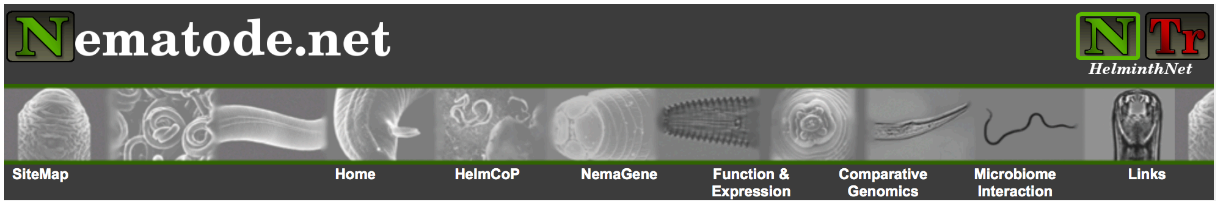
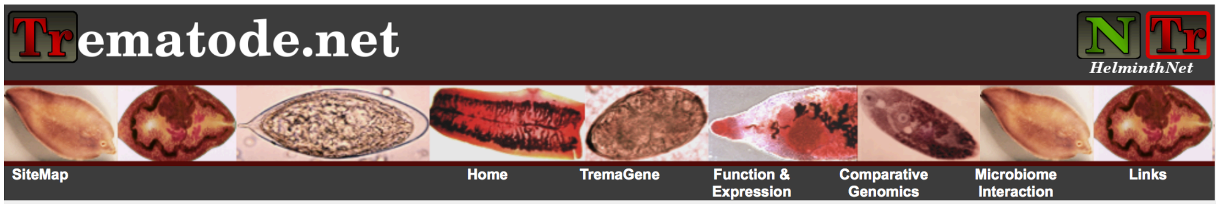
Washington University in St. Louis
SCHOOL OF MEDICINE

MCDONNELL GENOME INSTITUTE





Bioinformatics Workshop for Helminth Genomics

Section 3: Variome

Sponsors:

The Genome Institute Mitreva lab · Burroughs Wellcome Fund · New England BioLabs Inc. · amazon web services™

Table of contents – Curriculum

Section 3: Variome
Module 0: Re-sequencing genomes

Analysis of genetic variation is central to understanding population biology and molecular epidemiology of helminth parasites. Studying genome variations within and between populations can provide insights into geographical differentiation and gene flow, transmission patterns and evolution of parasites. In addition, genome-wide association studies (GWAS) and forward genetic screens (mapping-by-sequencing) can greatly facilitate identification of genetic variants correlated with phenotypes of biomedical interests (e.g., infection behavior, drug resistance, etc.)

NGS provides an unprecedented opportunity to characterize genetic variation in large number of samples at a reasonable cost. Sequencing individuals at a high coverage is the 'gold standard' for obtaining high-quality data, but budget constraints may require alternatives for studying large populations. Reduced representation and pooled sequencing approaches can be cost-efficient, but it is important to understand the strengths and weaknesses of each method to strategically design your experiment.

The following modules in this section will help you understand how we can turn raw sequencing data into reliable information about genetic variation.

Recommended reading:

DePristo, M. A., E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. Gabriel, D. Altshuler and M. J. Daly (2011). "A framework for variation discovery and genotyping using next-generation DNA sequencing data." Nat Genet **43**(5): 491-498.

Nielsen, R., J. S. Paul, A. Albrechtsen and Y. S. Song (2011). "Genotype and SNP calling from next-generation sequencing data." Nat Rev Genet **12**(6): 443-451.

Schlotterer, C., R. Tobler, R. Kofler and V. Nolte (2014). "Sequencing pools of individuals - mining genome-wide polymorphism data without big funding." Nat Rev Genet **15**(11): 749-763.

Variome – introduction (cont'd)

- Not all mismatches are SNPs!

  Errors in library preparation/basecalling/mapping etc.

- The basic idea behind finding probability of bases at a locus (genotype likelihoods) using Bayes theorem

$$P(A|B) = k \text{ X } \underbrace{P(B|A)}_{\text{Error model}} \text{ X } P(A)$$

genotype

data

Prior on genotype
(e.g. P(G)=0.3 if GC content is 60%)
(or P(non-ref)=1e-4, if SNP rate is known to be 0.01%)
(or… any other "prior" constraint you know about)

Some SNP calling programs

	published	citations
CRISP	2010	92
SNVer	2011	86
Samtools	2011	176
GATK (Genome Analysis Tool Kit)	2011	>2000
SomaticSniper	2012	128
Varscan-2	2012	404

Genome Analysis Tool Kit

Developed at The Broad Institute, Cambridge, MA

Installation: download directly from GATK website

Java Usage: a single jar file (except some preprocessing steps, which use bwa and picard tools)

Help for anything related to GATK, available at GATK website (with Guide, tools documentation and best practices)

Specifically, it is highly recommended to read the best practices before (or while) using GATK:
https://www.broadinstitute.org/gatk/guide/best-practices

The use forums (http://gatkforums.broadinstitute.org/) are also great, with usually very prompt responses by the GATK team

Before we start…

All figures in Module 1 and 2 are courtesy GATK online material (used here with permission)

Our dataset : 4 samples from male *D. viviparus* worms

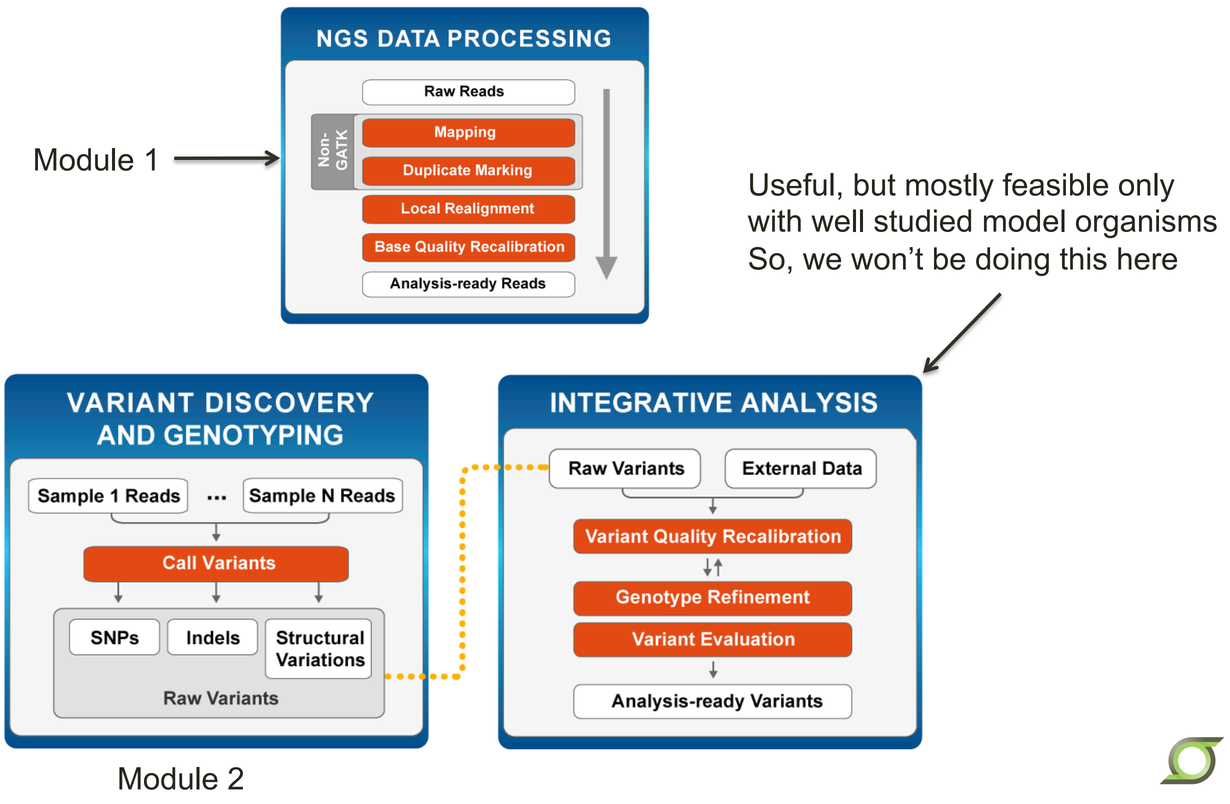
We selected just 2 contigs for illustration (You will usually do this on the whole genomes of your worm of interest, so your SNP calling will take more than the 2 hours we have here!)

Starting data :

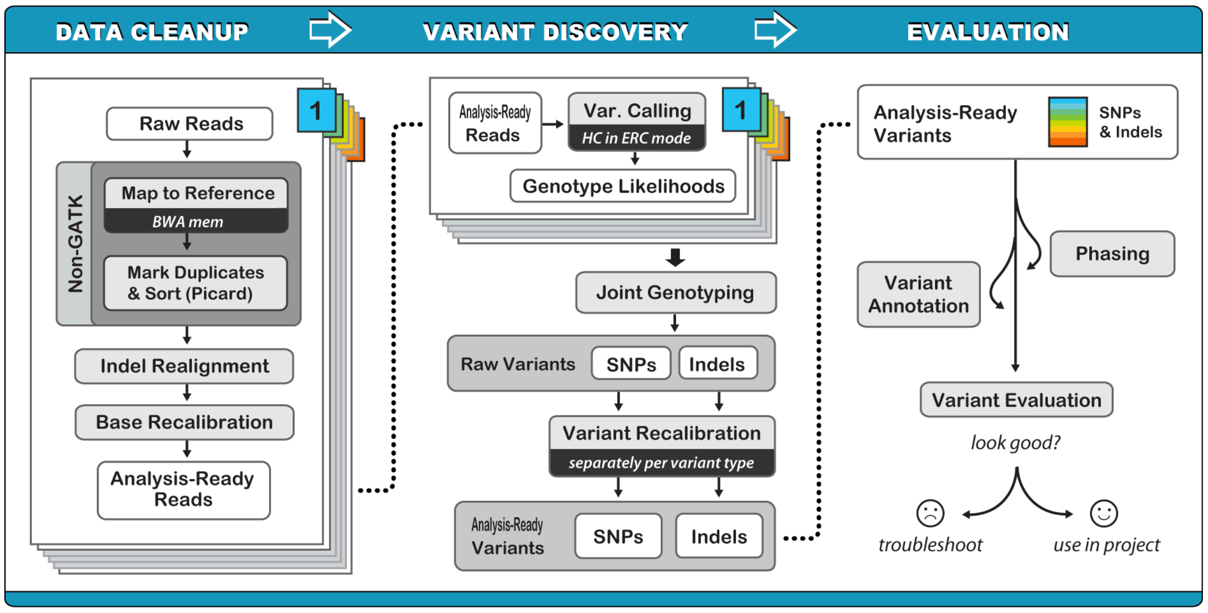
paired end reads in fastq format
("Section_3/module_1/bwa/reads/S1_1.fastq" etc)

reference sequence and annotation
(fasta, bed and gff3 files in "Section_3/reference" directory)

About GATK: Overall flow



GATK Process map



Section 3: Variome
Module 1: Processing and alignment

Preparing reference file and mapping

Preparing reference file
(You are here -> "Section_3/reference")

```
bwa index reference.fasta

samtools faidx reference.fasta

java -jar ~/bin/picard-tools-1.101/
CreateSequenceDictionary.jar R=reference.fasta
O=reference.dict
```

Mapping using bwamem
(Section_3/module_1/bwa)

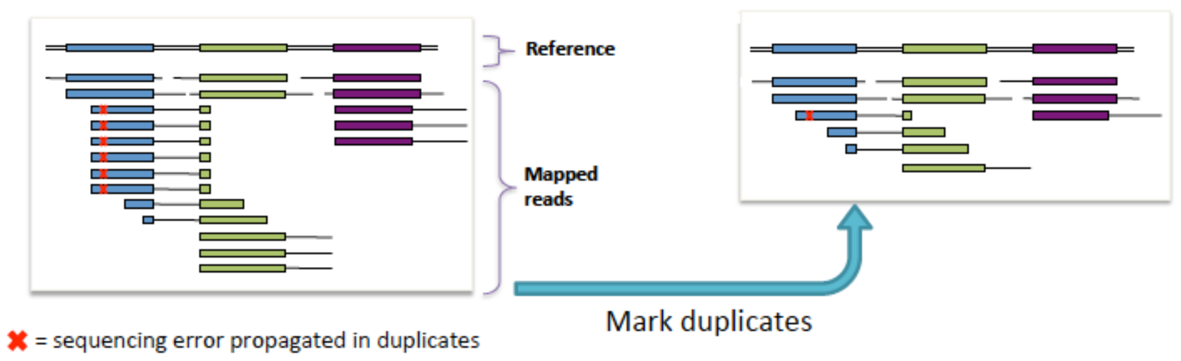
Important information about reads is also encoded simultaneously (library name, sample name, read group etc). These are useful for analysis later.

```
cd ../module_1/bwa

for i in S1 S2 S3 S4;do bwa mem -t 8 -M -R "@RG
\tID:"$i"_RG1\tPL:illumina\tPU:"$i"_RG1_UNIT1\tLB:"$i"-
lib1\tSM:"$i ../../reference/reference.fasta
reads/"$i"_1.fastq reads/"$i"_2.fastq >"$i".bwa.1.sam;done
```

Duplicate reads

Marking Duplicates



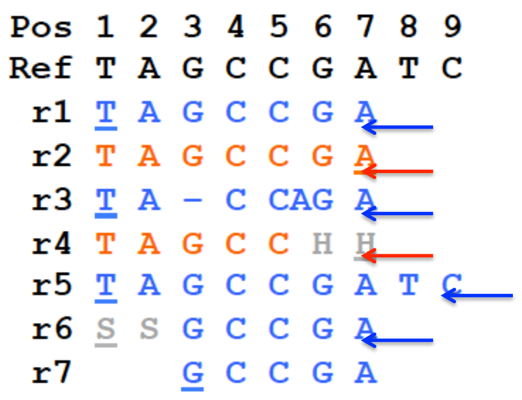
For correct estimation of variant likelihoods, we need our reads to represent the correct proportions of molecules in the library. (actually we also want our library to represent the proportions of original biological sample, and should be wary of biases introduced by PCR etc, but right now let's worry only about making sure we don't sequence a molecule more than once). One way of doing this is finding out which sequences are highly likely to originate from the same DNA fragment, and then removing all but one of that set.

Recognizing duplicates

Marking Duplicates

Finding reads that start at the same location. And, if paired end, that have their partners also mapping at the same starting location.

We can't simply compare the read sequences because sequencing is error prone and will likely lead to high underestimation of duplicates.



```
Pos 1 2 3 4 5 6 7 8 9
Ref T A G C C G A T C
 r1 T A G C C G A
 r2 T A G C C G A
 r3 T A - C CAG A
 r4 T A G C C H H
 r5 T A G C C G A T C
 r6 S S G C C G A
 r7     G C C G A
```

Blue maps to forward strand
Orange maps to reverse strand
Grey bases are clipped

Underlined is the expected 5' start of the read, given the mapping

So...what are the duplicate sets?

Removing Duplicates

(Section_3/module_1/bwa)

Sort and convert to bam

```
for i in S1 S2 S3 S4;do  samtools view -bS "$i".bwa.sam |
samtools sort - "$i".bwa.sorted;done
```

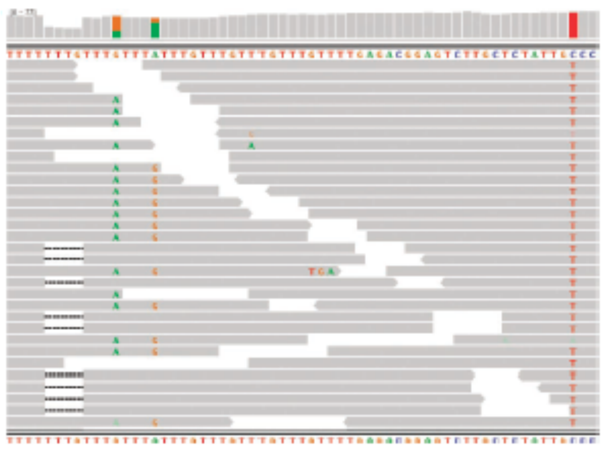
Removing duplicates with Picard tools

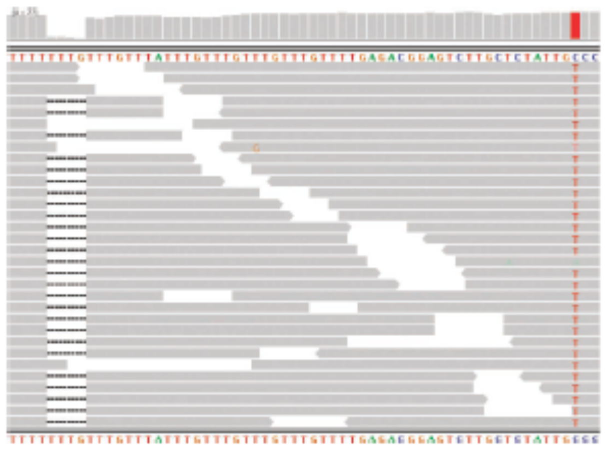
```
for i in S1 S2 S3 S4;do java -Xmx8g -jar ~/bin/picard.jar
MarkDuplicates MAX_FILE_HANDLES_FOR_READ_ENDS_MAP=1000
REMOVE_DUPLICATES=true INPUT="$i".bwa.sorted.bam
OUTPUT="$i".dedup.bam METRICS_FILE="$i".dedup_metrics
ASSUME_SORTED=true;done
```

Then you can look at some examples of before-and-after deduplication reads/ alignment using "`samtools faidx`" and "`samtools tview`" (or IGV)

Refining Alignments

Read aligners like bwa etc look at every read independently and try to find the best alignment for every read. This may lead to spurious SNPs because of slightly "off target" mappings, especially in presence of small indels (e.g. left figure below). Realigning all such reads in this region simultaneously by making use of multiple sequence alignment algorithms leads to more concordant alignments. This gets rid of many false positive SNPs which are merely mapping artifacts (right figure below)


HiSeq data, raw BWA alignments


HiSeq data, after MSA

Realignment around indels

(Section_3/module_1/bwa)

Index our de-duplicated bam files

```
for i in S1 S2 S3 S4;do samtools index "$i".dedup.bam;done
```

Find intervals to analyze

```
for i in S1 S2 S3 S4;do java -Xmx8g -jar ~/bin/
GenomeAnalysisTK.jar -T RealignerTargetCreator -R ~/
WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -I
"$i".dedup.bam -o "$i".realignment.intervals;done
```

Realign in these loci

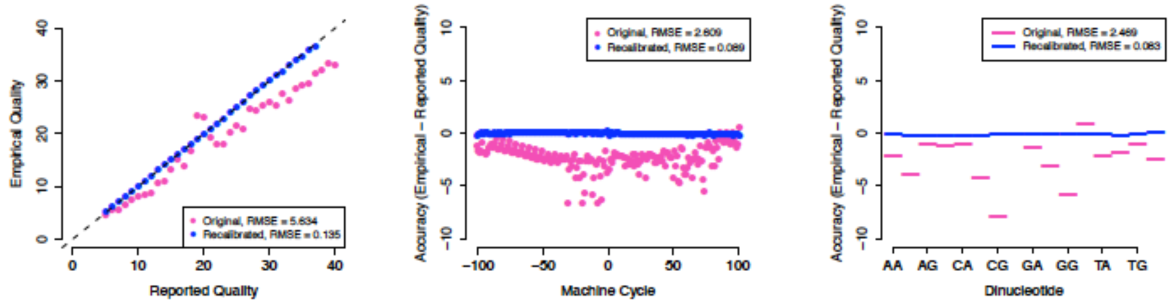
```
for i in S1 S2 S3 S4;do java -Xmx8g -jar ~/bin/
GenomeAnalysisTK.jar -T IndelRealigner -R ~/WORKSHOP_RESOURCES/
Section_3/reference/reference.fasta -I "$i".dedup.bam  -
targetIntervals "$i".realignment.intervals -o
"$i".dedup.realigned.bam;done
```

Base Recalibration (in presence of a truth set)

To improve base quality values, mismatches with reference are analyzed. Assuming that any mismatch which isn't a known SNP is an "error", base qualities can be readjusted to more closely model the reality (removing systematic errors in original base quality reports).

However, this can only be done in the presence of a substantial set of known True Positives (i.e. a large set of known SNPs). Since we don't have that (yet), we'll skip this and come back to it later…

The figure below shows the result of recalibrating errors from original reported qualities to those obtained using mapping data (after filtering out known SNPs).

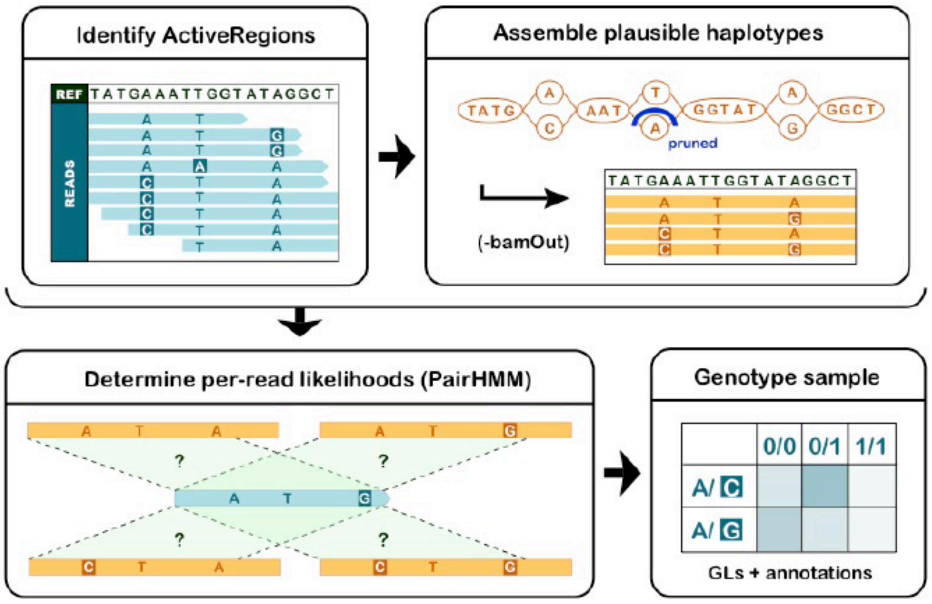


Paired end Hiseq data

Section 3: Variome
Module 2: Variant calling

Introduction

HaplotypeCaller is the workhorse of GATK's variant calling process. It calls variants by assembling reads in "active regions" into haplotypes (completely independent of reference sequence mapping) and then estimating likelihoods of genotypes at variant loci based on how well each read represents those assembled haplotypes.



Running HaplotypeCaller

(Section_3/module_2/haplo)

Prepare files

```
cd ../../module_2
mkdir haplo
cd haplo

for i in S1 S2 S3 S4;do ln -s ../../module_1/
bwa/"$i".dedup.realigned.bam;done

for i in S1 S2 S3 S4;do ln -s ../../module_1/
bwa/"$i".dedup.realigned.bai;done
```

Run HaplotypeCaller with GVCF

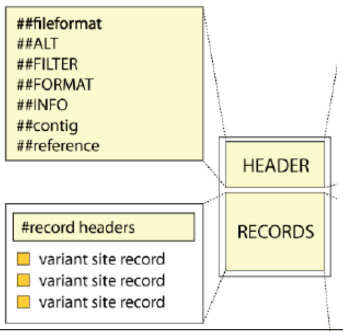
```
for i in S1 S2 S3 S4;do  java -Xmx8g -jar ~/bin/
GenomeAnalysisTK.jar -T HaplotypeCaller  -R ~/
WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -I
"$i".dedup.realigned.bam -ERC GVCF -ploidy 2 -o
"$i".dedup.g.vcf;done
```

Some default settings for HaplotypeCaller

--maxReadsInRegionPerSample	10000
--min_base_quality_score	10
--minReadsPerAlignmentStart	10
--sample_ploidy	2
--standard_min_confidence_threshold_for_calling	30.0
--standard_min_confidence_threshold_for_emitting	30.0
--max_alternate_alleles	6
--maxNumHaplotypesInPopulation	128

See Details at
https://www.broadinstitute.org/gatk/gatkdocs/
org_broadinstitute_gatk_tools_walkers_haplotypecaller_HaplotypeCaller.php

The VCF file format

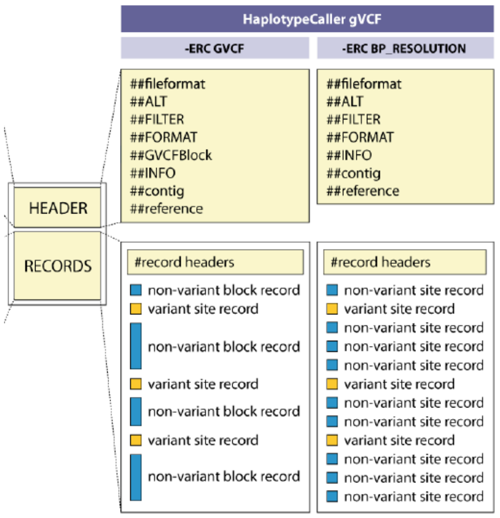


Full details: https://samtools.github.io/hts-specs/VCFv4.2.pdf

An example

```
##fileformat=VCFv4.1
##ALT=<ID=NON_REF,Description="Represents any possible alternative allele at this location">
##FILTER=<ID=LowQual,Description="Low quality">
##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt alleles in the order listed">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth (reads with MQ=255 or with bad mates are
filtered)”>
##INFO=<ID=AC,Number=A,Type=Integer,Description="Allele count in genotypes, for each ALT allele, in the same order as
listed">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency, for each ALT allele, in the same order as listed">
##contig=<ID=D_viviparus-1.0_Cont486,length=89705>
##contig=<ID=D_viviparus-1.0_Cont375,length=119898>
##reference=file:///home/ec2-user/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta
#CHROM      POS   ID      REF   ALT    QUAL FILTER        INFO  FORMAT      S1      S2      S3      S4
D_viviparus-1.0_Cont486  255    .         T       C       2156.88         .
        AC=2;AF=0.250;AN=8;DP=168;FS=0.000;MLEAC=2;MLEAF=0.250;MQ=60.00;QD=29.09;SOR=0.818
        GT:AD:DP:GQ:PGT:PID:PL        1/1:0,49:49:99:1|1:255_T_C:2197,147,0 0/0:35,0:35:99:.:.:0,99,1485
        0/0:39,0:39:99:.:.:0,102,1497        0/0:44,0:44:99:.:.:0,100,1742
```

Using GVCFs to combine sample-wise variants



```
##fileformat=VCFv4.1
.
.
.
##GVCFBlock0-1=minGQ=0(inclusive),maxGQ=1(exclusive)
##GVCFBlock1-2=minGQ=1(inclusive),maxGQ=2(exclusive)
.
.
.
#CHROM      POS   ID      REF   ALT      QUAL FILTER
        INFO  FORMAT       S1      S2      S3      S4
D_viviparus-1.0_Cont486  1       .       A       <NON_REF>.        .
        END=4        GT:DP:GQ:MIN_DP:PL        0/0:17:48:17:0,48,720
D_viviparus-1.0_Cont486  5       .       T       <NON_REF>.        .
        END=5        GT:DP:GQ:MIN_DP:PL        0/0:18:31:18:0,31,669
D_viviparus-1.0_Cont486  6       .       G       <NON_REF>.        .
        END=9        GT:DP:GQ:MIN_DP:PL        0/0:18:51:18:0,51,765
.
.
.
```

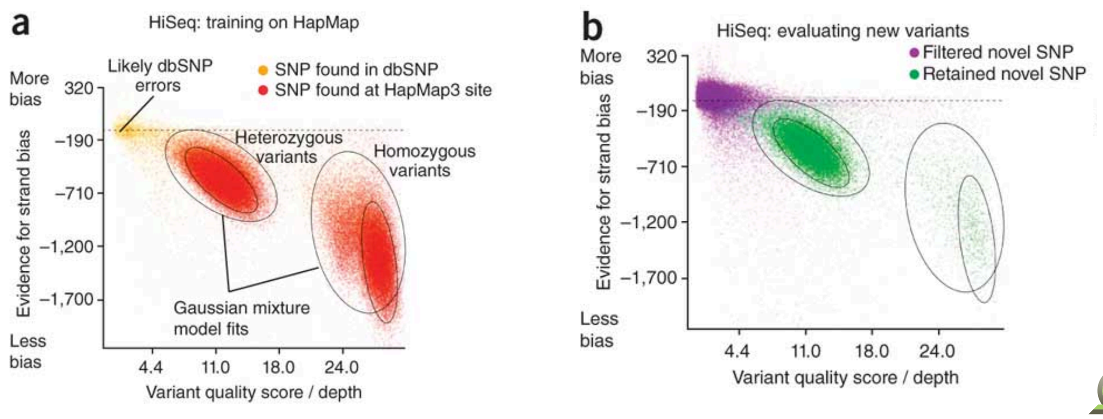
(Section_3/module_2/haplo)

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T GenotypeGVCFs -R
~/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta $(for
i in S1 S2 S3 S4;do echo -n "--variant "$i".dedup.g.vcf ";done)
-o all_raw.vcf
```

Variant Quality recalibration for refinement

To get a higher confidence set of real SNPs, we can look at a truth set (if we have one) of real SNPs and analyze what values various relevant metrics take for them. e.g. you may just pick up very rare (and potentially spurious) SNPs just because of very high depth of coverage. Looking at various metrics (Variant quality score/Depth, strand bias etc) may separate real SNPs with False Positives (figures below).

So, first we calibrate using known SNPs, then use those calibrations to filter out potential False Positives and obtain a final analysis-ready variant set.



Using hard filters

However, last page is useless for us since we don't actually have a truth set.

We still want to set up a filter to refine our raw variant set. So, we'll use some hard filters (i.e. thresholds pre-decided rather than dynamically calibrated based on data). We will use values recommended by GATK best practices, though these numbers can be changed based on any insight you may have into your specific case.

QD	: Quality by Depth	< 2.0
FS	: FisherStrand	> 60.0
MQ	: RMS Mapping Quality	< 40.0
MQRankSum	: Mapping Quality Rank Sum	< -12.5
ReadPosRankSum	: Read Position Rank Sum	< -8.0

In addition, we will also apply a depth of coverage filter (even though GATK team advises that it isn't as critical with HaplotypeCaller as with its older and almost obsolete cousin "UnifiedGenotyper"). We just want high confidence SNPs to generate a raw "truth set". So, we'll apply a relatively strict Depth filter. GATK used to suggest Depth of Coverage (DP) > (mean+5*sd).

We will use DP > (median + 2*MAD)

Setting stage for filtering SNPs

Collecting SNPs and getting coverage
(Section_3/module_2/var_filt)

Prepare Files

```
cd ..
mkdir var_filt
cd var_filt/
ln -s ../haplo/all_raw.vcf
```

Extract SNPs from the “raw” vcf file

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T
SelectVariants  -R ~/WORKSHOP_RESOURCES/Section_3/
reference/reference.fasta -V all_raw.vcf -selectType SNP -
o raw_snps.vcf
```

Getting DP filter threshold

Collecting SNPs and getting coverage
(Section_3/module_2/var_filt)

Finding base-wise coverages over the reference contigs (in order to find the DP filter threshold)

```
for i in S1 S2 S3 S4;do ln -s ../../module_1/
bwa/"$i".dedup.realigned.bam;done

for i in S1 S2 S3 S4;do coverageBed -abam
"$i".dedup.realigned.bam -b ../../reference/
reference.fasta.bed -d >"$i".coverage.bed;done
```

We will find the median and MAD (median absolute deviation) in R. This is done after adding the depths over all the samples:

```
S1<-read.table("S1.coverage.bed",header=F,stringsAsFactors=F)
S2<-read.table("S2.coverage.bed",header=F,stringsAsFactors=F)
S3<-read.table("S3.coverage.bed",header=F,stringsAsFactors=F)
S4<-read.table("S4.coverage.bed",header=F,stringsAsFactors=F)
sum<-S1$V6+S2$V6+S3$V6+S4$V6
summary(sum[sum<=(median(sum)+(2*mad(sum)))])
```

Applying SNP filters

Since we are only using DP to get a strict set for the purpose of base recalibration, we are sloppy here and using bedtools coveragebed utility to get coverage (also, partly because we want to introduce you to the convenient and useful coveragebed utility). If you really want to get proper depth numbers to set your DP filter, you should use the DepthofCoverage tool of GATK itself (as it takes care of any base filters that are applied in GATK before counting depths).

Also, remember that DP doesn't need to be used with HaplotypeCaller, and we won't use it to get our final SNP set anyway.

Now, we can apply our SNP filter!

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T
VariantFiltration -R ~/WORKSHOP_RESOURCES/Section_3/
reference/reference.fasta -V raw_snps.vcf -o
raw_snps_filtered.vcf  --filterExpression " QD < 2.0 " --
filterName "QD"  --filterExpression " FS > 60.0 " --
filterName "FS" --filterExpression " MQ < 40.0 " --filterName
"MQ" --filterExpression " MQRankSum < -12.5 " --filterName
"MQRankSum" --filterExpression " ReadPosRankSum < -8.0 " --
filterName "ReadPosRankSum" --filterExpression " DP > 268 "
--filterName "DP"
```

Applying indel filters

(Section_3/module_2/var_filt)

Now, we'll repeat filtering with indels too (using separate thresholds recommended by GATK best practices)

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T SelectVariants
-R ~/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -V
all_raw.vcf -selectType INDEL -o raw_indels.vcf
```

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T
VariantFiltration -R ~/WORKSHOP_RESOURCES/Section_3/reference/
reference.fasta -V raw_indels.vcf -o raw_indels_filtered.vcf
--filterExpression " QD < 2.0 " --filterName "QD"  --
filterExpression " FS > 200.0 " --filterName "FS" --
filterExpression " ReadPosRankSum < -20.0 " --filterName
"ReadPosRankSum"
```

Combining variants

(Section_3/module_2/var_filt)

We can now combine the SNPs and indels into a single variants file that can be used as a “truth set” to recalibrate bases (that we talked about in Module 1)

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T
CombineVariants  -R ~/WORKSHOP_RESOURCES/Section_3/
reference/reference.fasta --variant raw_snps_filtered.vcf
--variant raw_indels_filtered.vcf -o
raw_combined_filtered.vcf -genotypeMergeOptions UNSORTED --
printComplexMerges
```

Using Variant set for base quality recalibration

(Section_3/module_1/bwa)

Prepare Files, get recalibration data and apply it to update base quality values

```
cd ../../module_1/bwa
ln -s ../../module_2/var_filt/raw_combined_filtered.vcf

for i in S1 S2 S3 S4;do java -Xmx8g -jar ~/bin/
GenomeAnalysisTK.jar -T BaseRecalibrator  -R ~/
WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -I
"$i".dedup.realigned.bam -knownSites raw_combined_filtered.vcf
-o "$i".recal_data.table;done

for i in S1 S2 S3 S4;do java -Xmx8g -jar ~/bin/
GenomeAnalysisTK.jar -T PrintReads   -R ~/WORKSHOP_RESOURCES/
Section_3/reference/reference.fasta -I
"$i".dedup.realigned.bam -BQSR "$i".recal_data.table -o
"$i".recal_reads.bam;done
```

Variant Calling again

With this presumably better set of base qualities, we'll repeat our earlier steps for variant calling (i.e. haplotypecaller followed by combining the sample GVCFs)

(Section_3/module_2/haplo)

```
cd ../../module_2/haplo

for i in S1 S2 S3 S4;do ln -s ../../module_1/
bwa/"$i".recal_reads.bam;done
for i in S1 S2 S3 S4;do ln -s ../../module_1/
bwa/"$i".recal_reads.bai;done

for i in S1 S2 S3 S4;do  java -Xmx8g -jar ~/bin/
GenomeAnalysisTK.jar -T HaplotypeCaller  -R ~/WORKSHOP_RESOURCES/
Section_3/reference/reference.fasta -I "$i".recal_reads.bam  -ERC
GVCF -ploidy 2 -o "$i".recal.g.vcf -bamout
"$i".recal.haplo.bam ;done

java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T GenotypeGVCFs -R ~/
WORKSHOP_RESOURCES/Section_3/reference/reference.fasta $(for i in
S1 S2 S3 S4;do echo -n "--variant "$i".recal.g.vcf ";done) -o
all_recal.vcf
```

"-bamout" option is just to get a bam file which can then be visualized using IGV or "samtools tview" if you want to look at something closely.

Final SNPs hard filtering

We will again filter the variants with the hard filters introduced before. While we will stop here for the demonstration, usually one wants to see some sort of convergence of results before stopping. So, if you see a significant change in the number of variants detected as compared to the last round, you can do the same cycle all over again (i.e. using SNPs to recalibrate bases followed by calling and filtering variants again)

(Section_3/module_2/var_filt)

```
cd ../var_filt/
ln -s ../haplo/all_recal.vcf

java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T SelectVariants  -R
~/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -V
all_recal.vcf -selectType SNP -o recal_snps.vcf

java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T  VariantFiltration
-R ~/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -V
recal_snps.vcf -o recal_snps_filtered.vcf  --filterExpression " QD
< 2.0 " --filterName "QD"  --filterExpression " FS > 60.0 " --
filterName "FS" --filterExpression " MQ < 40.0 " --filterName "MQ"
--filterExpression " MQRankSum < -12.5 " --filterName "MQRankSum"
--filterExpression " ReadPosRankSum < -8.0 " --filterName
"ReadPosRankSum”
```

Final indel hard filtering

We do apply hard filters for indels again.

(Section_3/module_2/var_filt)

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T SelectVariants
-R ~/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta -V
all_recal.vcf -selectType INDEL -o recal_indels.vcf

java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T
VariantFiltration -R ~/WORKSHOP_RESOURCES/Section_3/reference/
reference.fasta -V recal_indels.vcf -o
recal_indels_filtered.vcf  --filterExpression " QD < 2.0 " --
filterName "QD"  --filterExpression " FS > 200.0 " --filterName
"FS" --filterExpression " ReadPosRankSum < -20.0 " --filterName
"ReadPosRankSum”
```

Combining variants for further analysis

Combining SNPs and indels gives a common variant file which can be used for further analysis. In our case, we have a pre-generated file which will be used in Module 4

```
java -Xmx8g -jar ~/bin/GenomeAnalysisTK.jar -T
CombineVariants  -R ~/WORKSHOP_RESOURCES/Section_3/reference/
reference.fasta --variant recal_snps_filtered.vcf --variant
recal_indels_filtered.vcf -o recal_combined_filtered.vcf -
genotypeMergeOptions UNSORTED --printComplexMerges
```

As said before, you should compare the change in variants after this round of recalibration and calling (but we will move on to Module 4 regardless of the change!)

Section 3: Variome
Module 2: Variant calling (cont' ed)

Visualization of variants
Variants in VCF format can be visualized using the Integrative Genomics Viewer (IGV), a high-performance visualization tool for interactive exploration of large, integrated genomic datasets (http://www.broadinstitute.org/igv/). IGV supports a wide variety of data types, including next-generation sequence data and genomic annotations.

Options for installing and running IGV
(http://www.broadinstitute.org/software/igv/download)

1. (Mac only) Download and run the Mac application; or
2. (All systems) Use the Java Web Start buttons; or
2. (All systems) Download the binary distribution and run IGV from the command line.

Creating a .genome File
1. Click Genomes>Create .genome File. IGV displays a window where you enter the information.
2. Enter an ID and a descriptive name for the genome (e.g., D_viviparus).
3. Enter the path to the FASTA file for the genome (`reference.fasta`). If the FASTA file has not already been indexed, an index will be created during the import process. This will generate a file with a ".fai" extension which must be in the same directory as the FASTA file.
4. Specify the gene file (`reference.gff3`).
5. Click Save. IGV displays the Genome Archive window.
6. Select the directory in which to save the genome archive (*.genome) file and click Save. IGV saves the genome and loads it into IGV.

Loading data
1. Select File>Load from File. IGV displays the Select Files window.
2. Select one or more data files or sample information files, then click OK.

Please load the following files:

```
recal_combined_filtered.vcf
S1.recal.haplo.bam
S1.dedup.realigned.bam
```

Section 3: Variome
Module 3: Variant annotation

Using SnpEff (http://snpeff.sourceforge.net), we will annotate and predict the effects of variants on genes (such as amino acid changes). SnpEff is written in Java and runs on Unix/Linux, OSX and Windows. It accepts input files in VCF/BED format, and can provide consequence terms defined by the Sequence Ontology (http://www.sequenceontology.org) and in HGVS notation (http://www.hgvs.org/mutnomen/).

Building databases
SnpEff needs a database to perform genomic annotations. In order to build a database for a new genome, you need to:

1. Configure a new genome in SnpEff's config file.
1a. Add genome entry to snpEff's configuration by editing the snpEff.config file.

```
gedit ~/bin/snpEff/snpEff.config
```

Add the following lines, save the file and exit gedit.

```
# Dictyocaulus_viviparus
D_viviparus.genome : Dictyocaulus_viviparus
```

1b (optional). If the genome uses a non-standard codon table, add codon table parameter. Please see SnpEff documentation for detail (http://snpeff.sourceforge.net/SnpEff_manual.html).

2. Create a directory for this new genome.

```
mkdir ~/bin/snpEff/data/D_viviparus/
```

3. Get the reference genome sequence in FASTA format.

```
ln -s ~/WORKSHOP_RESOURCES/Section_3/reference/reference.fasta
~/bin/snpEff/data/D_viviparus/sequences.fa
```

4. Get genome annotations from GFF file.

```
ln -s ~/WORKSHOP_RESOURCES/Section_3/reference/reference.gff3
~/bin/snpEff/data/D_viviparus/genes.gff
```

5. Build a SnpEff database.

```
java -Xmx8g -jar ~/bin/snpEff.jar build -gff3 -v D_viviparus
```

You can check the database to see if the features (genes, exons, UTRs, etc.) have been correctly incorporated, by taking a look at the database.

```
java -Xmx8g -jar ~/bin/snpEff.jar dump D_viviparus | less
```

Running SnpEff

1. Change directory to where the SnpEff output files will be saved.

```
cd ~/WORKSHOP_RESOURCES/Section_3/module_3
```

2. You can annotate the vcf file by running the following command. Command line option –v switches on the "verbose" mode, which can be useful for debugging.

```
java -Xmx8g -jar ~/bin/snpEff.jar -v D_viviparus
~/WORKSHOP_RESOURCES/Section_3/module_2/var_filt/recal_combined_f
iltered.vcf > recal_combined_filtered.eff.vcf
```

SnpEff adds annotation information (‘ANN’ tag) to the INFO field of a VCF file. The INFO field is the eighth column of a VCF file. SnpEff updates the header of the VCF file to add the command line options used to annotate the file as well as SnpEff's version, so you can keep track of what exactly was done.

```
less recal_combined_filtered.eff.vcf
```

3. SnpEff creates an additional output file showing overall statistics. This "stats" file is an HTML file, which can be opened using a web browser.

```
chrome snpEff_summary.html
```

4. SnpEff also generates a (tab separated) TXT file having counts of number of variants affecting each transcript and gene.

```
head snpEff_genes.txt
```

Filter and manipulate annotated VCF files using SnpSift

1. Once your genomic variants have been annotated, you need to filter them out in order to find the "interesting/relevant variants". SnpSift helps to perform this VCF file manipulation and filtering. It can be used to extract fields from a VCF file to a tab separated TXT format that you can easily load in R, Excel, etc.

```
cat recal_combined_filtered.eff.vcf |
~/bin/snpEff/scripts/vcfEffOnePerLine.pl | java -Xmx8g -jar
~/bin/SnpSift.jar extractFields - CHROM POS REF ALT AF
"ANN[*].ALLELE" "ANN[*].EFFECT" "ANN[*].IMPACT" "ANN[*].GENE"
"ANN[*].HGVS_C" "ANN[*].HGVS_P" > recal_combined_filtered.eff.txt
```

```
head recal_combined_filtered.eff.txt
```

2. You can now easily list, for instance, the coding variants identified in your genes of interest (e.g., DICVIV_10165 and DICVIV_11294)

```
cat recal_combined_filtered.eff.txt | grep -v "MODIFIER" | grep -
E "DICVIV_10165|DICVIV_11294"
```