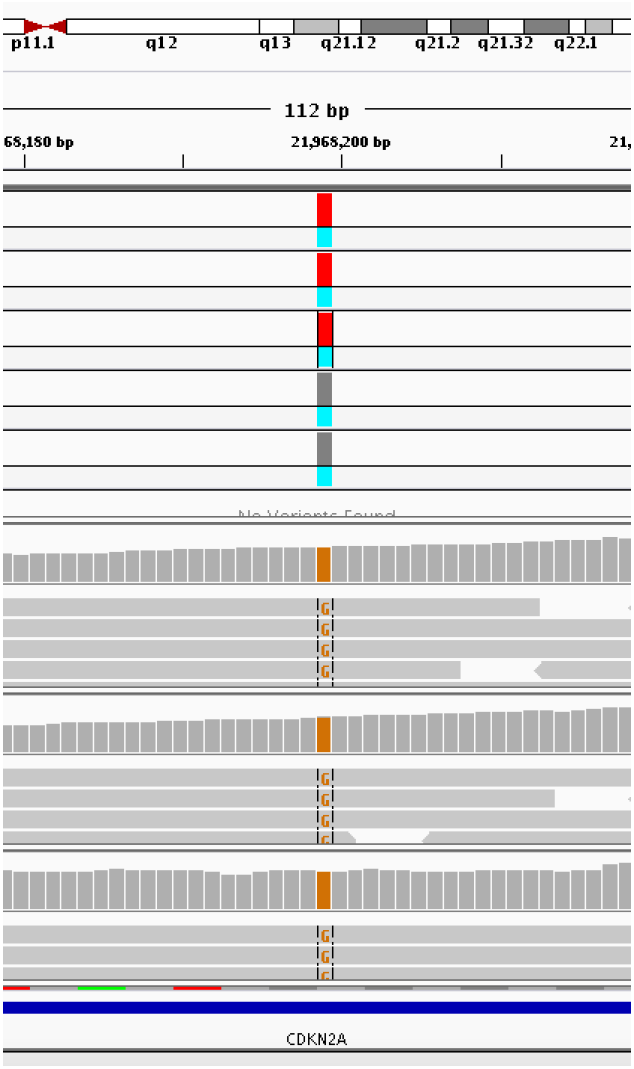


Figure 22: rs11515

8.6. CSMD3

No variants of clinical significance.

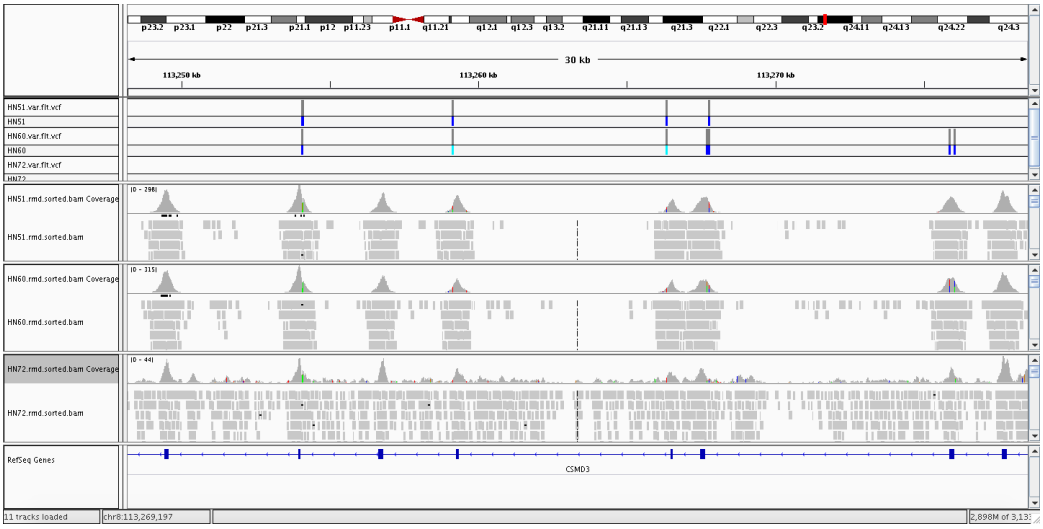


Figure 23: CSMD3

9. Comments

9.1. Bash Scripts

My bash script from section 4.3 (bam to vcf large script) had a bug. The BCF and VCF files generated from the samtools mpileup step would have headers, but otherwise were empty files. However when I extracted the line of code to generate a BCF and ran it in a seperate bash script (still using a for loop), it worked fine. The same was true for the second step BCF to VCF.

9.2. Venn Diagram

I was not able to generate any visual outputs using vcf-compare. Reading online, the code

```
vcf-compare -p prefix *.vcf.gz
```

is supposed to generate output prefix.png file with summary statistics. The files I generated were empty.