

# Replicate GATK Variant calling analysis

Gialitsis Nikolaos DS2.190005

## 1. Brief Description of the analyzed data and overview

My task was to perform an analysis on 15 discrete Human samples (HG00234-HG00251) in order to identify variants, a process referred to as “Variant Calling” using the recommended protocol from the Broad Institute.

<https://gatk.broadinstitute.org/hc/en-us/articles/360035535932-Germline-short-variant-discovery-SNPs-Indels->

Only exome alignments were utilized, so in this case we are interested in variants in protein coding regions, which is practical since a drug discovery process usually transpires around targeting a single problematic protein, so identifying SNPs is an important step towards Personalized Medicine.

Moreover, the reads result from germline sequencing, so we are interested in inherited variants, in contrast to somatic variants which are local and dynamic in nature. One important thing to note, is that all germline cells in a single body carry the same genetic information whereas somatic cells are different depending on their location , for example cancer is the result from mutations on a subset of all cells.

The reads should be resulting from sequencing the chromosome 11. The chr11 contains about 135M bps and represents 4-4.5% of total DNA in cells. It also contains 1300-1400 protein coding genes. (<https://ghr.nlm.nih.gov/chromosome/11>) . However for the first steps of the downstream analysis, I downloaded the whole reference genome and not just the particular chromosome.

Also, the reads are paired-end, so this results in more accuracy. I run the following command to investigate if they are single-ended or pair-ended:

**samtools view -c -f 1 file.bam**

an output of ‘1’ means that the reads are paired-end (as in our case).

While discussing with my colleagues with a biological background, I expressed the question, whether a single-nucleotide variant is the same as a mutation. This caused some debate but we found the following chart explaining the differences:

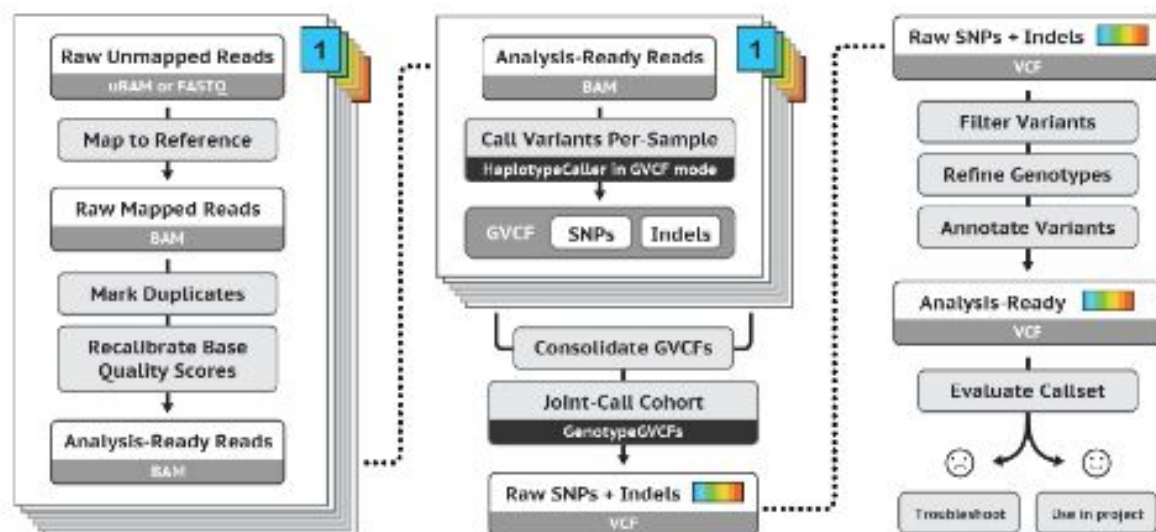
(<https://biodifferences.com/difference-between-mutation-and-variation.html>)

BASIS FOR COMPARISON	MUTATION	VARIATION
Meaning	Mutation is the natural and permanent change, causing changes in the DNA sequence in any living organisms.	Variation or genetic variation is seen in an individual of any species, groups or population and is observed in genes as well as in alleles. By the process of natural selections, mutations may bring evolutionary changes.
It affects	Mutations affect the single organisms.	Variations are seen in groups or populations of an individual.
Causing agent	Chemicals, ionising radiations, radioactive rays, chemical mutagens or x-rays.	Gene mutations, crossing over, genetic recombinations, genetic drift, gene flow, environmental factors.
Types	1.Germline or Hereditary or Fixed or Stable or Chromosomal mutations. 2.Somatic or Acquired or Dynamic or Unstable mutation.	1.Environmental variation. 2.Genetic variations. 3.Continuous variation. 4.Discontinuous variation.

So one important difference is that when talking about mutations, we refer to a change in a single individual whereas a mutation refers to genomic changes in groups of people, so the emphasis is on the population.

## 2. Protocol all analysis steps

The Broad Institute has developed the GATK pipeline which is summarized in the following steps:



## Preprocessing and mapping reads to reference

The input consists of the set of unmapped reads in raw form

Next, we map the reads to a reference genome, in our case the b37 human genome, also referred to as the hg19 reference

<https://gatk.broadinstitute.org/hc/en-us/articles/360035890711-GRCh37-hg19-b37-humanG1Kv37-Human-Reference-Discrepancies>

To index the reference genome and map the reads to it, I used the bwa aligner aka. the Burrows-Wheeler Aligner, which is recommended by the best practices protocol. bwa utilizes the Burrows-Wheeler Transform, a data compression method useful for string containing runs of characters such as in DNA:

Transformation				
1. Input	2. All rotations	3. Sort into lexical order	4. Take the last column	5. Output
^BANANA	^BANANA     ^BANANA A   ^BANAN NA   ^BANA ANA   ^BAN NANA   ^BA ANANA   ^B BANANA   ^	ANANA   ^B ANA   ^BAN A   ^BANAN BANANA   ^ NANA   ^BA NA   ^BANA ^BANANA     ^BANANA	ANANA   ^B ANA   ^BAN A   ^BANAN BANANA   ^ NANA   ^BA NA   ^BANA ^BANANA     ^BANANA	BNN^AA   A

The **bamToFastq** command was run in order to separate the paired-reads into fastq files and then map them to the reference. The fastq files report the mapper quality using quality scores and cigar strings.

The **bwa index** command was used to index the reference h19 genome and the **bwa mem** command was used to map the reads to sam.

Of course, before performing indexing, I sorted the bam files using **samtools sort**.

### **Marking Duplicates and Recalibration**

Duplicates (optical or library duplicates) affect the final variant call as the duplicated read can be interpreted as evidence towards a certain SNP, which is something that we have to avoid. The more you sequence a library the higher the number of duplicates. The number of optical duplications (nearby clusters on flow cell) is consistent because it is a preparation error.

In order to find duplicates, the algorithm does not examine the sequences themselves because sequences contain errors. The focus is on the positions of mapped reads (e.g where were the first reads mapped to) . So if two reads have their first mapped read in the same position, they are probably duplicates. A common practice is to identify duplicate sets and choose the most “representative”.

A duplicate status is indicated in SAM. Duplicates are usually not removed but tagged with a corresponding flag. Downstream analysis usually ignores the flagged reads and most GATK tools ignore those by default.

There are only a few cases where we might want to keep duplicates, for example in Amplicon Sequencing where all reads start at the same position by design. Another example would be when conducting allele-specific expression analysis.

The problem of marking duplicates is NP-hard

In order to mark the duplicated reads, I used the Picard tool, also developed by the Broad Institute and specifically I run the following command (all commands are in the log file submitted with this report)

```
java -Xmx2g -jar MarkDuplicates.jar INPUT=$file.sorted.mapped.bam  
OUTPUT=$file.dedup_reads.bam METRICS_file=$file.metrics.txt
```

A thing that I would like to mention is that I worked on the okeanos virtual machine, which had the java version 1.7 installed but the latest support for picard requires jdk 1.8. At first, I thought that updating the java version would be more troublesome than finding an older version of picard, I used the picard version 1.119 which support jdk 1.7. It turns out this was not a smart move since I was forced to update the java version in order to use gatk later on.

```
ubuntu@snf-872035:~/Desktop/picard-tools-1.119$ ls
AddCommentsToBam.jar          CollectTargetedPcrMetrics.jar  IlluminaBasecallsToFastq.jar  QualityScoreDistribution.jar
AddOrReplaceReadGroups.jar    CollectWgsMetrics.jar          IlluminaBasecallsToSam.jar     ReorderSam.jar
BamIndexStats.jar             CompareSAMs.jar                IntervalListTools.jar          ReplaceSamHeader.jar
BamToBfq.jar                  CreateSequenceDictionary.jar   libIntelDeflater.so           RevertOriginalBaseQualitiesA
BuildBamIndex.jar             DownsampleSam.jar              MakeSitesOnlyVcf.jar          RevertSam.jar
CalculateHsMetrics.jar         EstimateLibraryComplexity.jar  MarkDuplicates.jar             SamFormatConverter.jar
CheckIlluminaDirectory.jar    ExtractIlluminaBarcodes.jar    MarkIlluminaAdapters.jar       SamToFastq.jar
CleanSam.jar                   ExtractSequences.jar            MeanQualityByCycle.jar         SortSam.jar
CollectAlignmentSummaryMetrics.jar FastqToSam.jar                 MergeBamAlignment.jar          SplitVcfs.jar
CollectBaseDistributionByCycle.jar FifoBuffer.jar                 MergeSamFiles.jar              ValidateSamFile.jar
CollectGcBiasMetrics.jar       FilterSamReads.jar              MergeVcfs.jar                  VcfFormatConverter.jar
CollectInsertSizeMetrics.jar   FixMateInformation.jar          NormalizeFasta.jar             ViewSam.jar
CollectMultipleMetrics.jar     GatherBamFiles.jar              picard-1.119.jar
CollectRnaSeqMetrics.jar       htsjdk-1.119.jar               picard.jar
```

This process resulted in new files which (hopefully) do not contain any duplicates. I had to sort and index those bam files as before and deleted the previous “dirty” reads.

Next, I used the picard AddOrReplaceGroups command on the remaining files in order to fix any errors related to the read groups after the duplicate elimination.

Next we have to perform Base Quality Score Recalibration (BQSR) corrects the machine error after removing the duplicates.

for this step, I used **gatk BaseRecalibrator** with the following command:

```
gatk BaseRecalibrator \
-R b37.fasta \
-I $file.bam \
-L 11 \
--known-sites dbsnp_138.b37.vcf \
--known-sites Mills_and_1000G_gold_standard.indels.b37.vcf \
-O $file.recal_data.table
```

where I have used two datasets containing known variants . To do that, I run the **wget -c** and **gunzip -d** commands to store the data in .vcf format.

the -L 11 argument means that we are interested only in reads mapped to the chromosome 11.

In addition, the gatk recalibration step required a dictionary for the reference. This was done with the picard's CreateSequenceDictionary command which creates the b37.dict output file.

This procedure resulted in the following workspace:

```
gatk completion.sh
root@snf-872035:~/Desktop/workspace# ls | grep 'recal'
HG00234.recal_data.table
HG00234.recal_reads.bai
HG00234.recal_reads.bam
HG00235.recal_data.table
HG00235.recal_reads.bai
HG00235.recal_reads.bam
HG00236.recal_data.table
HG00236.recal_reads.bai
HG00236.recal_reads.bam
HG00237.recal_data.table
HG00237.recal_reads.bai
HG00237.recal_reads.bam
HG00238.recal_data.table
HG00238.recal_reads.bai
HG00238.recal_reads.bam
HG00239.recal_data.table
HG00239.recal_reads.bai
HG00239.recal_reads.bam
HG00240.recal_data.table
HG00240.recal_reads.bai
HG00240.recal_reads.bam
```

## Haplotype Caller

The haplotype caller is the main protagonist of the gatk pipeline. It utilizes pair-Hidden Markov Models on constructed Brujin-graphs in order to identify the haplotypes within the recalibrated read samples.

During the execution, the haplotype caller produces the following output:



```
root@snf-872035: ~/Desktop/workspace
21:34:04.328 INFO HaplotypeCaller - Done initializing engine
21:34:04.339 INFO HaplotypeCallerEngine - Tool is in reference confidence mode and the annotation, the following changes will be made to any specified annotations
randBiasBySample' will be enabled. 'ChromosomeCounts', 'FisherStrand', 'StrandOddsRatio' and 'QualByDepth' annotations have been disabled
21:34:04.384 INFO HaplotypeCallerEngine - Standard Emitting and Calling confidence set to 0.0 for reference-model confidence output
21:34:04.449 INFO NativeLibraryLoader - Loading libgkl_utils.so from jar:file:/home/ubuntu/Desktop/workspace/gatk-package-4.1.4.1-local.jar!/com/intel/gkl/native/
l_utils.so
21:34:04.471 INFO PairHMM - OpenMP multi-threaded AVX-accelerated native PairHMM implementation is not supported
21:34:04.471 WARN PairHMM - ***WARNING: Machine does not have the AVX instruction set support needed for the accelerated AVX PairHMM. Falling back to the MUCH slo
OGLESS CACHING implementation!
21:34:04.668 INFO ProgressMeter - Starting traversal
21:34:04.669 INFO ProgressMeter - Current Locus Elapsed Minutes Regions Processed Regions/Minute
21:34:14.676 INFO ProgressMeter - 1:2816515 0.2 9390 56317.5
21:34:27.574 INFO ProgressMeter - 1:6697669 0.4 22330 58498.9
21:34:37.580 INFO ProgressMeter - 1:10755815 0.5 35860 65380.3
21:34:47.583 INFO ProgressMeter - 1:14655465 0.7 48860 68315.0
21:34:57.587 INFO ProgressMeter - 1:18350069 0.9 61180 69370.3
21:35:07.594 INFO ProgressMeter - 1:22478069 1.0 74940 71457.6
21:35:17.594 INFO ProgressMeter - 1:26363069 1.2 87890 72313.6
21:35:27.599 INFO ProgressMeter - 1:30391767 1.4 101320 73306.1
21:35:37.638 INFO ProgressMeter - 1:34051492 1.5 113520 73263.9
21:35:47.687 INFO ProgressMeter - 1:38146492 1.7 127170 74067.4
21:35:57.747 INFO ProgressMeter - 1:42012356 1.9 140060 74318.2
21:36:07.752 INFO ProgressMeter - 1:45941569 2.1 153160 74663.0
21:36:17.759 INFO ProgressMeter - 1:50003127 2.2 166700 75153.3
21:36:27.758 INFO ProgressMeter - 1:53691559 2.4 179000 75058.7
21:36:37.935 INFO ProgressMeter - 1:57822352 2.6 192770 75465.9
21:36:47.935 INFO ProgressMeter - 1:61769593 2.7 205930 75679.4
21:36:57.936 INFO ProgressMeter - 1:65776336 2.9 219290 75937.6
21:37:08.002 INFO ProgressMeter - 1:69625023 3.1 232120 75967.1
21:37:18.590 INFO ProgressMeter - 1:73376921 3.2 244630 75690.0
21:37:28.590 INFO ProgressMeter - 1:77393921 3.4 258020 75918.0
21:37:38.592 INFO ProgressMeter - 1:81200720 3.6 270710 75928.0
21:37:48.592 INFO ProgressMeter - 1:84315905 3.7 281100 75321.2
21:37:59.999 INFO ProgressMeter - 1:87885905 3.9 293000 74703.9
21:38:10.003 INFO ProgressMeter - 1:91992493 4.1 306690 75006.1
21:38:20.006 INFO ProgressMeter - 1:95825642 4.3 318470 75070.5
21:38:30.011 INFO ProgressMeter - 1:99548348 4.4 331880 75046.1
21:38:40.016 INFO ProgressMeter - 1:103571348 4.6 345290 75241.3
21:38:50.021 INFO ProgressMeter - 1:107381115 4.8 357990 75273.6
21:39:00.457 INFO ProgressMeter - 1:110617930 4.9 368780 74806.5
21:39:10.463 INFO ProgressMeter - 1:115005954 5.1 383410 75229.3
21:39:20.467 INFO ProgressMeter - 1:118995195 5.3 396710 75373.3
21:39:30.476 INFO ProgressMeter - 1:123053768 5.4 410240 75549.5
```

A thing to note is that after reading the warning displayed above, I realized that there exists a compatibility with OpenMP which can take advantage of multiple CPUs through parallel programming to speed up the process. This would be much welcomed since this step takes a long time to complete; for only six samples it has been running for over 24 hours in the okeanos virtual machine.

The Haplotype Caller produces the desired Variant Call Format files (.vcf)

```
root@snf-872035:~/Desktop/workspace# ls | grep 'vcf'
dbsnp_138.b37.vcf
dbsnp_138.b37.vcf.gz
dbsnp_138.b37.vcf.idx
HG00234.variants.g.vcf
HG00234.variants.g.vcf.idx
HG00235.variants.g.vcf
HG00235.variants.g.vcf.idx
HG00236.variants.g.vcf
HG00236.variants.g.vcf.idx
HG00237.variants.g.vcf
HG00237.variants.g.vcf.idx
HG00238.variants.g.vcf
HG00238.variants.g.vcf.idx
HG00239.variants.g.vcf
HG00239.variants.g.vcf.idx
HG00240.variants.g.vcf
Mills_and_1000G_gold_standard.indels.b37.vcf
Mills_and_1000G_gold_standard.indels.b37.vcf.idx
```

a .vcf file looks like this

```
validation false --create-output-bam-index true --create-output-b
-output-sam-program-record true --add-output-vcf-command-line tru
-only-vcf-output false --help false --version false --showHidden
s 20 --gcs-project-for-requester-pays --disable-tool-default-rea
tations false --allow-old-rms-mapping-quality-annotation-data fal
##GVCFBBlock0-1=minGQ=0 (inclusive),maxGQ=1 (exclusive)
##GVCFBBlock1-2=minGQ=1 (inclusive),maxGQ=2 (exclusive)
##GVCFBBlock10-11=minGQ=10 (inclusive),maxGQ=11 (exclusive)
##GVCFBBlock11-12=minGQ=11 (inclusive),maxGQ=12 (exclusive)
##GVCFBBlock12-13=minGQ=12 (inclusive),maxGQ=13 (exclusive)
##GVCFBBlock13-14=minGQ=13 (inclusive),maxGQ=14 (exclusive)
##GVCFBBlock14-15=minGQ=14 (inclusive),maxGQ=15 (exclusive)
##GVCFBBlock15-16=minGQ=15 (inclusive),maxGQ=16 (exclusive)
##GVCFBBlock16-17=minGQ=16 (inclusive),maxGQ=17 (exclusive)
##GVCFBBlock17-18=minGQ=17 (inclusive),maxGQ=18 (exclusive)
##GVCFBBlock18-19=minGQ=18 (inclusive),maxGQ=19 (exclusive)
##GVCFBBlock19-20=minGQ=19 (inclusive),maxGQ=20 (exclusive)
##GVCFBBlock2-3=minGQ=2 (inclusive),maxGQ=3 (exclusive)
##GVCFBBlock20-21=minGQ=20 (inclusive),maxGQ=21 (exclusive)
##GVCFBBlock21-22=minGQ=21 (inclusive),maxGQ=22 (exclusive)
##GVCFBBlock22-23=minGQ=22 (inclusive),maxGQ=23 (exclusive)
##GVCFBBlock23-24=minGQ=23 (inclusive),maxGQ=24 (exclusive)
##GVCFBBlock24-25=minGQ=24 (inclusive),maxGQ=25 (exclusive)
##GVCFBBlock25-26=minGQ=25 (inclusive),maxGQ=26 (exclusive)
##GVCFBBlock26-27=minGQ=26 (inclusive),maxGQ=27 (exclusive)
##GVCFBBlock27-28=minGQ=27 (inclusive),maxGQ=28 (exclusive)
##GVCFBBlock28-29=minGQ=28 (inclusive),maxGQ=29 (exclusive)
##GVCFBBlock29-30=minGQ=29 (inclusive),maxGQ=30 (exclusive)
##GVCFBBlock3-4=minGQ=3 (inclusive),maxGQ=4 (exclusive)
##GVCFBBlock30-31=minGQ=30 (inclusive),maxGQ=31 (exclusive)
##GVCFBBlock31-32=minGQ=31 (inclusive),maxGQ=32 (exclusive)
##GVCFBBlock32-33=minGQ=32 (inclusive),maxGQ=33 (exclusive)
##GVCFBBlock33-34=minGQ=33 (inclusive),maxGQ=34 (exclusive)
##GVCFBBlock34-35=minGQ=34 (inclusive),maxGQ=35 (exclusive)
##GVCFBBlock35-36=minGQ=35 (inclusive),maxGQ=36 (exclusive)
##GVCFBBlock36-37=minGQ=36 (inclusive),maxGQ=37 (exclusive)
##GVCFBBlock37-38=minGQ=37 (inclusive),maxGQ=38 (exclusive)
##GVCFBBlock38-39=minGQ=38 (inclusive),maxGQ=39 (exclusive)
##GVCFBBlock39-40=minGQ=39 (inclusive),maxGQ=40 (exclusive)
##GVCFBBlock4-5=minGQ=4 (inclusive),maxGQ=5 (exclusive)
##GVCFBBlock40-41=minGQ=40 (inclusive),maxGQ=41 (exclusive)
##GVCFBBlock41-42=minGQ=41 (inclusive),maxGQ=42 (exclusive)
##GVCFBBlock42-43=minGQ=42 (inclusive),maxGQ=43 (exclusive)
:
```

## Refinement and Final notes

The next step according to the GATK best practices is to filter, refine and annotate the previously produced variants.

<https://gatkforums.broadinstitute.org/gatk/discussion/4723/genotype-refinement-workflow>

The following is a diagram depicting how this can be achieved in high-level.



```
graph TD; A[Recalibrated Variants] --> B[CalculateGenotypePosteriors]; B --> C[VariantFilteration lowGQ]; C --> D[VariantAnnotator PossibleDeNovos]; D --> E[Functional Annotation]; F[From HapMap/Omni] --> B; G[From *.ped file] --> B; H[Filter out GQ < 20] --> C; I[loConf: child GQ >= 10; AC < max(4, 0.1% samples)] --> D; J[hiConf: all GQs >= 20; AC < max(4, 0.1% samples)] --> D;
```

The flowchart illustrates the GATK best practices pipeline for variant calling. It begins with 'Recalibrated Variants', which leads to 'CalculateGenotypePosteriors'. This step receives input from 'From HapMap/Omni' and 'From \*.ped file'. The output of 'CalculateGenotypePosteriors' is 'Population Priors' and 'Family Priors', which then feed into 'VariantFilteration lowGQ'. 'Filter out GQ < 20' also feeds into 'VariantFilteration lowGQ'. The output of 'VariantFilteration lowGQ' is 'VariantAnnotator PossibleDeNovos'. 'VariantAnnotator PossibleDeNovos' receives input from 'loConf: child GQ >= 10; AC < max(4, 0.1% samples)' and 'hiConf: all GQs >= 20; AC < max(4, 0.1% samples)'. The final output of 'VariantAnnotator PossibleDeNovos' is 'Functional Annotation'.

In my case, if I were to proceed with the analysis, the following step would be to merge the produced .vcf for each sample and produce a single file containing all identified variants. Then I would try to refine the variants using a pipeline similar to that depicted in the previous picture.

(You can find all commands I run in the log file)