Variant Calling

Viz

18234929

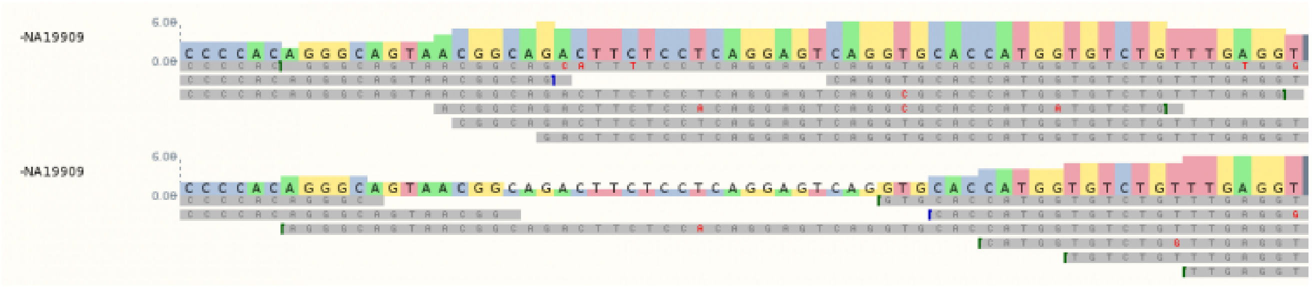
Introduction

Variant calling is a method which is used to call the variants between sequences/samples. It can be used to map the common base-pair differences in DNA sequences among induviduals of a population. Some variants may be common across all populations while some variants are exclusive to certain populations. These variations can be inherited (germline variants) or developed (somatic variants). Mutations and recombinations create the diversity of the variants.

- ### Mutations
  - A mutation is a permanent alteration to a DNA sequence. De novo (new) mutations occur when there is an error during DNA replication that is not corrected by DNA repair enzymes. It is only once the error is copied by DNA replication, and fixed in the DNA that it is considered to be a mutation. Mutations may be beneficial to the organism; deleterious (harmful) to the organism; or neutral (have no effect on the fitness of the organism). Somatic mutations can accumulate in our cells and are mostly harmless. They can lead to local changes in tissues such as moles appearing on the skin, and can also have more serious effects - for example leading to cancer.
- ### Recombination
  - Recombination is another major source of genetic variation Each of us has a mixture of genetic material from our parents. The mixing of this genetic material occurs during recombination when homologous DNA strands align and cross over. Recombination effectively 'shuffles' maternal and paternal DNA, creating new combinations of variants in the daughter germ-cells.

Process

1. Whole genome or whole exome is sequenced to create FASTQ files.
2. Alignment of sequences to a reference genome, creating BAM or CRAM files.
3. Identify where the aligned reads differ from the reference genome and write to a VCF file.



Somatic VC vs Germline VC

- In germline variant calling,

    - The reference genome is the standard for the species of interest.
    - This allows us to identify genotypes
    - As most genomes are diploid, we expect to see that at any given locus, either all reads have the same base, indicating homozygosity, or approximately half of all reads have one base and half have another, indicating heterozygosity. An exception to this would be the sex chromosomes in male mammals.
- In somatic variant calling,
    - The reference is a related tissue from the same individual.
    - We expect to see mosaicism between cells.

VCF Format

VCF is the standard file format for storing variation data. These files include variant identifiers which are unique combinations of letters and numbers that are assigned to known entities, such as genes, variants and proteins. Variant identifiers are particularly useful when searching for information about a variant because they are unambiguous, unique and stable, unlike descriptive names, which can be used differently by different people, be identical between species and change over time.Variants may have identifiers from multiple databases.Different types of identifiers are used for short variants and for structural variants.

Analysis

Variants can be analysed in different ways. In order to determine which genes the variants hit and what effects they have, tools such as the Ensembl VEP and SnpEff can be used for this. The raw VCF files should be filtered and merged before analysis (This also reduces file size)

Further analysis can be performed using

- Ensembl
- Uniprot
- GWAS Catalog
- European Variation Archive (EVA)

This can be followed by predicting the effect of variation on protein structure and function.

Types of studies and further information is listed here.

Data Provided

- FASTQ files generated from ***100bp PE*** libraries on an Illumina HiSeq machine (2 samples)
    - HN50
        - HN51_S2_normal.BD1LYPACXX.lane_3_P1_I12.hg19.sequence.fastq
        - HN51_S2_normal.BD1LYPACXX.lane_3_P2_I12.hg19.sequence.fastq
    - HN60
        - HN60_s2_normal.BD1LYPACXX.lane_7_P1_I16.hg19.sequence.fastq
        - HN60_s2_normal.BD1LYPACXX.lane_7_P2_I16.hg19.sequence.fastq
- Indexed genome
    - hg19.fa [FASTA format]

Pipeline

The pipeline consists of 5 scripts [1 main & 4 subscripts]

Main Script

Calls subscripts and runs the whole pipeline

```
#!/bin/sh
#$ -N VAR_CALL
#$ -q all.q
#$ -cwd
#$ -S /bin/bash
#$ -v PATH
#$ -v LD_LIBRARY_PATH

#[P] {script runs commands in parallel}

##MAIN SCRIPT
##ENTRYPOINT
##Runs the variant calling pipeline [both {samtools,bcftools},{gatk}]

proc_ids=() ##Used to store process IDs

##SUBSCRIPT-fast_qc.sh[P] [Runs fastqc on samples]
##REQUIRES user validation of output
nohup fast_qc.sh /data4/nextgen2015/pilib/MA5112_Assign3/
wait $!

##Following lines are run after fast_qc.sh output validation
##no need to trim adapters because adapter contamination is < 0.1% according to fastqc

##SUBSCRIPT-bwa.sh[P] [Runs BWA for the samples]
##WAIT until user validation of fast_qc.sh output
nohup bwa.sh
wait $!

##SUBSCRIPT-vc.sh[P] [Variant calling-{samtools,bcftools}]
##DEPENDS on bwa.sh output
nohup vc.sh
proc_ids+=("$!")

##SUBSCRIPT-vc.gatk.sh[P] [Variant calling[P]-{gatk}]
##DEPENDS on bwa.sh output
nohup vc.gatk.sh HN51 HN60 #sample file prefixes
proc_ids+=("$!")

for i in ${proc_ids[@]}; do
    wait $i
done

echo "Done!"
```

Quality check using FASTQC

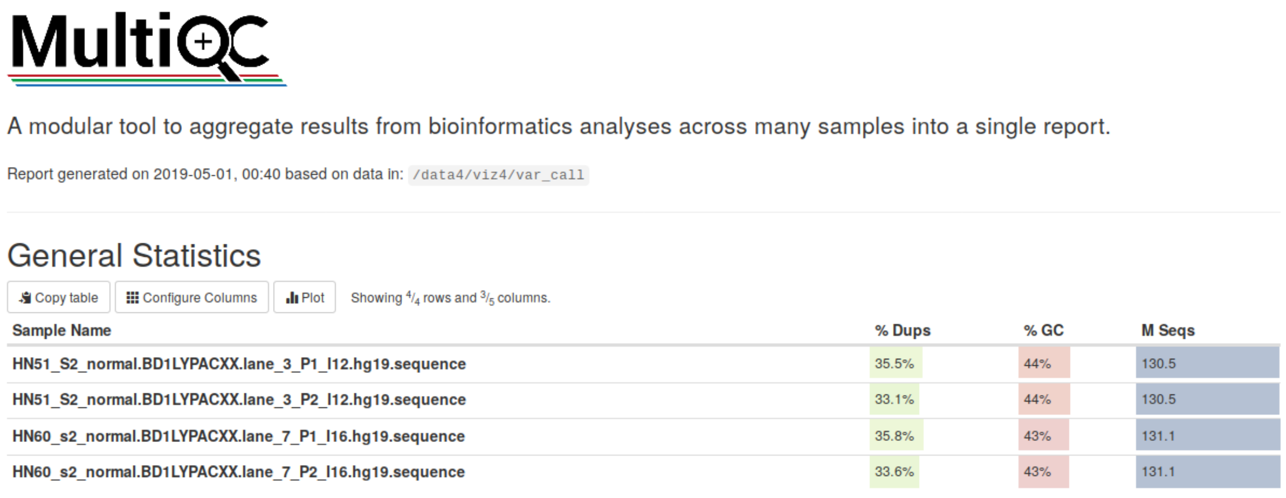
Initially, the FASTQ files are subject to quality checks using ***fastqc***

```
#!/bin/sh

#module load python
#module load fastqc

show_spinner()
{
  local -r pid="${1}"
  local -r delay='0.75'
  local spinstr='\|/-'
  local temp
  while ps a | awk '{print $1}' | grep -q "${pid}"; do
    temp="${spinstr#?}"
    printf " [%c]  " "${spinstr}"
    spinstr=${temp}${spinstr%"${temp}"}
    sleep "${delay}"
    printf "\b\b\b\b\b\b"
  done
  printf "    \b\b\b\b"
}

##FUNCTIONS:
paralellize_fastqc() ##Runs fastqc on each sample file
{
nohup fastqc -o "./" $1 &> "./"$1.out& ##fastqc & write output to directory of the bash
script
proc_ids+=($!) ##Store process IDs
}

##STANDALONE SCRIPT/SUBSCRIPT
##ENTRYPOINT
##Check paramter count ($#)
if [ $# -eq 0 ]
 then
    "Please give path to fastq files followed by file extension(without .) for analysis"
    exit 1
fi

proc_ids=() ##Used to store process IDs
#DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"

##Set extension to look for
##Note: This is a stupid overhaul
if [ -z $2 ]
then
   ext="*"

else
```

```
    ext=$2
fi

cd "$1" ##cd to given dir

##if cd unsuccessful then dir doesn't exist
if [ $? != 0 ]
then
  echo "Check file path"
  exit 1
fi

##Print info
echo "Dir: $(pwd)"
echo "Ext: $ext"
echo "Files: $(ls)"

##Choose files by extension & run fastqc
for f in *.$ext; do
  echo "File Chosen-> $f"
  paralellize_fastqc $f ##Called for each file
done

##Wait for processes to complete
for id in ${proc_ids[@]}; do
  ##show_spinner $id
  wait $id
done

echo $?
multiqc "./" ##Run multiqc on directory
```

FASTQC analyses the 4 files from 2 samples and provides fastqc (*html*) output for each file. ***MultiQC*** aggregates all the fastqc output files into a single html file.

MultiQC Report



MultiQC

A modular tool to aggregate results from bioinformatics analyses across many samples into a single report.

Report generated on 2019-05-01, 00:40 based on data in: `/data4/viz4/var_call`

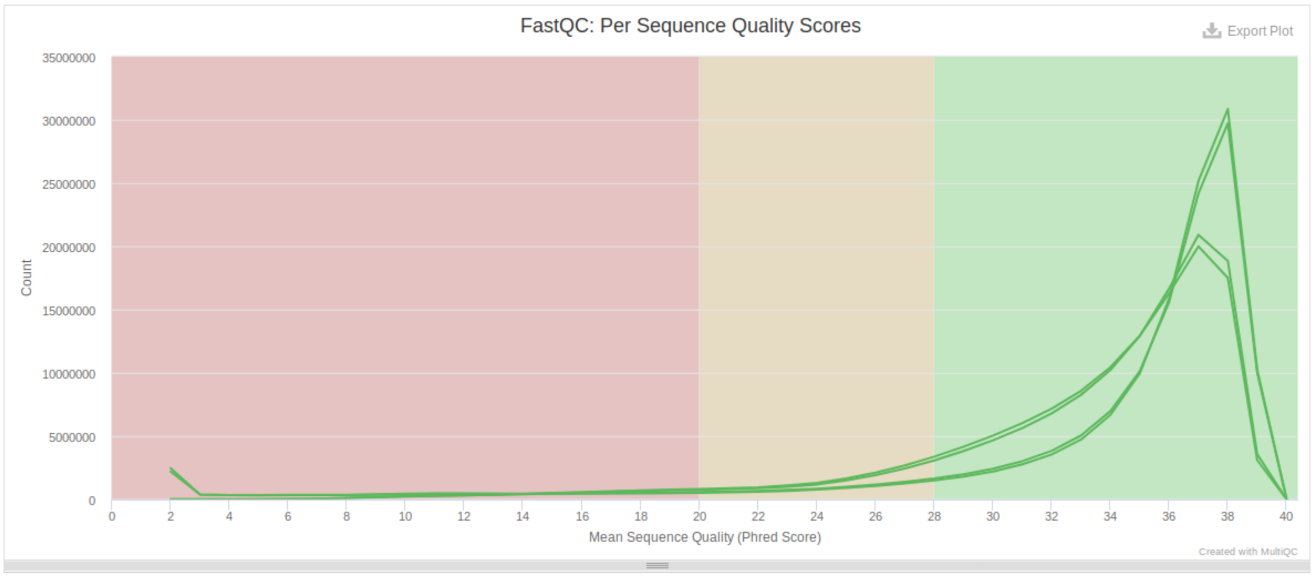
General Statistics

Showing $^4/_4$ rows and $^3/_5$ columns.

Sample Name	% Dups	% GC	M Seqs
HN51_S2_normal.BD1LYPACXX.lane_3_P1_I12.hg19.sequence	35.5%	44%	130.5
HN51_S2_normal.BD1LYPACXX.lane_3_P2_I12.hg19.sequence	33.1%	44%	130.5
HN60_s2_normal.BD1LYPACXX.lane_7_P1_I16.hg19.sequence	35.8%	43%	131.1
HN60_s2_normal.BD1LYPACXX.lane_7_P2_I16.hg19.sequence	33.6%	43%	131.1

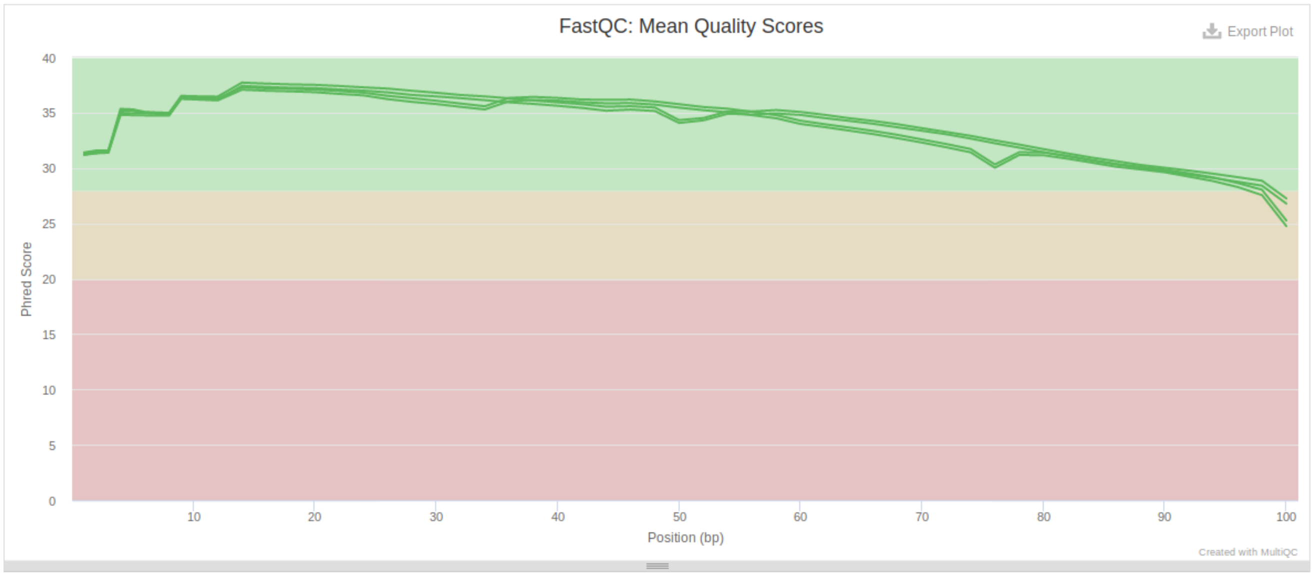
Per Sequence Quality Scores

The number of reads with average quality scores. Shows if a subset of reads has poor quality.



Sequence Quality Histograms

The mean quality value across each base position in the read.



Adapter Content

The cumulative percentage count of the proportion of your library which has seen each of the adapter sequences at each position.

No samples found with any adapter contamination > 0.1%

From the above reports we can determine that the ***sequences are of good quality and have less than 0.1 % of adapter contamination, therefore adapter trimming and low quality reads removal can be skipped.***

Alignment using BWA

BWA is a software package for mapping low-divergent sequences against a large reference genome. It consists of three algorithms:

- BWA-backtrack

- BWA-SW
- BWA-MEM

BWA-MEM is recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads. Thus BWA-MEM is chosen.

Note: The reference genome is already indexed hence indexing step can be skipped.

```
#!/bin/bash

#module load bwa

##SUBSCRIPT
##ENTRYPOINT
##Runs bwa on samples
##WAIT until user has validated fast_qc.sh output

proc_ids=() ##Used to store process IDs
##Runs bwa mem in background on HN51 sequences
nohup bwa mem -t 16 -R '@RG\tID:HN51\tSM:BD1LYPACXX.lane_3_I12\tLB:Lib1'
/data4/pilib/genomes/hs/hg19/hg19.fa
/data4/nextgen2015/pilib/MA5112_Assign3/HN51_S2_normal.BD1LYPACXX.lane_3_P1_I12.hg19.seq
uence.fastq
/data4/nextgen2015/pilib/MA5112_Assign3/HN51_S2_normal.BD1LYPACXX.lane_3_P2_I12.hg19.seq
uence.fastq | samtools view -Sb - > HN51.bam&
proc_ids+=("$!") ##Store process ID
##Runs bwa mem in bg on HN60 sequences
nohup bwa mem -t 16 -R '@RG\tID:HN60\tSM:BD1LYPACXX.lane_7_I16\tLB:Lib2'
/data4/pilib/genomes/hs/hg19/hg19.fa
/data4/nextgen2015/pilib/MA5112_Assign3/HN60_s2_normal.BD1LYPACXX.lane_7_P1_I16.hg19.seq
uence.fastq
/data4/nextgen2015/pilib/MA5112_Assign3/HN60_s2_normal.BD1LYPACXX.lane_7_P2_I16.hg19.seq
uence.fastq | samtools view -Sb - > HN60.bam&
proc_ids+=("$!") ##Store process ID

##Wait until all processes are complete
for id in ${proc_ids[@]}; do
  wait $id
done
```

```
[main] Version: 0.7.12-r1039
[main] CMD: bwa mem -t 16 -R @RG\tID:HN60\tSM:BD1LYPACXX.lane_7_I16\tLB:Lib2
/data4/pilib/genomes/hs/hg19/hg19.fa
/data4/nextgen2015/pilib/MA5112_Assign3/HN60_s2_normal.BD1LYPACXX.lane_7_P1_I16.hg19.seq
uence.fastq
/data4/nextgen2015/pilib/MA5112_Assign3/HN60_s2_normal.BD1LYPACXX.lane_7_P2_I16.hg19.seq
uence.fastq
[main] Real time: 19159.679 sec; CPU: 106941.705 sec
```

It aligns the HN51 & HN60 samples agains human hg19 reference genome. Paired-End read files can be provided one after the other to BWA for alignment. The aligned output(*Sequence Aligned format file (or).sam file*) is streamed to samtools and compressed on-the-fly to *.bam* files. This is facilitated by the *samtools view* command.

Post-Processing using SAMTOOLS

Samtools are a collection of utilities for manipulating Sequence Alignment/Map (*.SAM*) format. It imports from and exports to the SAM format, does sorting, merging and indexing, and allows to retrieve reads in any regions swiftly.

The BAM files from BWA are the post-processed to

- Fix 'mate-pairing' information for paired end reads (*fixmate*)
- Sort the aligned reads (*sort*)
- Remove duplicate reads (*rmdup*)
- Index the aligned reads (*index*)

```
samtools fixmate $1.bam $1.fixed.bam
samtools sort $1.fixed.bam $1.sorted ##Sort bam file
samtools rmdup $1.sorted.bam $1.sorted.rmdup.bam ##Remove duplicates
samtools index $1.sorted.rmdup.bam ##Index bam files
```

Variant Calling using BCFTOOLS

Bcftools is a collection of utilities used for variant calling and manipulating VCFs. The sorted and duplicates removed BAM files are then provided to BCFTOOLS for multi-way piling up of reads. The *mpileup* utility provides a summary of the coverage of mapped reads on a reference sequence at a single base pair resolution.

```
bcftools mpileup -f ref.fa -s l100_n1000_d300_31_1.bam |
sed -n '500,515p'
[mpileup] 1 samples in 1 input files
<mpileup> Set max per-file depth to 8000
10000 515 C 25  ..,...........,..,..,.,., JJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 516 G 25  ..,...........,..,..,.,., JJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 517 T 25  ..,...........,..,..,.,., JJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 518 T 25  ..,...........,..,..,.,., JJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 519 C 25  ..,...........,..,..,.,., JJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 520 G 26  .$.,...........,..,..,.,.,^], >JJJJJJJJJJJJJJJJJJJJJJJJ@
]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 521 G 26  .,...........,..,..,.,.,,^].  DJJJJJJJJJJJJJJJJJJJJJJJJA7
]]]]]]]]]]]]]]]]]]]]]]]]]]


10000 522 T 25  ,...........,..,..,.,.,,. JJJJJJJJJJJJJJJJJJJJJJJJA7
```

```
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 523 A 25  tTTTTTTTTTTTTtTTtTTtTtTttT JJJJJJJJJJJJJJJJJJJJJJJA7
]]]]]]]]]]]]]]]]]]]]]]]]]
10000 524 C 27  ,...........,..,..,.,.,,... JJJJJJJJJJJJJJJJJJJJJJJJG88
]]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 525 A 27  ,...........,..,..,.,.,,... JJJJJJJJJJJJJJJJJJJJJJJJJHH
]]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 526 A 27  ,...........,..,..,.,.,,... JJJJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 527 A 27  ,...........,..,..,.,.,,... JJJJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 528 C 27  ,...........,..,..,.,.,,... JJJJJJJJJJJJJJJJJJJJJJJJJJJ
]]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 529 T 28  ,$...........,..,..,.,.,,...^], >JJJJJJJJJJJJJJJJJJJJJJJJJJB
]]]]]]]]]]]]]]]]]]]]]]]]]]]]
10000 530 T 27  ...........,..,..,.,.,,..., JJJJJJJJJJJJJJJJJJJJJJJJJJE
]]]]]]]]]]]]]]]]]]]]]]]]]]]
```

The above is an example output of *samtools mpileup* command. BCFTOOLS has its own mpileup utilty which we will be using. This 'depth coverage' output can be piped to BCFTOOLS for variant calling.

```
#!/bin/sh
#$ -N SAMTOOLS
#$ -q all.q
#$ -cwd
#$ -S /bin/bash
#$ -v PATH
#$ -v LD_LIBRARY_PATH

#module load samtools
#module load bcftools

##FUNCTIONS:
parallel_samtools() ##Runs variant calling in parallel
{
##No need to index reference seqs using samtools faidx because they're already indexed
##fixmate checks the two mates from a paired-end bam (name sorted) and then updates the
flags and insert sizes
samtools fixmate $1.bam $1.fixed.bam
samtools sort $1.fixed.bam $1.sorted ##Sort bam file
samtools rmdup $1.sorted.bam $1.sorted.rmdup.bam ##Remove duplicates
samtools index $1.sorted.rmdup.bam ##Index bam files
##multi-way 'pile' seqs based on read groups
bcftools mpileup -O u --threads 16 -f /data4/pilib/genomes/hs/hg19/hg19.fa
$1.sorted.rmdup.bam | bcftools call -m  > $1.raw.vcf
#coverage depth formula is : (read count*Avg read length)/total bases
bcftools view $1.raw.vcf | vcfutils.pl varFilter -d 10 > $1.flt.vcf ##Filter variants
}

##SUBSCRIPT
##ENTRYPOINT
##Runs variant calling
##WAIT for bwa.sh output
```

```
proc_ids=() ##Used to store process IDs

##Alternate way of saying "nohup" because nohup doesn't directly support calling
functions in a script
( trap "true" HUP ; parallel_samtools HN51 ) &> ./HN51.sam.out &
proc_ids+=("$!")
( trap "true" HUP ; parallel_samtools HN60 ) &> ./HN60.sam.out &
proc_ids+=("$!")

##Wait until process completion
for i in ${proc_ids[@]}; do
        wait $i
done
```

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT
BD1LYPACXX.lane_3_I12
chr1    10009   .       A       .       39.5871 .
DP=1;MQ0F=0;AN=2;DP4=1,0,0,0;MQ=10      GT      0/0
chr1    10010   .       C       .       39.5871 .
DP=1;MQ0F=0;AN=2;DP4=1,0,0,0;MQ=10      GT      0/0
chr1    10011   .       C       .       39.5871 .
DP=1;MQ0F=0;AN=2;DP4=1,0,0,0;MQ=10      GT      0/0
chr1    10012   .       C       .       39.5871 .
DP=1;MQ0F=0;AN=2;DP4=1,0,0,0;MQ=10      GT      0/0
```

The mode for variant calling is set to ***-m*** which denotes *multi-allelic calling* (Call variants with more than two alleles). The sample of the raw VCF file is provided above.

The called variants are then filtered using *vcfutils.pl* with a depth coverage of *10*, calculated using the formula:

$$Coverage_{depth} = \frac{(Read_{count} * Read_{avg.Length})}{Total.Number_{bases}}$$

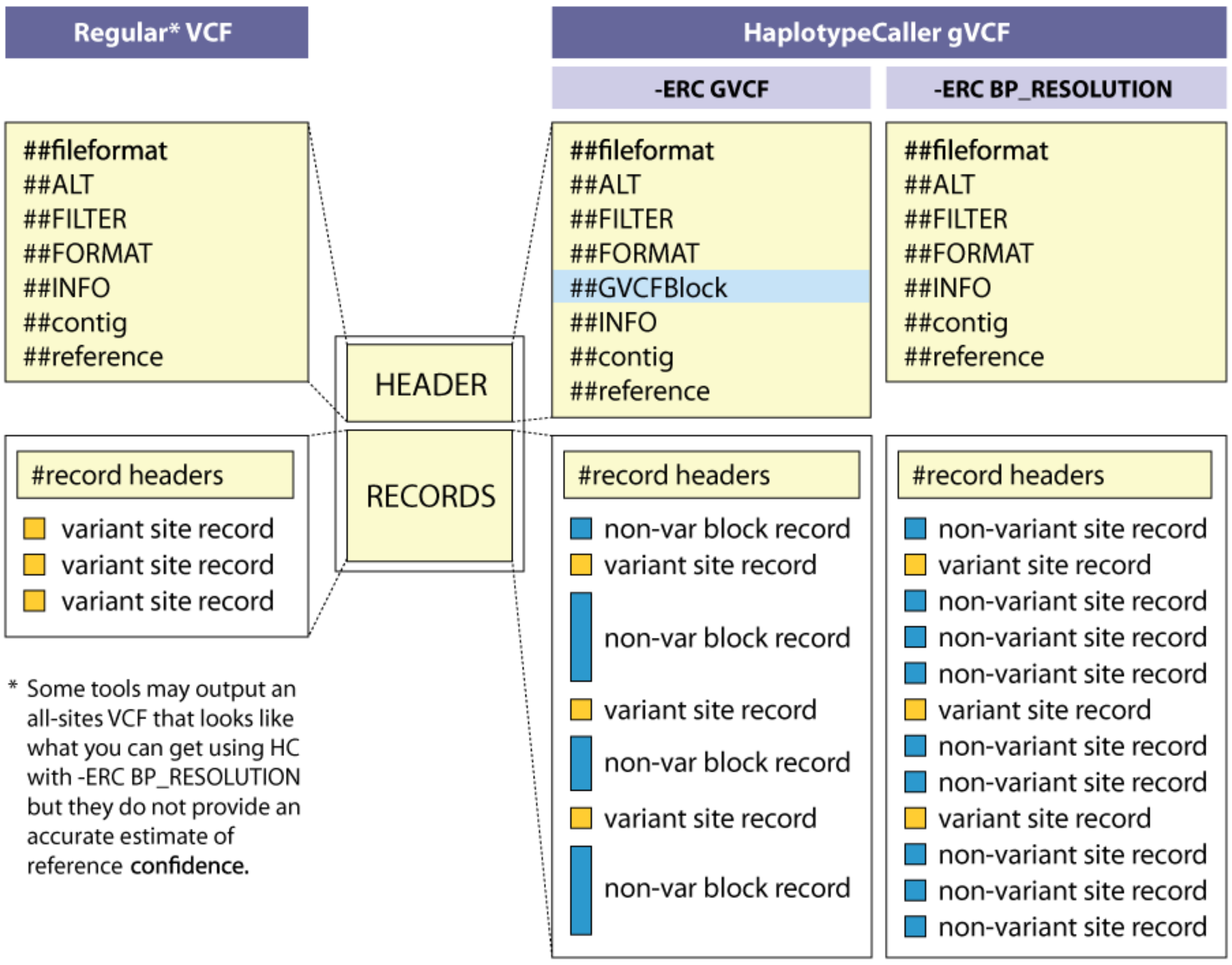
This depth value of 10 takes into account only 10x coverage.

Variant Calling using GATK

GATK is a toolkit which offers a wide variety of tools for variant discovery and genotyping. Its powerful processing engine and high-performance computing features make it capable of taking on projects of any size. It provides two modes of operation.

- Regular VCF
- GVCF (Genomic VCF)
    - GVCF
    - BasePair Resolution

Image below illustrates the comparison between VCF & GVCF. ***GVCF Mode*** is used as instructed.



```
#!/bin/bash
#$ -N GATK
#$ -q all.q
#$ -cwd
#$ -S /bin/bash
#$ -v PATH
#$ -v LD_LIBRARY_PATH

#module load jdk
#module load GATK or use custom GATK

##SUBSCRIPT
##ENTRYPOINT
##Runs gatk variant calling
##WAIT for bwa.sh output

proc_ids=()

##Runs gatk Haplotype Calling
##nohup gatk HaplotypeCaller --native-pair-hmm-threads 16 -R
/data4/pilib/genomes/hs/hg19/hg19.fa -I $1.sorted.rmdup.bam -O
##$1.gatk.vcf -ERC GVCF -L $1.flt.vcf &> $1.gatk.out&

nohup gatk HaplotypeCaller --native-pair-hmm-threads 16 -R
```

```
/data4/pilib/genomes/hs/hg19/hg19.fa -I $1.sorted.rmdup.bam -O "$1"_gatk.vcf -ERC GVCF
&> "$1"_gatk.out&
proc_ids+=("$!")
##nohup gatk HaplotypeCaller --native-pair-hmm-threads 16 -R
/data4/pilib/genomes/hs/hg19/hg19.fa -I $2.sorted.rmdup.bam -O
##$2.gatk.vcf -ERC GVCF -L $2.flt.vcf &> $2.gatk.out&
nohup gatk HaplotypeCaller --native-pair-hmm-threads 16 -R
/data4/pilib/genomes/hs/hg19/hg19.fa -I $2.sorted.rmdup.bam -O "$2"_gatk.vcf  -ERC GVCF
&> "$2"_gatk.out&
proc_ids+=("$!")

##Wait until process completion
for id in ${proc_ids[@]}; do
  wait $id
done

##MERGE VCF files
##CANT use DBImport needs intervals
##gatk GenomicsDBImport --reader-threads 16 -V "$1"_gatk.vcf   -V "$2"_gatk.vcf --
genomicsdb-workspace-path . -L $1.flt.vcf -L
##$2.flt.vcf
#gatk CombineGVCFs -R /data4/pilib/genomes/hs/hg19/hg19.fa -V
#"$1"_gatk.vcf   -V "$2"_gatk.vcf -O HN_merged.vcf
##gatk GenomicsDBImport --reader-threads 16 -V $1.gatk.vcf   -V $2.gatk.vcf --
genomicsdb-workspace-path . -L $1.flt.vcf -L $2.flt.vcf

##Genotype the called haplotypes for further processing

nohup gatk GenotypeGVCFs -R /data4/pilib/genomes/hs/hg19/hg19.fa -V "$1"_gatk.vcf -O
"$1"_gatk.g.vcf  &> geno_"$1"_gatk.out
nohup gatk GenotypeGVCFs -R /data4/pilib/genomes/hs/hg19/hg19.fa -V "$2"_gatk.vcf -O
"$2"_gatk.g.vcf  &> geno_"$2"_gatk.out

##Extract data from vcf files- Chromosome,Location,genotype
#bcftools query HN51.gatk.gz -f '%CHROM\t%POS[\t%GT]\n' > HN51_matrix.txt
##Extract sample names
#bcftools query -l HN51.gatk.gz > HN51_names.txt
```

HaplotypeCaller is a GATK utility which is used to call germline SNPs and indels via local re-assembly of haplotypes. The parameter for threading (*--native-pair-hmm-threads*) is set to 16 and mode(*-ERC*) is set to GVCF. The reference file is provided with *-R* and VCF files are passed to the *-V* parameter. Multiple VCF files can be passed as input by using multiple *-V* paramters.

The called haplotypes are then genotyped using *GenotypeGVCFs*. *GenotypeGVCFs* is used to perform joint genotyping on a single-sample GVCF from HaplotypeCaller or a multi-sample GVCF from CombineGVCFs or GenomicsDBImport. *The same parameters from HaplotypeCaller apply to GenotypeGVCFs.*

Final Steps

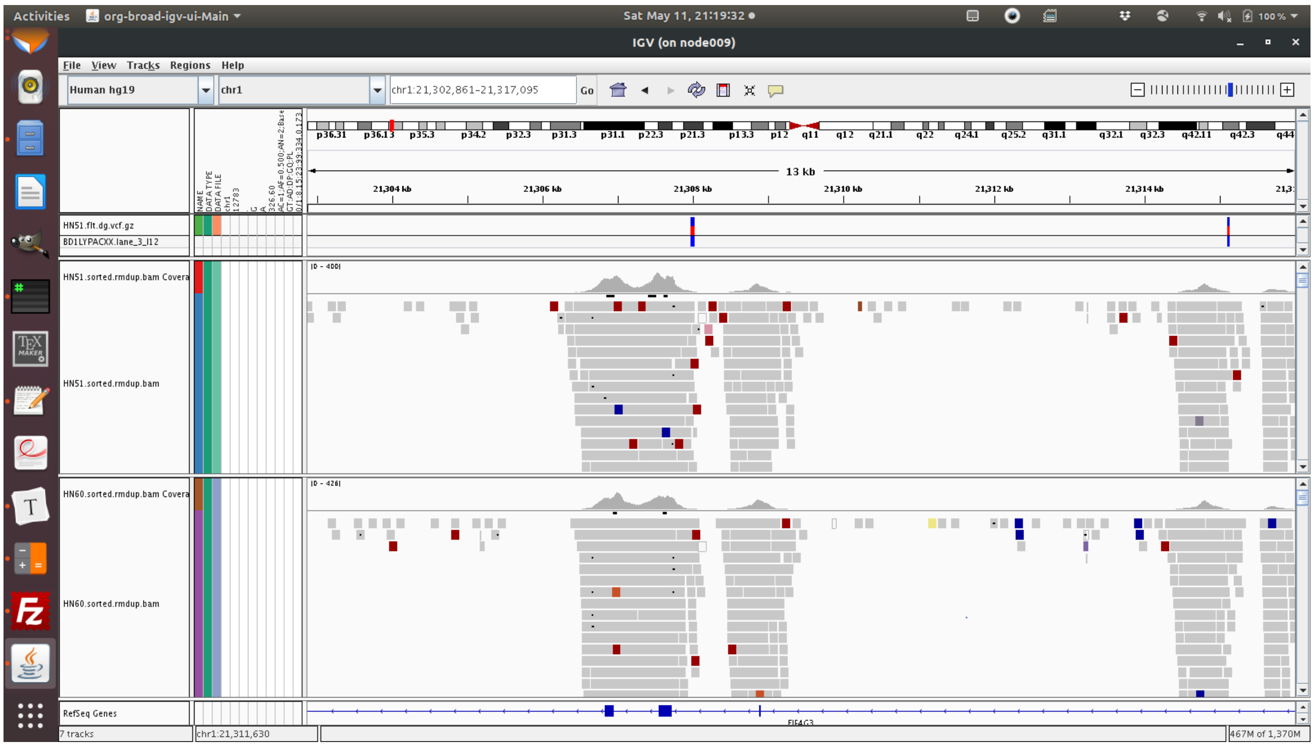
```

##COMPARISON-(Move to a different script)
##Compress outputs to save space, -c param prints the compressed output to STDOUT
```

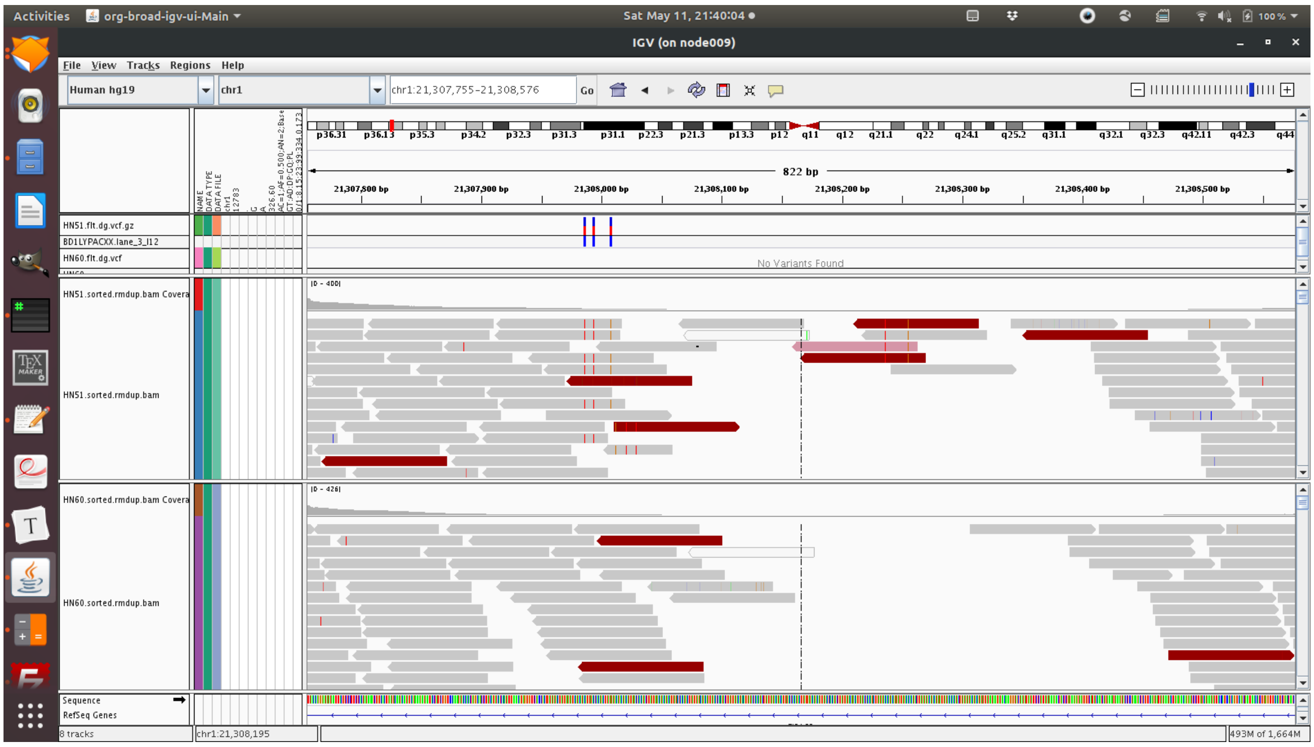
```
##Sample 1 from GATK & samtools
bgzip -c "$1".flt.vcf > "$1".sam.gz
bgzip -c "$1"_gatk.g.vcf > "$1".gatk.gz
##Sample 2 from GATK ad samtools
bgzip -c "$2".flt.vcf > "$2".sam.gz
bgzip -c "$2"_gatk.g.vcf > "$2".gatk.gz

##Index the SNP location regions in the format "chr:beginPos-endPos"
##using tabix. -p param is the file type
#Sample 1
tabix -p vcf "$1".sam.gz
tabix -p vcf "$1".gatk.gz
#Sample 2
tabix -p vcf "$2".sam.gz
tabix -p vcf "$2".gatk.gz

##Compare the sample files from different pipelines {GATK,samtools}
##Comparison provides indepth statistics such as SNP count, indel count etc etc
#Sample 1
bcftools stats "$1".sam.gz "$1".gatk.gz  > "$1".stats.txt
#Sample 2
bcftools stats "$2".sam.gz "$2".gatk.gz  > "$2".stats.txt

##Plot the stats using plot-vcfstats
#module load python
mkdir plots
cd plots
mkdir "$1"
mkdir "$2"

plot-vcfstats -p "$1"/ -r "$1".stats.txt
plot-vcfstats -p "$2"/ -r "$2".stats.txt

echo "Done!"
```

Comparison

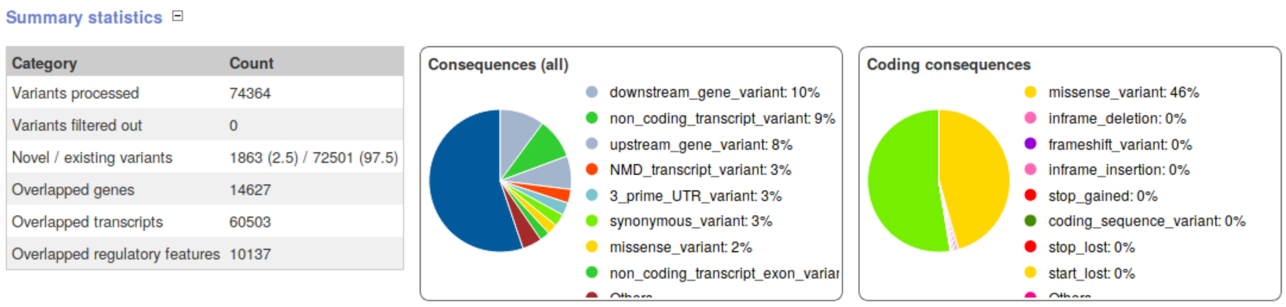
The genome (*FASTA*), aligned reads files of samples (*BAM*) and the variant files (*VCF*) are loaded into IGV for manual comparison. IGV shows a slight difference in peaks and variants exclusive to a single sample (*HN51*)

IGV





VEP Statistics



Summary statistics ⊟

Category	Count
Variants processed	74364
Variants filtered out	0
Novel / existing variants	1863 (2.5) / 72501 (97.5)
Overlapped genes	14627
Overlapped transcripts	60503
Overlapped regulatory features	10137

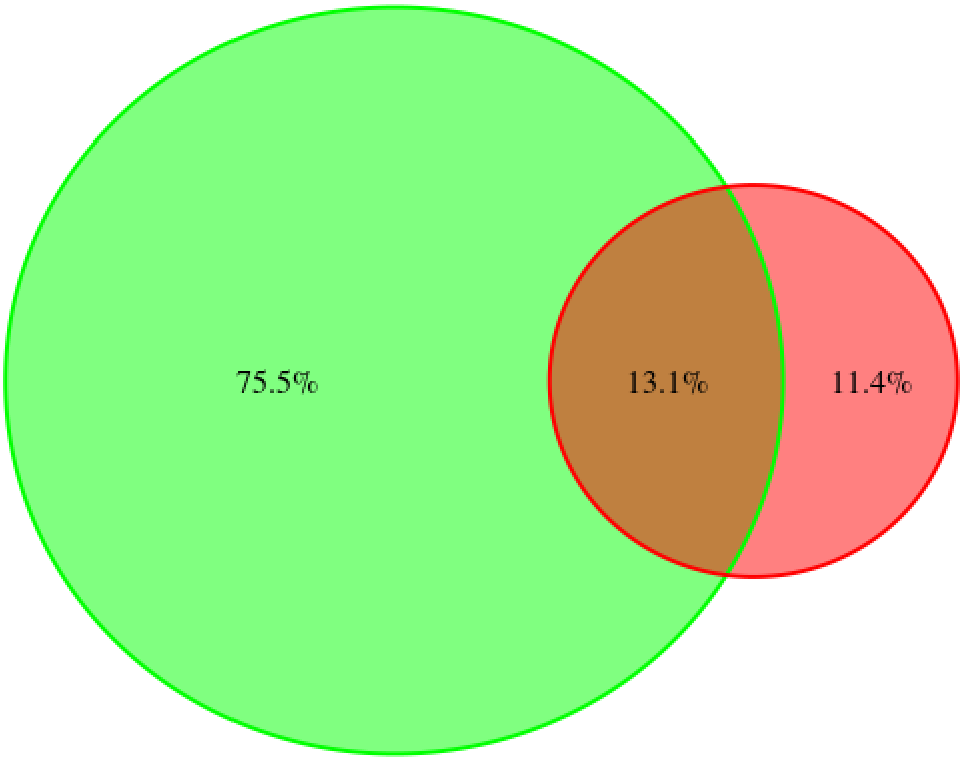
Consequences (all)

- ● downstream_gene_variant: 10%
- ● non_coding_transcript_variant: 9%
- ● upstream_gene_variant: 8%
- ● NMD_transcript_variant: 3%
- ● 3_prime_UTR_variant: 3%
- ● synonymous_variant: 3%
- ● missense_variant: 2%
- ● non_coding_transcript_exon_variar
- ● Others

Coding consequences

- ● missense_variant: 46%
- ● inframe_deletion: 0%
- ● frameshift_variant: 0%
- ● inframe_insertion: 0%
- ● stop_gained: 0%
- ● coding_sequence_variant: 0%
- ● stop_lost: 0%
- ● start_lost: 0%
- ● Others

Comparison in R



```
library(VennDiagram)
stats<-read.table("/home/viz/var_call/HN51.stats.txt",sep = "\t",header = TRUE)
stats[stats$id==1,]$id="GATK" #rename 1 with GATK
stats[stats$id==0,]$id="SAM"  #rename 0 as SAM
stats[stats$id==2,]$id="COMB" #rename 2 as Combined
stats<-stats[,-1]
snps.gatk<-subset(subset(stats, id=="GATK"), key == "number of multiallelic SNP
sites:")$value #we take a subset of stats data to isolate GATK and the corresponding
multiallelic values
snps.sam<-subset(subset(stats, id=="SAM"), key == "number of multiallelic SNP
sites:")$value
snps.comb<-subset(subset(stats, id=="COMB"), key == "number of multiallelic SNP
sites:")$value
VennDiagram::draw.pairwise.venn(snps.gatk+snps.comb,snps.sam+snps.comb,snps.comb,print.m
ode = "percent",col = c("red","green"),fill = c("red","green"))
```

Venn diagram shows that the *green area* (GATK) has more vairants found than the *red area* (BCFTOOLS) and 13.2% of the variants are common to both GATK and BCFTOOLS.

Conclusion

The above analysis shows that GATK is more efficient in detecting germline variants than BCFTOOLS pipeline. This is consistent across both the samples HN51 & HN60, thus GATK pipeline is more preferred than BCFTOOLS pipeline because of it's robustness and efficiency.

Note: GATK is supposed to be lenient but only when asked to be so (with -*LE* option)