# Whole Genome Sequencing Analysis

Table of Contents

[Whole Genome Sequencing Analysis 1](#_Toc450744616)

[Introduction 1](#_Toc450744617)

[Index reference genome 2](#_Toc450744618)

[Quality Control metrics of short reads 2](#_Toc450744619)

[Trim short reads 3](#_Toc450744620)

[Data Cleanup 4](#_Toc450744621)

[Map to Reference 5](#_Toc450744622)

[Mark Duplicates & Sort (Picard) 5](#_Toc450744623)

[Indel Realignment 7](#_Toc450744624)

[Base Recalibration 9](#_Toc450744625)

[Variant Discovery Step 11](#_Toc450744626)

[Var. Calling (Haplotype Caller) 11](#_Toc450744627)

[Joint Genotyping 12](#_Toc450744628)

[Variant Recalibration 13](#_Toc450744629)

[Evaluation 15](#_Toc450744630)

[Variant Annotation 15](#_Toc450744631)

[IGV 19](#_Toc450744632)

[Autosomal recessive 20](#_Toc450744633)

[Other tools 21](#_Toc450744634)

## Introduction

 The advantages of Whole Genome Sequencing (WGS) allows us to call variants (SNPs/INDELs) as well as determine Copy Number Variation (CNV). WGS is presumably better than Whole Exome Sequencing (WES) due to every nucleotide of the DNA is sequenced, which allows us to identify these variants in the intronic/intergenic regions, whereas WES only sequences the exon regions.

 In this example, a Mom and Dad brought their sick child into the clinic. The doctors ordered that their genome be sequenced. We are going to examine a very small portion on chromosome 20 of this family’s genome.

 We will be using Broad’s Genome Analysis ToolKit ([GATK](https://www.broadinstitute.org/gatk/)) to call variants. Their [Best Practices Guide](https://www.broadinstitute.org/gatk/guide/best-practices.php) provide step-by-step recommendations for performing variant discovery analysis in high-throughput sequencing (HTS) data. GATK enables discovery of SNPs and small INDELs (typically < 50bp). GATK does not yet enable discovery of structural variants (SVs) or copy number variants (CNVs) (other tools can do this).



## Index reference genome

 Before we start any analysis, we first need to download the human reference genome and index it with the tools that will be aligning the short read sequences to it. I’ve already downloaded human (hg19) chromosome 20 from UCSC. *This indexing step only needs to be done once!*

 There are a lot of different DNA aligners out there. We will be using the aligner, [BWA](http://bio-bwa.sourceforge.net/), for indexing and aligning the short reads to the hg19 reference genome:

## Change directory to **genome**

cd genome/

## Index fasta file

bwa index ucsc.hg19.chr20.fa

 This command will create 5 files with these extensions (\*.amb, \*.ann, \*.bwt, \*.pac, \*.sa).

## Quality Control metrics of short reads

 One of the first things you need to do before doing any sort of alignments is to check the quality of the short reads you received from your Sequencing Facility. A great tool that runs a variety of quality metric checks is [FastQC](http://www.bioinformatics.babraham.ac.uk/projects/fastqc/).

## Changed directory to **wholeGenome**

cd ../wholeGenome/

## Make a directory called **fastqc**

mkdir fastqc

## FastQC

fastqc --noextract --nogroup -o fastqc \*.fastq.gz

 Here is a brief description of the options used:

|  |  |
| --- | --- |
| Option | Description |
| --noextract | Do not uncompress the output file after creating it. |
| --nogroup | Disable grouping of bases for reads > 50bp. |
| -o | Create all output files in the specified output directory. Please note this directory must exist as the program will not create it. |

 Let’s review the FastQC output from these 3 samples. Open the home folder on your Desktop and then open **wholeGenome** and then **fastqc**. Inside this folder you will see html and zip files for each sample (since this is paired-end data, we will see two sets of html and zip files for each sample: father, mother, son). Open the father html files. You should see something like this:



 Notice the left-pane side titled “Summary”. It contains all of the QC metrics that were measured on your sample. The right-pane side contains the graphics for each metric measured. At this point, you will want to look over any metrics where a yellow exclamation point or a red X were marked to see what the problem might be. There are a variety of tools that can help clean up the problem areas of your short reads (won’t have time to go over any of these).

## Trim short reads

 Trimming poor quality bases from your short reads is beneficial. Poor quality bases can cause misalignments or cause the short read to not align to the reference genome.

 We’ll be using the trimming tool [Trim Galore!](http://www.bioinformatics.babraham.ac.uk/projects/trim_galore/), which depends on another trimming tool, [Cutadapt](https://cutadapt.readthedocs.io/en/stable/). The advantages of Trim Galore! is that you can remove bases from either side of the short read, remove short reads that fall below a certain quality threshold, remove primer adapter contamination, as well as remove biased methylation positions, and much more.

## Trim Galore

## Father

trim\_galore \

--paired \

--fastqc\_args “--nogroup --noextract" \

father\_NA12877\_L3\_R1.fastq.gz \

father\_NA12877\_L3\_R2.fastq.gz

## Mother

trim\_galore \

--paired \

--fastqc\_args “--nogroup --noextract" \

mother\_NA12878\_L2\_R1.fastq.gz \

mother\_NA12878\_L2\_R2.fastq.gz

## Son

trim\_galore \

--paired \

--fastqc\_args “--nogroup --noextract" \

son\_NA12882\_L6\_R1.fastq.gz \

son\_NA12882\_L6\_R2.fastq.gz

mkdir -p trimmed/fastqc

mv \*\_fastqc.\* trimmed/fastqc/

mv \*trimming\_report.txt trimmed/

 Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| --paired | This option performs length trimming of quality/adapter/RRBS trimmed reads for paired-end files. |
| --fastqc\_args | Passes extra arguments to FastQC. |

Trim Galore! creates new fastq files that are the final product of the trimming options specified above. A trimming report file is created that gives a description of which primer adapter was found and removed. We set Trim Galore! to run FastQC on the trimmed files and so you will have html and zip files for each of the samples. It is wise to check the QC metrics on the trimmed fastq files to make sure everything looks okay or better than the raw data.

## Data Cleanup



 Now, you can start the GATK analysis pipeline. As you can see in the above diagram, the first couple of steps are non-GATK, meaning the tools aren’t supported by them.

### Map to Reference

The first step is to use BWA to align the trimmed, short reads to the human hg19 reference genome index that we created at the very beginning of this example.

## BWA mem

## Father

bwa mem -M \

-R ‘@RG\tID:ABC123.LANE3\tLB:LIB-father\tPL:ILLUMINA\tSM:father’ \

../genome/ucsc.hg19.chr20.fa \

father\_NA12877\_L3\_R1\_val\_1.fq.gz \

father\_Na12877\_L3\_R2\_val\_2.fq.gz > father\_aligned.sam

## Mother

bwa mem -M \

-R ‘@RG\tID:ABC123.LANE2\tLB:LIB-mother\tPL:ILLUMINA\tSM:mother’ \

../genome/ucsc.hg19.chr20.fa \

mother\_NA12878\_L2\_R1\_val\_1.fq.gz \

mother\_Na12878\_L2\_R2\_val\_2.fq.gz > mother\_aligned.sam

## Son

bwa mem -M \

-R ‘@RG\tID:ABC123.LANE6\tLB:LIB-son\tPL:ILLUMINA\tSM:son’ \

../genome/ucsc.hg19.chr20.fa \

son\_NA12882\_L6\_R1\_val\_1.fq.gz \

son\_Na12882\_L6\_R2\_val\_2.fq.gz > son\_aligned.sam

## View the first 10 lines of the SAM alignment file

head father\_aligned.sam

 Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| -M | Mark shorter split hits as secondary (for Picard compatibility) |
| -R *STR* | Complete read group head line. ‘\t’ can be used in *STR* and will be converted to a TAB in the output SAM. The read group ID will be attached to every read in the output. An example is ‘@RG\tID:foo\tSM:bar’. |

 BWA mem creates one output file, a SAM file. This SAM file contains all of the short reads from the trimmed fastq file. If a short read mapped to the reference genome, information about where, quality of the alignment, number of matches/mismatches, etc… will be represented in the SAM file.

### Mark Duplicates & Sort (Picard)

#### Sort

 The aligned reads in the SAM file must be sorted in a particular way for the downstream GATK tools to work properly. [Picard Tools](http://broadinstitute.github.io/picard/) has the necessary tools needed to sort the SAM file and convert it to a BAM file.

## Picard Tools SortSam

## Father

java \

-jar ../tools/picard-tools-2.2.1/picard.jar SortSam \

INPUT= father\_aligned.sam \

OUTPUT= father\_aligned\_sorted.bam \

SORT\_ORDER= coordinate

## Mother

java \

-jar ../tools/picard-tools-2.2.1/picard.jar SortSam \

INPUT= mother\_aligned.sam \

OUTPUT= mother\_aligned\_sorted.bam \

SORT\_ORDER= coordinate

## Son

java \

-jar ../tools/picard-tools-2.2.1/picard.jar SortSam \

INPUT= son\_aligned.sam \

OUTPUT= son\_aligned\_sorted.bam \

SORT\_ORDER= coordinate

 Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| INPUT | The BAM or SAM file to sort. Required. |
| OUTPUT | The sorted BAM or SAM output file. Required. |
| SORT\_ORDER | Sort order of output file. Required. Possible values: {unsorted, queryname, coordinate, duplicate} |

Picard Tools SortSam creates one output file, a sorted BAM file. This BAM file cannot be viewed (like when we viewed the previous SAM file). BAM files are binary compressed files (containing 0’s and 1’s), whereas the SAM file is considered like a text file. These BAM files are much smaller than SAM and they are very useful for viewing the aligned reads in a genomics viewer like [IGV](http://www.broadinstitute.org/software/igv/home), [UCSC Genomics Browser](http://genome.ucsc.edu/), [SeqMonk](http://www.bioinformatics.babraham.ac.uk/projects/seqmonk/), [IGB](http://bioviz.org/igb/index.html) and more.

#### Mark Duplicates

 It is important that we remove any type of duplicate short reads from our aligned data. We don’t want a rogue PCR overamplification read to give us a false positive on a possible variant. Picard Tools MarkDuplicates will be used to remove any duplicate found in the sorted, aligned BAM file.

## Picard Tools MarkDuplicates

## Father

java \

-jar ../tools/picard-tools-2.2.1/picard.jar MarkDuplicates \

INPUT= father\_aligned\_sorted.bam \

OUTPUT= father\_dedup.bam \

METRICS\_FILE= father\_dedup\_metrics.txt \

CREATE\_INDEX= TRUE \

ASSUME\_SORT\_ORDER= TRUE

## Mother

java \

-jar ../tools/picard-tools-2.2.1/picard.jar MarkDuplicates \

INPUT= mother\_aligned\_sorted.bam \

OUTPUT= mother\_dedup.bam \

METRICS\_FILE= mother\_dedup\_metrics.txt \

CREATE\_INDEX= TRUE \

ASSUME\_SORT\_ORDER= TRUE

## Son

java \

-jar ../tools/picard-tools-2.2.1/picard.jar MarkDuplicates \

INPUT= son\_aligned\_sorted.bam \

OUTPUT= son\_dedup.bam \

METRICS\_FILE= son\_dedup\_metrics.txt \

CREATE\_INDEX= TRUE \

ASSUME\_SORT\_ORDER= TRUE

 Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| INPUT | One or more input SAM or BAM files to analyze. Must be coordinate sorted. Required. |
| OUTPUT | The output file to write marked records to. Required. |
| METRICS\_FILE | File to write duplicate metrics to. Required. |
| CREATE\_INDEX | Whether to create a BAM index when writing a coordinate-sorted BAM file. |
| ASSUME\_SORT\_ORDER | If not null, assume that the input file has this order even if the header says otherwise. |

 Picard Tools MarkDuplicates will create two files: 1) a metrics file that gives stats on how many short reads were duplicates, and 2) an aligned, sorted, de-duplicated BAM file.

#### How many short reads aligned to the reference genome?

 This step isn’t in the GATK Best Practices Guide, but it is a good idea to find out how many of the trimmed short reads actually aligned to the reference genome. Picard Tools has a wealth of tools that look at various QC metrics of the alignment files, but I like a very simple tool from [SAMtools](http://www.htslib.org/), called flagstat.

## SAMTools flagstat

samtools flagstat father\_dedup.bam

samtools flagstat mother\_dedup.bam

samtools flagstat son\_dedup.bam

 The alignment stats are printed out onto the screen. Ideally, you would like to see a very high percentage of the short reads aligning to the reference genome. If you don’t see that, then it’s possible that your sample is contaminated with another species.

### Indel Realignment

 The algorithms that are used in the initial mapping step tend to produce various types of artifacts. For example, reads that align on the edges of indels often get mapped with mismatching bases that might look like evidence for SNPs, but are actually mapping artifacts. The realignment process identifies the most consistent placement of the reads relative to the indel in order to clean up these artifacts. This is a two-step process.

#### RealignerTargetCreator

## GATK RealignerTargetCreator

## Father

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T RealignerTargetCreator \

-R ../genome/ucsc.hg19.chr20.fa \

-I father\_dedup.bam \

-L chr20:10000000-10200000 \

-ped ceuTrio.ped \

-o father\_known.intervals

## Mother

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T RealignerTargetCreator \

-R ../genome/ucsc.hg19.chr20.fa \

-I mother\_dedup.bam \

-L chr20:10000000-10200000 \

-ped ceuTrio.ped \

-o mother\_known.intervals

## Son

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T RealignerTargetCreator \

-R ../genome/ucsc.hg19.chr20.fa \

-I son\_dedup.bam \

-L chr20:10000000-10200000 \

-ped ceuTrio.ped \

-o son\_known.intervals

\*\*\*NOTE: for whole genome/exome data, need ‘-known’ sites. This can be found in their GATK Resource Bundle.

Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| -T | Name of the tool to run |
| -R | Reference sequence file |
| -I | Input file containing sequence data (BAM or CRAM) |
| -L | One or more genomic intervals over which to operate |
| -ped | Input file containing pedigree information |
| -o | Output file |

 GATK RealignerTargetCreator will output a file that contains a list of target intervals to pass to the IndelRealigner tool.

#### IndelRealigner

## GATK IndelRealigner

## Father

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T IndelRealigner \

-R ../genome/ucsc.hg19.chr20.fa \

-I father\_dedup.bam \

-targetIntervals father\_known.intervals \

-ped ceuTrio.ped \

-o father\_realigned.bam

## Mother

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T IndelRealigner \

-R ../genome/ucsc.hg19.chr20.fa \

-I mother\_dedup.bam \

-targetIntervals mother\_known.intervals \

-ped ceuTrio.ped \

-o mother\_realigned.bam

## Son

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T IndelRealigner \

-R ../genome/ucsc.hg19.chr20.fa \

-I son\_dedup.bam \

-targetIntervals mother\_known.intervals \

-ped ceuTrio.ped \

-o mother\_realigned.bam

\*\*\*NOTE: for whole genome/exome data, need ‘-known’ sites. This can be found in their GATK Resource Bundle.

 Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| -T | Name of the tool to run |
| -R | Reference sequence file |
| -I | Input file containing sequence data (BAM or CRAM) |
| -targetIntervals | Infput file containing list of known indels |
| -ped | Input file containing pedigree information |
| -o | Output file |

 GATK IndelRealigner will output a file containing a realigned version of your input BAM file.

### Base Recalibration

Variant calling algorithms rely heavily on the quality scores assigned to the individual base calls in each sequence read. These scores are per-base estimates of error emitted by the sequencing machines. Unfortunately, the scores produced by the machines are subject to various sources of systematic technical error, leading to over- or under-estimated base quality scores in the data. Base quality score recalibration (BQSR) is a process in which GATK applies machine learning to model these errors empirically and adjust the quality scores accordingly. This allows us to get more accurate base qualities, which in turn improves the accuracy of our variant calls. This is a two-step process.

 However, since our example is a small portion of an entire genome, these two steps won’t work, so *we will skip these steps*. When working with a whole genome/exome, please follow the below steps to recalibrate the bases.

## GATK BaseRecalibrator

## Do all of these steps for each sample that you have!

## First pass

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T BaseRecalibrator \

-R ../genome/ucsc.hg19.chr20.fa \

-I father\_realigned.bam \

-L chr20:10000000-10200000 \

-ped ceuTrio.ped \

-knownSites:dbsnp \

-knownSites:mills \

-knownSties:1000g \

-o father\_recal\_initial\_data.grp

## Second pass

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T BaseRecalibrator \

-R ../genome/ucsc.hg19.chr20.fa \

-I father\_realigned.bam \

-L chr20:10000000-10200000 \

-ped ceuTrio.ped \

-knownSites:dbsnp \

-knownSites:mills \

-knownSties:1000g \

-BQSR father\_recal\_initial\_data.grp \

-o father\_recal\_final\_data.grp

## GATK AnalyzeCovariates

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T AnalyzeCovariates \

-R ../genome/ucsc.hg19.chr20.fa \

-L chr20:10000000-10200000 \

-ped ceuTrio.ped \

-before father\_recal\_initial\_data.grp \

-after father\_recal\_final\_data.grp \

-plots father\_recalibration\_plots.pdf \

-csv father\_recalibration\_plots.csv

## GATK PrintReads

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T PrintReads \

-R ../genome/ucsc.hg19.chr20.fa \

-I father\_realigned.bam \

-ped ceuTrio.ped \

-L chr20:10000000-10200000 \

-BQSR father\_recal\_initial\_data.grp \

-o father\_recal.bam

\*\*\*NOTE: for whole genome/exome data, need ‘-knownSites’. This can be found in their GATK Resource Bundle.

## Variant Discovery Step



 Now, we have a cleaned-up aligned BAM file that is ready for variant calling.

### Var. Calling (Haplotype Caller)

The HaplotypeCaller is capable of calling SNPs and indels simultaneously via local de-novo assembly of haplotypes in an active region. In other words, whenever the program encounters a region showing signs of variation, it discards the existing mapping information and completely reassembles the reads in that region. This allows the HaplotypeCaller to be more accurate when calling regions that are traditionally difficult to call, for example when they contain different types of variants close to each other. It also makes the HaplotypeCaller much better at calling indels than position-based callers like UnifiedGenotyper (one of GATKs other variant calling tools).

## GATK HaplotypeCaller

## Father

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T HaplotypeCaller \

-R ../genome/ucsc.hg19.chr20.fa \

-I father\_realigned.bam \

-L chr20:10000000-10200000 \

--genotyping\_mode DISCOVERY \

-stand\_emit\_conf 10 \

-stand\_call\_conf 30 \

-o father.hc.g.vcf \

-ped ceuTrio.ped \

--emitRefConfidence GVCF

## Mother

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T HaplotypeCaller \

-R ../genome/ucsc.hg19.chr20.fa \

-I mother\_realigned.bam \

-L chr20:10000000-10200000 \

--genotyping\_mode DISCOVERY \

-stand\_emit\_conf 10 \

-stand\_call\_conf 30 \

-o mother.hc.g.vcf \

-ped ceuTrio.ped \

--emitRefConfidence GVCF

## Son

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T HaplotypeCaller \

-R ../genome/ucsc.hg19.chr20.fa \

-I son\_realigned.bam \

-L chr20:10000000-10200000 \

--genotyping\_mode DISCOVERY \

-stand\_emit\_conf 10 \

-stand\_call\_conf 30 \

-o son.hc.g.vcf \

-ped ceuTrio.ped \

--emitRefConfidence GVCF

\*\*\*NOTE: For whole genome/exome data, use ‘-dbsnp’. This can be found in their GATK Resource Bundle.

 \*\*\*NOTE: For whole genome/exome data, use ‘\*\_recal.bam’ file instead of the ‘\*\_realigned.bam’ file.

Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| -T | Name of the tool to run |
| -R | Reference sequence file |
| -I | Input file containing sequence data (BAM or CRAM) |
| -L | One or more genomic intervals over which to operate |
| --genotyping\_mode | Specifies how to determine the alternate alleles to use for genotyping |
| -stand\_emit\_conf | The minimum phred-scaled confidence threshold at which variants should be emitted (and filtered with LowQual if less than the calling threshold) |
| -stand\_call\_conf | The minimum phred-scaled confidence threshold at which variants should be called |
| -o | Output file |
| -ped | Input file containing pedigree information |
| --emitRefConfidence | Mode for emitting reference confidence scores |

 GATK HaplotypeCaller will output either a VCF or gVCF file with raw, unfiltered SNP and INDEL calls. Regular VCFs must be filtered either by variant recalibration (best) or hard-filtering before use in downstream analysis. If using the reference-confidence model workflow for cohort analysis, the output is a gVCF file that must first be run through GenotypeGVCFs and then filtering before further analysis.

### Joint Genotyping

At this step, which applies only to the DNA workflow, we gather all the per-sample gVCFs and pass them all together to the joint genotyping tool, GenotypeGVCFs. This produces a set of joint-called SNP and INDEL calls ready for filtering. This cohort-wide analysis empowers sensitive detection of variants even at difficult sites, and produces a squared-off matrix of genotypes that provides information about all sites of interest in all samples considered, which is important for many downstream analyses.

## GATK GenotypeGVCFs

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T GenotypeGVCFs \

-R ../genome/ucsc.hg19.chr20.fa \

-ped ceuTrio.ped \

--variant son.hc.g.vcf \

--variant father.hc.g.vcf \

--variant mother.hc.g.vcf \

-o family.raw.snps.indels.vcf

\*\*\*NOTE: For whole genome/exome data, use ‘-dbsnp’. This can be found in their GATK Resource Bundle.

 Here’s a brief explanation of the options used:

|  |  |
| --- | --- |
| Option | Description |
| -T | Name of the tool to run |
| -R | Reference sequence file |
| -ped | Input file for pedigree information |
| --variant | One or more input gVCF files |
| -o | Output file |

 GATK GentoypeGVCFs will output a combined, genotyped VCF file of raw variants.

### Variant Recalibration

The GATK's variant calling tools are designed to be very lenient in order to achieve a high degree of sensitivity. This is good because it minimizes the chance of missing real variants, but it does mean that we need to filter the raw callset they produce in order to reduce the amount of false positives, which can be quite large.

The best way to filter the raw variant callset is to use variant quality score recalibration (VQSR), which uses machine learning to identify annotation profiles of variants that are likely to be real, and assigns a VQSLOD score to each variant that is much more reliable than the QUAL score calculated by the caller. In the first step of this two-step process, the program builds a model based on training variants, then applies that model to the data to assign a well-calibrated probability to each variant call. We can then use this variant quality score in the second step to filter the raw call set, thus producing a subset of calls with our desired level of quality, fine-tuned to balance specificity and sensitivity.

However, since our example is a small portion of an entire genome, these two steps won’t work, so *we will skip these steps*. When working with a whole genome/exome, please follow the below steps to recalibrate the variants. Note, you can [apply hard filters](https://www.broadinstitute.org/gatk/guide/article?id=2806) if you don’t have whole genome/exome data

#### Variant Recalibration & Apply Recalibration

## GATK VariantRecalibrator on **SNPs**

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T VariantRecalibrator \

-R ../genome/ucsc.hg19.chr20.fa \

-input family.raw.snps.indels.vcf \

-resource:hapmap \

-resource:omni \

-resource:1000G \

-resource:dbsnp \

-ped ceuTrio.ped \

-an DP -an QD -an FS -an SOR -an MQ -an MQRankSum -an ReadPosRankSum \

-mode SNP \

-tranche 100.0 -tranche 99.9 -tranche 99.0 -tranche 90.0 \

-recalFile family.snps.recal \

-tranchesFile family.snps.tranches \

-rscriptFile family.snps.recal.plots.R

## GATK ApplyRecalibration on **SNPs**

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T ApplyRecalibration \

-R ../genome/ucsc.hg19.chr20.fa \

-input family.raw.snps.indels.vcf \

-mode SNP \

--ts\_filter\_level 99.0 \

-recalFile family.snps.recal \

-tranchesFile family.snps.tranches \

-ped ceuTrio.ped \

-o family.snps.recalibrated.indels.raw.vcf

## GATK VariantRecalibrator on **INDELs**

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T VariantRecalibrator \

-R ../genome/ucsc.hg19.chr20.fa \

-input family.snps.recalibrated.indels.raw.vcf \

-resource:mills \

-resource:dbsnp \

-ped ceuTrio.ped \

-an DP -an QD -an FS -an SOR -an MQ -an MQRankSum -an ReadPosRankSum \

-mode INDEL \

-tranche 100.0 -tranche 99.9 -tranche 99.0 -tranche 90.0 \

--maxGaussians 4 \

-recalFile family.indels.recal \

-tranchesFile family.indels.tranches \

-rscriptFile family.indels.recal.plots.R

## GATK ApplyRecalibration on **INDELs**

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T ApplyRecalibration \

-R ../genome/ucsc.hg19.chr20.fa \

-input family.snps.recalibrated.indels.raw.vcf \

-mode INDEL \

--ts\_filter\_level 99.0 \

-recalFile family.indels.recal \

-tranchesFile family.indels.tranches \

-ped ceuTrio.ped \

-o family.snps.indels.recalibrated.vcf

\*\*\*NOTE: For whole genome/exome data, use various resources found in their GATK Resource Bundle.

 This multistep process will output a recalibrated SNP/INDEL file for further downstream analysis.

## Evaluation



 There are several options now to evaluate and refine the variant and genotype calls further. Phasing and genotype refinement can be performed on whole genome data. If you’re interested in performing these steps, refer to [GATK Best Practice Guide](https://www.broadinstitute.org/gatk/guide/best-practices).

### Variant Annotation

 Now that we have a recalibrated variant file, it’s time to identify the variants that may be causing the disease in the child. We are going to filter the recalibrated variant file on Quality score and various Inheritance modes (ie autosomal recessive, de novo, X-linked).

 Here we will use several tools to identify the variants based on the inheritance model. First, we’ll use GATK SelectVariants to select each individual out of the recalibrated variant file and place filters like Homozygous/Heterozygous and Quality > 100. Then, we will need to compress the filtered variant files with bgzip and index them with tabix. [VCFTools](https://vcftools.github.io/index.html) will then be used on the compressed variant files to identify common/unique variants between the family members.

#### Autosomal Recessive

## Autosomal Recessive Variants

# GATK SelectVariants

# Son

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T SelectVariants \

-R ../genome/ucsc.hg19.chr20.fa \

--variant family.raw.snps.indels.vcf \

-sn son \

-o son\_analysisRead\_Qual100.HomVar.vcf \

-select “QUAL > 100.0 && (vc.getGenotype(‘son’).isHomVar())”

# Father

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T SelectVariants \

-R ../genome/ucsc.hg19.chr20.fa \

--variant family.raw.snps.indels.vcf \

-sn father \

-o father\_analysisRead\_Qual100.Het.vcf \

-select “QUAL > 100.0 && (vc.getGenotype(‘father’).isHet())”

# Mother

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T SelectVariants \

-R ../genome/ucsc.hg19.chr20.fa \

--variant family.raw.snps.indels.vcf \

-sn mother \

-o mother\_analysisRead\_Qual100.Het.vcf \

-select “QUAL > 100.0 && (vc.getGenotype(‘mother’).isHet())”

# bgzip

bgzip son\_analysisReady\_Qual100.HomVar.vcf

bgzip father\_analysisReady\_Qual100.Het.vcf

bgzip mother\_analysisReady\_Qual100.Het.vcf

# tabix

tabix -f -p vcf son\_analysisReady\_Qual100.HomVar.vcf.gz

tabix -f -p vcf father\_analysisReady\_Qual100.Het.vcf.gz

tabix -f -p vcf mother\_analysisReady\_Qual100.Het.vcf.gz

# VCFTools

vcf-isec -n =3 -f \

son\_analysisReady\_Qual100.HomVar.vcf.gz \

father\_analysisReady\_Qual100.Het.vcf.gz \

mother\_analysisReady\_Qual100.Het.vcf.gz \

| bgzip -c > son.father.mother.common.HomVar.Het.vcf.gz

# tabix

tabix -f -p vcf son.father.mother.common.HomVar.Het.vcf.gz

#### de novo

## de novo Variants

# GATK SelectVariants

# Son

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T SelectVariants \

-R ../genome/ucsc.hg19.chr20.fa \

--variant family.raw.snps.indels.vcf \

-sn son \

-o son\_analysisReady\_Qual100.Het.vcf \

-select “QUAL > 100.0 && (vc.getGenotype(‘son’).isHet())”

# Father

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T SelectVariants \

-R ../genome/ucsc.hg19.chr20.fa \

--variant family.raw.snps.indels.vcf \

-sn father \

-o father\_analysisReady\_Qual100.Het.HomVar.vcf \

-select “QUAL > 100.0 && (vc.getGenotype(‘father’).isHet() || vc.getGenotype(‘father’).isHomVar())”

# Mother

java \

-jar ../tools/GenomeAnalysisTK-3.5/GenomeAnalysisTK.jar \

-T SelectVariants \

-R ../genome/ucsc.hg19.chr20.fa \

--variant family.raw.snps.indels.vcf \

-sn mother \

-o mother\_analysisReady\_Qual100.Het.HomVar.vcf \

-select “QUAL > 100.0 && (vc.getGenotype(‘mother’).isHet() || vc.getGenotype(‘mother’).isHomVar())”

# bgzip

bgzip son\_analysisReady\_Qual100.Het.vcf

bgzip father\_analysisReady\_Qual100.Het.HomVar.vcf

bgzip mother\_analysisReady\_Qual100.Het.HomVar.vcf

# tabix

tabix -f -p vcf son\_analysisReady\_Qual100.Het.vcf.gz

tabix -f -p vcf father\_analysisReady\_Qual100.Het.HomVar.vcf.gz

tabix -f -p vcf mother\_analysisReady\_Qual100.Het.HomVar.vcf.gz

# VCFTools

vcf-isec -c -f \

son\_analysisReady\_Qual100.Het.vcf.gz \

father\_analysisReady\_Qual100.Het.HomVar.vcf.gz \

mother\_analysisReady\_Qual100.Het.HomVar.vcf.gz \

| bgzip -c > son.Het\_uniqueFrom\_father.mother.Het.HomVar.vcf.gz

tabix

tabix -f -p vcf son.Het\_uniqueFrom\_father.mother.Het.HomVar.vcf.gz

#### X-linked

 Note that since there are only short reads that aligned to chromosome 20 in this example dataset, that this step is here for reference when you have whole genome/exome data.

## X-linked

# VCFTools

vcf-isec -n =2 -f \

son\_analysisReady\_Qual100.HomVar.vcf.gz \

mother\_analysisReady\_Qual100.Het.vcf.gz \

| bgzip -c > son.HomVar\_commonWith\_mother.Het.Xlinked.vcf.gz

# tabix

tabix -f -p vcf son.HomVar\_commonWith\_mother.Het.Xlinked.vcf.gz

#### Effect on Function

 These variant VCF files do not contain information about which gene the variant is in, what type of change occurred (non-synonymous, stop gained, stop loss, start gained, etc…), functional impact scores, conservation scores, and so forth. [SNPEff](http://snpeff.sourceforge.net/) will add these various effect on functions to the variant file and allow you to output it into a text file for easier viewing.

 SNPEff needs at least 4GB RAM to run properly and the VM I designed contains 2GB because I wasn’t sure of how much RAM your laptops had. If you have more than 4GB on your laptop, you can go back to the VirtualBox settings and modify the RAM for HCGS to be greater than 4GB.

## Autosomal Recessive

# SNPEff

java -Xmx4g \

-jar ../tools/snpEff/snpEff.jar \

-v \

-s son.father.mother.common.HomVar.Het\_snpeff\_summary.html \

GRCh37.75 \

son.father.mother.common.HomVar.Het.vcf.gz \

> son.father.mother.common.HomVar.Het\_snpeff\_report.vcf

# SNPSift

cat son.father.mother.common.HomVar.Het\_snpeff\_report.vcf | \

perl ../tools/snpEff/scripts/vcfEffOnePerLine.pl | \

java -Xmx4g \

-jar ../tools/snpEff/SnpSift.jar extractFields \

- CHROM POS ID REF ALT QUAL FILTER AF AC DP MQ “ANN[\*].ALLELE” “ANN[\*].EFFECT” “ANN[\*].IMPACT” “ANN[\*].GENE” “ANN[\*].GENEID” “ANN[\*].FEATURE” “ANN[\*].FEATUREID” “ANN[\*].BIOTYPE” “ANN[\*].RANK” “ANN[\*].HGVS\_C” “ANN[\*].HGVS\_P” “ANN[\*].CDNA\_POS” “ANN[\*].CDNA\_LEN” “ANN[\*].CDS\_POS” “ANN[\*].CDS\_LEN” “ANN[\*].AA\_POS” “ANN[\*].AA\_LEN” “ANN[\*].DISTANCE” “ANN[\*].ERRORS” “LOF[\*].GENE” “LOF[\*].GENEID” “LOF[\*].NUMTR” “LOF[\*].PERC” “NMD[\*].GENE” “NMD[\*].GENEID” “NMD[\*].NUMTR” “NMD[\*].PERC” \

> son.father.mother.common.HomVar.Het\_snpeff\_report.txt

## Autosomal Dominant de novo

# SNPEff

java -Xmx4g \

-jar ../tools/snpEff/snpEff.jar \

-v \

-s son.Het\_uniqueFrom\_father.mother.Het.HomVar\_snpeff\_summary.html \

GRCh37.75 \

son.father.mother.common.HomVar.Het.vcf.gz \

> son.Het\_uniqueFrom\_father.mother.Het.HomVar\_snpeff\_report.vcf

# SNPSift

cat son.Het\_uniqueFrom\_father.mother.Het.HomVar\_snpeff\_report.vcf | \

perl ../tools/snpEff/scripts/vcfEffOnePerLine.pl | \

java -Xmx4g \

-jar ../tools/snpEff/SnpSift.jar extractFields \

- CHROM POS ID REF ALT QUAL FILTER AF AC DP MQ “ANN[\*].ALLELE” “ANN[\*].EFFECT” “ANN[\*].IMPACT” “ANN[\*].GENE” “ANN[\*].GENEID” “ANN[\*].FEATURE” “ANN[\*].FEATUREID” “ANN[\*].BIOTYPE” “ANN[\*].RANK” “ANN[\*].HGVS\_C” “ANN[\*].HGVS\_P” “ANN[\*].CDNA\_POS” “ANN[\*].CDNA\_LEN” “ANN[\*].CDS\_POS” “ANN[\*].CDS\_LEN” “ANN[\*].AA\_POS” “ANN[\*].AA\_LEN” “ANN[\*].DISTANCE” “ANN[\*].ERRORS” “LOF[\*].GENE” “LOF[\*].GENEID” “LOF[\*].NUMTR” “LOF[\*].PERC” “NMD[\*].GENE” “NMD[\*].GENEID” “NMD[\*].NUMTR” “NMD[\*].PERC” \

> son.Het\_uniqueFrom\_father.mother.Het.HomVar\_snpeff\_report.txt

## X-linked

# SNPEff

java -Xmx4g \

-jar ../tools/snpEff/snpEff.jar \

-v \

-s son.HomVar\_commonWith\_mother.Het.Xlinked\_snpeff\_summary.html \

GRCh37.75 \

son.HomVar\_commonWith\_mother.Het.Xlinked.vcf.gz \

> son.HomVar\_commonWith\_mother.Het.Xlinked\_snpeff\_report.vcf

# SNPSift

cat son.HomVar\_commonWith\_mother.Het.Xlinked\_snpeff\_report.vcf | \

perl ../tools/snpEff/scripts/vcfEffOnePerLine.pl | \

java -Xmx4g \

-jar ../tools/snpEff/SnpSift.jar extractFields \

- CHROM POS ID REF ALT QUAL FILTER AF AC DP MQ “ANN[\*].ALLELE” “ANN[\*].EFFECT” “ANN[\*].IMPACT” “ANN[\*].GENE” “ANN[\*].GENEID” “ANN[\*].FEATURE” “ANN[\*].FEATUREID” “ANN[\*].BIOTYPE” “ANN[\*].RANK” “ANN[\*].HGVS\_C” “ANN[\*].HGVS\_P” “ANN[\*].CDNA\_POS” “ANN[\*].CDNA\_LEN” “ANN[\*].CDS\_POS” “ANN[\*].CDS\_LEN” “ANN[\*].AA\_POS” “ANN[\*].AA\_LEN” “ANN[\*].DISTANCE” “ANN[\*].ERRORS” “LOF[\*].GENE” “LOF[\*].GENEID” “LOF[\*].NUMTR” “LOF[\*].PERC” “NMD[\*].GENE” “NMD[\*].GENEID” “NMD[\*].NUMTR” “NMD[\*].PERC” \

> son.HomVar\_commonWith\_mother.Het.Xlinked\_snpeff\_report.txt

#### CADD

 In case your laptop doesn’t have 4GB RAM, then here’s an alternative to this example dataset. Take the output VCF files from VCFTools and load them into a Web-based annotation tool, [CADD](http://cadd.gs.washington.edu/), under “Score variants”. Note that if you want to use whole genome/exome data, that this Web-based tool won’t accept more than 100,000 variants (~2MB file). After selecting your file and placing a check in the checkbox to include annotation in the output, click “Upload variants”. CADD will annotate your variant list and provide you a file to download for further review.

## IGV

 Now, let’s view some of the above variants in a Genomics Viewer/Browser. To do this, we will use Broad’s IGV. Opening IGV is a little bit awkward with the VM I built. Here’s how to do it:

## Change directory to **IGV** (in **tools** directory)

cd tools/IGV\_2.3.72/

## Now to run IGV

./igv.sh

## Do not close the Terminal window while running IGV!

 The above command will open IGV. We want to load in the recalibrated alignment files. In our example they are the \*\_realigned.bam files. For a whole genome/exome analyses, they are the \*\_recal.bam files.

 With IGV open, click “File” and then select “Load from File…”. Find the **wholeGenome** directory and load in father\_realigned.bam. Now, go back and load in the other two files: mother\_realigned.bam and son\_realigned.bam.

 Now, let’s look at some of the variants identified from the CADD output files.

### Autosomal recessive

 After unzipping the CADD output file, open the \*.tsv file in LibreOffice Calc (aka Excel) and place filters on the column headers. The very last column is titled “PHRED” and select a filter there with the values are >= 10. This will reduce the list down to 4 autosomal recessive variants.

 Let’s look at the first one at chr20:10006291. In the IGV window select “chr20” from the chromosome dropdown box and then in the window to the right, type “chr20:10006291”. After hitting Enter/Return, you should see something like this:



 Let’s look at the second one at chr20:10081750:



## Other tools

 There are other tools out there besides the ones I mentioned above. As I mentioned earlier, these tools we used are what a majority of groups use to call variants (ie BWA, Picard Tools and GATK).

 Here is a brief list of some of the other whole genome/exome tools out there:

|  |  |  |
| --- | --- | --- |
| Aligners | Variant Callers | Effect on Function |
| [Bowtie2](http://bowtie-bio.sourceforge.net/bowtie2/index.shtml) | [Freebayes](https://github.com/ekg/freebayes) | [Gemini](http://gemini.readthedocs.io/en/latest/) |
|  | [SAM Tools](http://www.htslib.org/) | [dbNSFP](https://sites.google.com/site/jpopgen/dbNSFP) (used with SNPEff) |
|  |  | [ANNOVAR](http://annovar.openbioinformatics.org/en/latest/) |
|  |  | [CADD](http://cadd.gs.washington.edu/) |
|  |  | [Ingenuity Variant Analysis](http://www.ingenuity.com/products/variant-analysis) ($$) |
|  |  | [Cartagenia Bench Lab NGS](https://cartagenia.com/cartagenia-bench-lab) ($$) |