

Variant calling

Variant calling process and terminology,
challenges,
statistical approaches,
read simulation, VCF format,
variant calling with Bcftools and Freebayes,
variant filtering and annotation
Capstone project I
Capstone project II

Required software

Install dwgsim

```
$ conda install -c bioconda dwgsim
```

Install bcftools

```
$ conda install -c bioconda bcftools
```

Install vcftools

```
$ conda install -c bioconda vcftools
```

Install bedtools

```
$ conda install -c bioconda bedtools
```

Install freebayes

```
$ conda install -c bioconda freebayes
```

Install bamaddrg

```
$ conda install -c bioconda bamaddrg
```

Required software

Install sambamba

```
$ conda install -c bioconda sambamba
```

Install SnpEff

```
$ conda install -c "bioconda/label/cf201901" snpeff
```

Install SnpSift

```
$ conda install -c "bioconda/label/cf201901" snpsift
```

Introduction to variant calling

Variant calling is a process of identification of differences between the reference and the sequencing reads

Variant calling process



Clean and trim the reads

Align the reads to the reference

Correct and refine alignments (optional)

Call the variants

Filter the variant based on user-defined properties

Annotate the variants

Introduction to variant calling

Variant calling terminology

- **Ploidy:** number of complete sets of chromosomes in a cell, corresponds to number of possible individual genetic variants per cell. The ploidy of procaryotes is 1.
- If we consider a collection of cells as a studied population (somatic mutations, prokaryotes), then the number of possible alleles is equal to **$N \times \text{ploidy}$** , where N is a number of cells
- **Haplotype** is a group of alleles inherited together from the same parent, these alleles are located close together and rarely undergo recombination. These alleles are also called “phased variants”
- **Genotype** is a collection of all known alleles
- **Single nucleotide polymorphisms (SNPs)** are single nucleotide substitutions, that occur with the frequency of over 1% in the population
- **Single nucleotide variant (SNVs)** are the same as **SNPs**, but with frequency below 1%

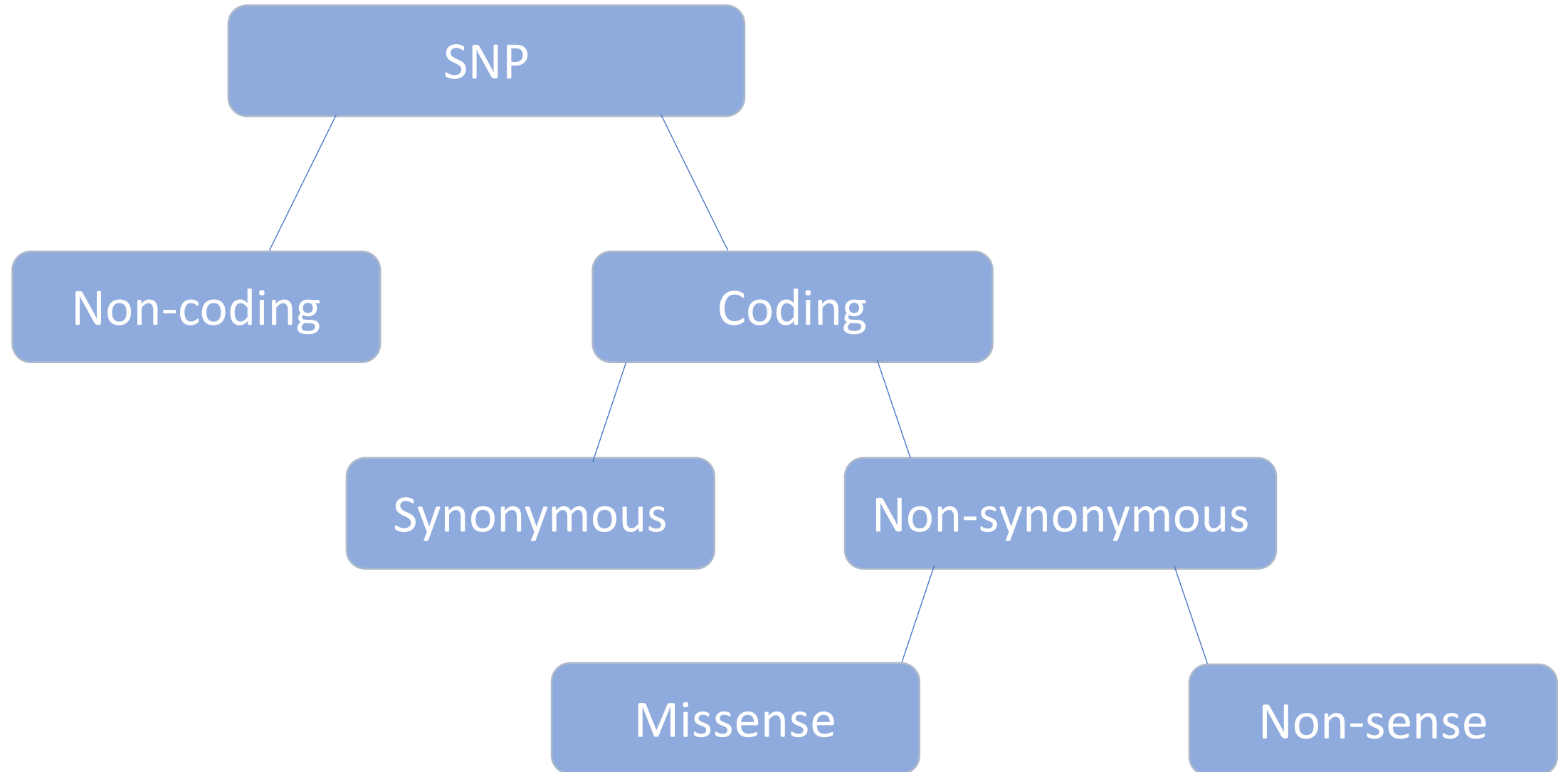
Introduction to variant calling

Variant calling terminology

- There is no clarity in SNP vs SNV definition and they are frequently used interchangeably
- **Short insertions and deletions (Indels)** are insertions or deletions of relatively small stretches of nucleotides. One of the length threshold used is less than 1000 bp to call the insertion or deletion an indel
- **Gross chromosomal rearrangements (GCRs)** – large deletions, insertions, amplifications or translocations (over 1000 bp)

Introduction to variant calling

Classification of single nucleotide substitutions



Introduction to variant calling

Challenges in variant calling

- **Library preparation bias:** Success of a variant detection depends on coverage, i.e. a genomic position must be covered by a certain number of reads to consider a call reliable. Different library preparation protocols are prone to region biases, leading to insufficient coverage of certain regions
- **Artifact mutations** introduced during the library preparation process, for example endonuclease-mediated DNA fragmentation introduces more artifactual SNV/Indels compared to ultrasonication (PMID: 31899787)
- **PCR/Optical duplicates** can falsely increase allele frequencies or even introduce mutations
- **Errors** introduced during sequencing and base calling (platform-dependent)
- **Read alignment issues**, such as mapping repeat regions, alignment around indels

Introduction to variant calling

Statistical approaches in variant calling

- The software that detects the variants are called **variant callers**
- The purpose of a variant caller is to detect variants and separate true variants (signal) from various artifacts and errors introduced by laboratory procedures and sequencing (noise)
- Detection of SNPs and Indels is relatively straightforward, however sophisticated statistics is required to separate signal from noise

[Nat Rev Genet.](#) Author manuscript; available in PMC 2013 Mar

PMCID: PMC3593722

11.

NIHMSID: NIHMS436685

Published in final edited form as:

PMID: [21587300](#)

[Nat Rev Genet.](#) 2011 Jun; [12\(6\)](#): 443–451.

doi: [10.1038/nrg2986](#)

Genotype and SNP calling from next-generation sequencing data

[Rasmus Nielsen](#),^{*‡§} [Joshua S. Paul](#),^{||} [Anders Albrechtsen](#),[‡] and [Yun S. Song](#)^{§||}

Introduction to variant calling

Statistical approaches in variant calling

- We should differentiate between **variant** and genotype calling
- Variant calling aims at identifying bases in the query sequence that are different from the reference; genotype calling is a determination of genotype (heterozygous or homozygous)
- In prokaryotes variant calling and genotyping are the same
- Early methods in variant and genotype calling were based on simple cutoffs, and did not allow for uncertainty in the determination of variants and genotypes
- Current methods use probabilistic models that quantify the uncertainty of the calls

Introduction to variant calling

Early methods in variant calling

Based on simple threshold, for example:

- We include all variant with a quality **Q** \geq **20** (1/100 probability of variant called wrong)
- Calculate allele frequencies for variant sites
- If the allele frequency is **20 – 80%** call heterozygous genotype (for diploid organism), otherwise call homozygous genotype
- This method works quite well for sequencing depth over x20

Introduction to variant calling

Probabilistic methods in variant calling

- For moderate and low sequencing depths (< 20), simple threshold methods result in under-calling of heterozygous phenotypes
- Probabilistic methods provide quality scores that reflect the posterior probability of genotypes
- Genotype likelihood are based on quality scores for each read
- ✓ Let $X_i \rightarrow$ data in read i for a particular individual at a particular site with genotype G
- ✓ The probability $p(X_i | G)$ is then given by a simple rescaling of the quality score of X_i , and the genotype likelihood, $p(X | G)$, can be calculated directly from the data by taking the product of $p(X_i | G)$ over all i .

$$P(G \mid X_i) = (P(X_i \mid G) * P(G)) / P(X_i)$$

Introduction to variant calling

Probabilistic methods in variant calling

- Basic probabilistic model assumes independence of errors in the reads
- This assumption can be violated by the presence of PCR duplicates and alignment errors
- Some variant callers use a weighing scheme to account for correlated errors
- Variant calling can be improved by recalibrating base quality scores using empirical data
- A prior genotype probability can be assumed as equal probability for all possible genotypes or based on empirical data – genotypes observed in variant databases, like dbSNP

Introduction to variant calling

Commonly used, well established variant callers

bcftools created by Heng Li the author of samtools and many other tools

<http://www.htslib.org/doc/1.0/bcftools.html>

FreeBayes, originated from MIT <https://github.com/freebayes/freebayes>

GATK, from Broad Institute <https://gatk.broadinstitute.org/hc/en-us>

Specifically, GATK for Microbes: <https://gatk.broadinstitute.org/hc/en-us/articles/360060004292-Introducing-GATK-for-Microbes>

VarScan2, a well documented versatile alignments <https://varscan.sourceforge.net/>

Introduction to variant calling

Practice variant calling using simulated data

We will use ***dwgsim*** software <https://github.com/nh13/DWGSIM> to simulate the reads data

Other read simulation software:

wgsim <https://github.com/lh3/wgsim> by author of samtools

readsim <https://sourceforge.net/p/readsim/wiki/Home/> long read simulator for PacBio and Nanopore

Art <https://www.niehs.nih.gov/research/resources/software/biostatistics/art/index.cfm> a set of versatile and very sophisticated tools for read simulations

Introduction to variant calling

Simulate reads with **dwgsim**

dwgsim simulates reads from reference *fasta* to evaluate alignments and variant calling methods

SYNTAX: **dwgsim** [option] <reference> <output_prefix>

\$ **dwgsim -h** # read the help and try to understand the options

\$ **mkdir sim_reads** # we will work in the directory **sim_reads/** for this task

\$ **cd sim_reads**

\$ **dwgsim -H -N 10000 -1 100 -2 100 ../GENOMES/M_tuber/M_tuber_H37Rv.fasta output** # -H – simulate the reads from a haploid genome; -N <INT> - simulate N number of reads; -1 <INT> -2 <INT> - simulate read 1 and read 2 reads, where INT is length of the reads

Introduction to variant calling

Simulate reads with **dwgsim**

dwgsim output files

- *output.bfast.fastq.gz* – interleaved fastq file (read 1 is followed by read 2 in the same file)
- *output.bwa.read1.fastq.gz* – Read1 fastq file
- *output.bwa.read2.fastq.gz* – Read 2 fastq file
- *output.mutations.txt* – simulated mutations in tab delimited file
- *output.mutations.vcf* – simulated mutations in VCF file

Let's create a larger more, more realistic data set while setting sequencing errors and mutation rates

```
$ dwgsim -H -C 30.0 -1 100 -2 100 -e 0.002 -E 0.004 -r 0.001 -r 0.1
```

```
../GENOMES/M_tuber/M_tuber_H37Rv.fasta output # -C FLOAT – read coverage; -e – base error  
frequency in Read 1; -E - base error frequency in Read 2; -r – mutation frequency; -R – faction of  
indels among mutations
```

Introduction to variant calling

Simulate reads with **dwgsim**

dwgsim output files

- *output.bfast.fastq.gz* – interleaved fastq file (read 1 is followed by read 2 in the same file)
- *output.bwa.read1.fastq.gz* – Read1 fastq file
- *output.bwa.read2.fastq.gz* – Read 2 fastq file
- *output.mutations.txt* – simulated mutations in tab delimited file
- *output.mutations.vcf* – simulated mutations in VCF file

Let's create a larger more, more realistic data set while setting sequencing errors and mutation rates

```
$ dwgsim -H -C 30.0 -1 100 -2 100 -e 0.002 -E 0.004 -r 0.001 -r 0.1
```

```
../GENOMES/M_tuber/M_tuber_H37Rv.fasta output # -C FLOAT – read coverage; -e – base error  
frequency in Read 1; -E - base error frequency in Read 2; -r – mutation frequency; -R – faction of  
indels among mutations
```

Introduction to variant calling

Let's take a look at the reads

```
$ zcat output.bwa.read1.fastq.gz | head
```

Verify read numbers for both fastq files

```
$ zcat output.bwa.read1.fastq.gz | grep '@NC_000962' | wc -l
```

```
$ zcat output.bwa.read2.fastq.gz | grep '@NC_000962' | wc -l
```

```
$ wc -l output.mutations.txt # how many mutations
```

```
$ head output.mutations.txt # Take a look the the mutation file
```

NC_000962.3	1	T	A	3
NC_000962.3	10	G	C	3
NC_000962.3	38	G	A	3
NC_000962.3	56	A	T	3
NC_000962.3	58	C	-	3
NC_000962.3	63	C	A	3
NC_000962.3	70	C	T	3
NC_000962.3	92	G	C	3
NC_000962.3	96	T	C	3
NC_000962.3	106	C	T	3

Introduction to variant calling

Mutation *txt* file format:

1. the chromosome/contig name
2. the one-based position
3. the original reference base
4. the new reference base(s)
5. the variant strand(s)

Interpreting the mutation file, from the manual:

contig4 4 T K 1

A heterozygous mutation at position 4 of contig4 on the first strand, mutating the T base to a heterozygous K (G or T) SNP

contig6 22 A - 2

A heterozygous deletion of T at position 22 on the second strand

Introduction to variant calling

VCF file format

Let's take a look at the VCF file in **dwgsim**

\$ head output.mutations.vcf

```
##fileformat=VCFv4.1
##contig=<ID=NC_000962.3,length=4411532>
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=pl,Number=1,Type=Integer,Description="Phasing: 1 - HET contig 1, #2 - HET contig #2, 3 - HOM both contigs">
##INFO=<ID=mt,Number=1,Type=String,Description="Variant Type: SUBSTITUTE/INSERT/DELETE">
#CHROM  POS      ID      REF      ALT      QUAL     FILTER  INFO
NC_000962.3  1      .      T      A      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3  10     .      G      C      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3  38     .      G      A      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3  56     .      A      T      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
```

VCF format is text, human readable format designed to describe genomic variation

VCF can contain information on any number of samples/individuals, serving as a single data base for a large variant calling project

VCF files can get incredibly complex; we need specialized software to access and manipulate VCF files

Introduction to variant calling

VCF file format (continuation)

```
##fileformat=VCFv4.1
##contig=<ID=NC_000962.3,length=4411532>
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=pl,Number=1,Type=Integer,Description="Phasing: 1 - HET contig 1, #2 - HET contig #2, 3 - HOM both contigs">
##INFO=<ID=mt,Number=1,Type=String,Description="Variant Type: SUBSTITUTE/INSERT/DELETE">
#CHROM  POS      ID      REF      ALT      QUAL     FILTER  INFO
NC_000962.3  1      .      T      A      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3  10     .      G      C      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3  38     .      G      A      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3  56     .      A      T      100      PASS    AF=1.0;pl=3;mt=SUBSTITUTE
```

VCF format consists out 2 section header and a body

Lines in a header section start with #, special keywords start with ##

Header contains data on the format version of VCF file, **VCFv4.1** in our case

Contig data, ID and length, there is only 1 contig in our case since we are working with bacteria

Introduction to variant calling

VCF file format (continuation)

```
##fileformat=VCFv4.1
##contig=<ID=NC_000962.3,length=4411532>
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=pl,Number=1,Type=Integer,Description="Phasing: 1 - HET contig 1, #2 - HET contig #2, 3 - HOM both contigs">
##INFO=<ID=mt,Number=1,Type=String,Description="Variant Type: SUBSTITUTE/INSERT/DELETE">
#CHROM POS ID REF ALT QUAL FILTER INFO
NC_000962.3 1 . T A 100 PASS AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3 10 . G C 100 PASS AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3 38 . G A 100 PASS AF=1.0;pl=3;mt=SUBSTITUTE
NC_000962.3 56 . A T 100 PASS AF=1.0;pl=3;mt=SUBSTITUTE
```

Data stored in INFO lines describe variations across **all samples**

VCF file can also contain FORMAT lines, data types shown in FORMAT descriptor applies to individual samples

Introduction to variant calling

VCF file format (continuation)

Body of the file: 9 mandatory columns, and additional columns for individual samples, see

https://en.wikipedia.org/wiki/Variant_Call_Format

Name	Brief description (see the specification for details).	
1	CHROM	The name of the sequence (typically a chromosome) on which the variation is being called. This sequence is usually known as 'the reference sequence', i.e. the sequence against which the given sample varies.
2	POS	The 1-based position of the variation on the given sequence.
3	ID	The identifier of the variation, e.g. a dbSNP rs identifier, or if unknown a ".". Multiple identifiers should be separated by semi-colons without white-space.
4	REF	The reference base (or bases in the case of an indel) at the given position on the given reference sequence.
5	ALT	The list of alternative alleles at this position.
6	QUAL	A quality score associated with the inference of the given alleles.
7	FILTER	A flag indicating which of a given set of filters the variation has failed or PASS if all the filters were passed successfully.
8	INFO	An extensible list of key-value pairs (fields) describing the variation. See below for some common fields. Multiple fields are separated by semicolons with optional values in the format: <key>=<data>[,data].
9	FORMAT	An (optional) extensible list of fields for describing the samples. See below for some common fields.
+	SAMPLEs	For each (optional) sample described in the file, values are given for the fields listed in FORMAT

Introduction to variant calling

VCF file format (continuation)

Common INFO fields (**describe alleles**). There are many more fields described in VCF docs.

Name	Brief description
AA	ancestral allele
AC	allele count in genotypes, for each ALT allele, in the same order as listed
AF	allele frequency for each ALT allele in the same order as listed (use this when estimated from primary data, not called genotypes)
AN	total number of alleles in called genotypes
BQ	RMS base quality at this position
CIGAR	cigar string describing how to align an alternate allele to the reference allele
DB	dbSNP membership

Introduction to variant calling

VCF file format (continuation)

Common FORMAT fields (describe **genotypes**). There are many more fields described in VCF docs.

Name	Brief description
AD	Read depth for each allele
ADF	Read depth for each allele on the forward strand
ADR	Read depth for each allele on the reverse strand
DP	Read depth
EC	Expected alternate allele counts
FT	Filter indicating if this genotype was “called”
GL	Genotype likelihoods

- You don't need to remember INFO and FORMAT fields, they are represented large number of identifiers, many of which are invented, by the authors of variant calling software.
- Normally, only a few of those will be useful in your analysis.

Introduction to variant calling

VCF file format (continuation)

Closer look at VCF file variant entry

```
NC_000962.3    20004    .    G    T    363.989    .    AB=0;ABP=0;AC=1;AF=1;AN=1;AO=18;CIGAR=1X;DP=18;DPB=18;DPRA=0;EPP=3.49285;EPPR=0;
TI=0;LEN=1;MEANALT=1;MQM=60;MQMR=0;NS=1;NUMALT=1;ODDS=83.8115;PAIRED=1;PAIREDR=0;PAO=0;PQA=0;PQR=0;PRO=0;QA=479;QR=0;RO=0;RPL=18;RPP=42.0968;RF
=0;RPR=0;RUN=1;SAF=8;SAP=3.49285;SAR=10;SRF=0;SRP=0;SRR=0;TYPE=snp    GT:DP:RO:QR:AO:QA:GL    1:18:0:0:18:479:-43.3724,0
```

NC_000962.3	20004	.	G	T	363.989	.
1	2	3	4	5	6	7

Field	ID	Meaning
1	CHROM	Chromosome, contig
2	POS	Position of the variant
3	ID	Variant ID, like dbSNP id is available
4	REF	Reference allele
5	ALT	Alternative allele
6	QUAL	Variant quality
7	FILTER	This field is filled after applying a variant filter, before filtering this field is shown as ‘

Introduction to variant calling

VCF file format (continuation)

Closer look at VCF file variant entry

```
NC_000962.3    20004    .    G    T    363.989    .    AB=0;ABP=0;AC=1;AF=1;AN=1;AO=18;CIGAR=1X;DP=18;DPB=18;DPRA=0;EPP=3.49285;EPPR=0;
TI=0;LEN=1;MEANALT=1;MQM=60;MQMR=0;NS=1;NUMALT=1;ODDS=83.8115;PAIRED=1;PAIREDR=0;PAO=0;PQA=0;PQR=0;PRO=0;QA=479;QR=0;RO=0;RPL=18;RPP=42.0968;RI
=0;RPR=0;RUN=1;SAF=8;SAP=3.49285;SAR=10;SRF=0;SRP=0;SRR=0;TYPE=snp    GT:DP:RO:QR:AO:QA:GL    1:18:0:0:18:479:-43.3724,0
```

Red boxes select an INFO field

```
AB=0;ABP=0;AC=1;AF=1;AN=1;AO=18;CIGAR=1X;DP=18;DPB=18;DPRA=0;EPP=3.49285;EPPR=
```

INFO field descriptions are located in the header of the vcf files, where lines start with '##'

We can search for the INFO tag of interest with great, like this:

```
$ grep '^##INFO=<ID=AF' output.mutations.vcf # This command will search for AF tag on a VCF header
```

Introduction to variant calling

VCF file format (continuation)

Closer look at VCF file variant entry

```
NC_000962.3    20004    .    G    T    363.989    .    AB=0;ABP=0;AC=1;AF=1;AN=1;AO=18;CIGAR=1X;DP=18;DPB=18;DPRA=0;EPP=3.49285;EPPR=0;TI=0;LEN=1;MEANALT=1;MQM=60;MQMR=0;NS=1;NUMALT=1;ODDS=83.8115;PAIRED=1;PAIREDR=0;PAO=0;POA=0;POR=0;PRO=0;QA=479;QR=0;RO=0;RPL=18;RPP=42.0968;RF=0;RPR=0;RUN=1;SAF=8;SAP=3.49285;SAR=10;SRF=0;SRP=0;SRR=0;TYPE=snp    GT:DP:RO:QR:AO:QA:GL    1:18:0:0:18:479:-43.3724,0
```

Red boxes select Genotype and Sample fields, there may be multiple sample fields, this example contains one sample

Genotype field tags match data in a sample field

```
GT:DP:RO:QR:AO:QA:GL    1:18:0:0:18:479:-43.3724,0
```

GT → 1 : Genotype of this sample at this site

DP → 18 : Unfiltered reads depth at this site

RO → 0 : Reference allele observation count

QR → 0 : Sum of qualities for reference allele observation

AO → 18 : Alternative allele observation counts

QA → 479 : Sum of qualities for alternative allele observations

GL → -43.3724,0 : Genotype Likelihood, log10-scaled likelihoods of the data given the called genotype for each possible genotype generated from the reference and alternate alleles given the sample ploidy

Introduction to variant calling

VCF file format (continuation)

Closer look at VCF file variant entry

We can find out the meaning of FORMAT tags from VCF header

```
##FORMAT=<ID=GT, Number=1, Type=String, Description="Genotype">  
##FORMAT=<ID=GQ, Number=1, Type=Float, Description="Genotype Quality, the Phred-scaled marginal (or u  
e">  
##FORMAT=<ID=GL, Number=G, Type=Float, Description="Genotype Likelihood, log10-scaled likelihoods of  
sible genotype generated from the reference and alternate alleles given the sample ploidy">  
##FORMAT=<ID=DP, Number=1, Type=Integer, Description="Read Depth">  
##FORMAT=<ID=RO, Number=1, Type=Integer, Description="Reference allele observation count">  
##FORMAT=<ID=QR, Number=1, Type=Integer, Description="Sum of quality of the reference observations">  
##FORMAT=<ID=AO, Number=A, Type=Integer, Description="Alternate allele observation count">  
##FORMAT=<ID=QA, Number=A, Type=Integer, Description="Sum of quality of the alternate observations">
```

Use grep to find a description of the FORMAT tag, for example to search for GT tag

```
$ grep '##FORMAT=<ID=GT' freebayes.bwa.vcf
```

Introduction to variant calling

VCF file format (continuation)

Where to get information about VCF format:

- ✓ Official documentation: <https://samtools.github.io/hts-specs/VCFv4.2.pdf>
- ✓ Wikipedia article (it is rather good): https://en.wikipedia.org/wiki/Variant_Call_Format
- ✓ GATK explanation of VCF format: <https://gatk.broadinstitute.org/hc/en-us/articles/360035531692-VCF-Variant-Call-Format>

NEVER try to edit, filter or otherwise manipulate VCF files using general text processing tools! It will never end well.

Use only specialized software to work with VCF files!

Introduction to variant calling

VCF file format (continuation)

A list of VCF manipulation software:

- **Picard** <http://broadinstitute.github.io/picard/index.html> - java-based command line tools to manipulate SAM, BAM, CRAM, and VCF files
- **VCF-kit** <https://github.com/AndersenLab/VCF-kit> - command line set of utilities to manipulate VCF files
- **VCFtools** <https://vcftools.github.io/> - another command line set of tools to manipulate and analyze VCFs
- **BCFtools** <https://samtools.github.io/bcftools/bcftools.html> - a set of tools written by Heng Li to manipulate and analyze VCFs and their binary representations – BCF files

Calling variants

Calling variants with simulated reads using **BCFtools**

Change to **sim_reads/** directory

```
$ cd sim_reads
```

Map simulated reads with bwa

```
$ bwa mem -t 2 ../GENOMES/M_tuber/bwa_index/M_tuber_H37Rv.fasta output.bwa.read1.fastq.gz  
output.bwa.read2.fastq.gz > simulated.bwa.sam
```

Map simulated reads with bowtie 2

```
$ bowtie2 -p 2 --fr --very-sensitive -X 1000 -x ../GENOMES/M_tuber/bowtie2_index/M_tuber_H37Rv -1  
output.bwa.read1.fastq.gz -2 output.bwa.read2.fastq.gz -S simulated.bowtie2.sam # -X – 1000 sets  
maximum insert size, works well for most of the libraries
```

Check basic statistics for both sam files

```
$ samtools flagstat simulated.bwa.sam
```

```
$ samtools flagstat simulated.bowtie2.sam
```

Calling variants

Calling variants with simulated reads using **BCFtools**

Convert sam files to bam, sort and index

```
$ samtools view -@ 2 -b simulated.bwa.sam > simulated.bwa.bam
```

```
$ samtools view -@ 2 -b simulated.bowtie2.sam > simulated.bowtie2.bam
```

Sort

```
$ samtools sort -@ 2 simulated.bwa.bam -o simulated.bwa.sorted.bam
```

```
$ samtools sort -@ 2 simulated.bowtie2.bam -o simulated.bowtie2.sorted.bam
```

Index

```
$ samtools index simulated.bwa.sorted.bam
```

```
$ samtools index simulated.bowtie2.sorted.bam
```

Remove intermediate files

```
$ rm *.sam *.bowtie2.bam *.bwa.bam
```

Calling variants

Calling variants with simulated reads using **BCFtools**

Convert sam files to bam, sort and index

```
$ samtools view -@ 2 -b simulated.bwa.sam > simulated.bwa.bam
```

```
$ samtools view -@ 2 -b simulated.bowtie2.sam > simulated.bowtie2.bam
```

Sort

```
$ samtools sort -@ 2 simulated.bwa.bam -o simulated.bwa.sorted.bam
```

```
$ samtools sort -@ 2 simulated.bowtie2.bam -o simulated.bowtie2.sorted.bam
```

Index

```
$ samtools index simulated.bwa.sorted.bam
```

```
$ samtools index simulated.bowtie2.sorted.bam
```

Remove intermediate files

```
$ rm *.sam *.bowtie2.bam *.bwa.bam
```

Calling variants

Calling variants with simulated reads using **BCFtools**

Generate genotype likelihoods in VCF or BCF format using **bcftools mpileup**

Get familiar with **bcftools mpileup** by reading the manual
<https://samtools.github.io/bcftools/bcftools.html#mpileup>

bcftools variant calling how-to <https://samtools.github.io/bcftools/howtos/variant-calling.html>

Call variants in a two-stage process, first calculate genotype likelihoods with **mpileup**, then call variants with **call**

```
$ bcftools mpileup -Ovu -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta simulated.bwa.sorted.bam >  
genotypes.bwa.vcf # -O – specify output format v – VCF, u – uncompressed
```

```
$ bcftools mpileup -Ovu -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta simulated.bowtie2.sorted.bam >  
genotypes.bowtie2.vcf
```

Calling variants

Calling variants with simulated reads using **BCFtools**

Read BCFtools **call** manual and try to understand the options:

Call variants using **bcftools call**

`$ bcftools call --ploidy 1 -vm -O v genotypes.bwa.vcf > variants.bwa.vcf` # -O v – output vcf; --ploidy – set ploidy, 1 in our case; -v – output variant sites only; -m mutiallelic caller, alternative model for multiallelic and rare-variant calling designed to overcome known limitations in -c calling model (conflicts with -c)

`$ bcftools call --ploidy 1 -vm -O v genotypes.bowtie2.vcf > variants.bowtie2.vcf`

Put both commands on the same line to avoid creating an intermediate genotype.vcf file

`$ bcftools mpileup --threads 2 -Ov -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta simulated.bwa.sorted.bam | bcftools call --threads 2 --ploidy 1 -vm -O v > test.vcf` # bcftools commands can be connected into pipelines; used 2 processor cores with --threads command

Calling variants

Calling variants with simulated reads using **BCFtools**

Print detailed VCF statistics with **bcftools stats**

```
$ bcftools stats variants.bwa.vcf > variants.bwa.stats
```

```
$ less variants.bwa.stats # this file quite detailed, go through the stats command output and try to understand what it means
```

Run the same command on bowtie2 vcf file

```
$ bcftools stats variants.bowtie2.vcf > variants.bowtie2.stats
```

```
$ less variants.bowtie2.stats
```

Compare VCF files with stats

First, we will need to compress and index them

```
$ bgzip variants.bowtie2.vcf > variants.bowtie2.vcf.gz
```

```
$ bcftools index variants.bowtie2.vcf.gz # this will create an index with .csi format
```

```
$ bgzip variants.bwa.vcf > variants.bwa.vcf.gz
```

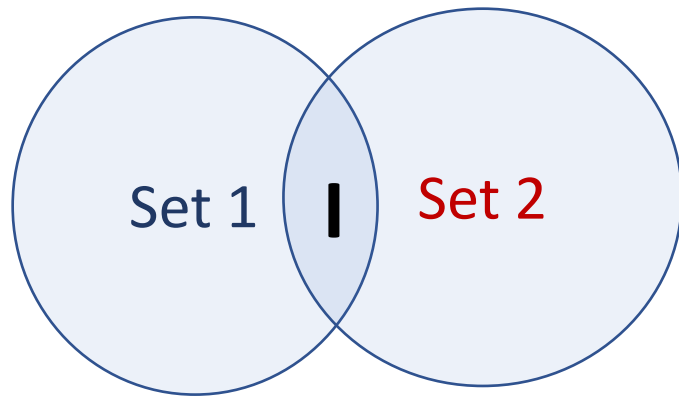
```
$ bcftools index variants.bwa.vcf.gz
```

Calling variants

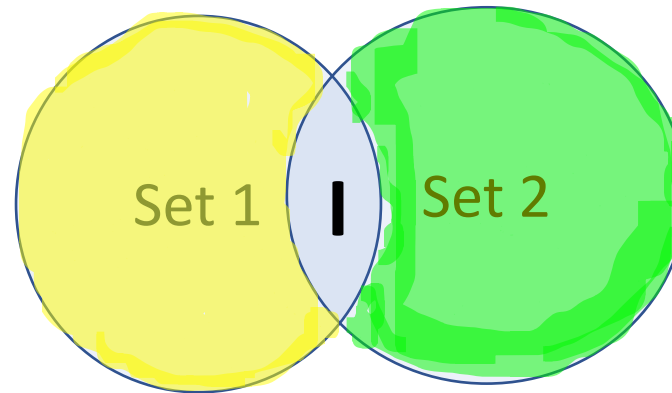
Calling variants with simulated reads using **BCFtools**

Compare VCF files with stats

\$ `bcftools isec variants.bwa.vcf.gz variants.bowtie2.vcf.gz -p vcf_comparison` # **isec** command will create intersects and complements of 2 files and print the output to **vcf_comparison/** directory



I is an Intersect



Yellow : complement of Set1

Green : complement of Set2

Calling variants

Calling variants with simulated reads using **BCFtools**

Compare VCF files with **isec**

bcftools isec command created a directory with 4 vcf files and a README files

```
$ ls -lh vcf_comparison
```

```
$ cat vcf_comparison/README.txt # print README file
```

How many variants are in common between BWA and bowtie2 vcf files?

```
$ bcftools stats vcf_comparison/0002.vcf | grep 'number of records'
```

```
$ bcftools stats vcf_comparison/0003.vcf | grep 'number of records'
```

How many variants are unique to BWA vcf file

```
bcftools stats vcf_comparison/0000.vcf | grep 'number of records'
```

How many variants are unique to bowtie2 vcf file

```
bcftools stats vcf_comparison/0001.vcf | grep 'number of records'
```


Calling variants

Calling variants with simulated reads using **BCFtools**

Determine **concordance** between “**ground truth**” data set and variants based in BWA and bowtie2 alignments

- ✓ Ground truth data set is a collection of variants generated by read simulation software and saved in **mutations.vcf** file
- ✓ **Concordance** is a state in which things agree and do not conflict with each other (Enc. Britannica)
- ✓ In our case concordance indicates similarity between variant call results

Let's use **vcf-compare** from **vcftools** suite, which is a good alternative to **bcftools**

Scan through **vcftools** manual - https://vcftools.github.io/man_latest.html

Read vcf-compare help

\$ **vcf-compare -h**

Calling variants

Calling variants with simulated reads using **BCFtools**

Determine **concordance** between “**ground truth**” data set and variants based in BWA and bowtie2 alignments

vcftools requires **tabix** index, tabix is a generic indexer for tab delimited files compressed with bgzip

```
$ tabix variants.bwa.vcf.gz
```

```
$ tabix variants.bowtie2.vcf.gz
```

```
$ bgzip output.mutations.vcf
```

```
$ tabix output.mutations.vcf.gz
```

```
$ mkdir vcf_comparison # Create vcf_comparison directory in sim_reads folder
```

```
$ cd vcf_comparison
```

```
$ vcf-compare ../variants.bwa.vcf.gz ../output.mutations.vcf.gz > bwa_vs_truth.txt # compare BWA derived variants with ground truth data set
```

```
$ cat bwa_vs_truth.txt
```

```
$ grep ^VN bwa_vs_truth.txt | cut -f 2- # grab data for Venn Diagram
```

Calling variants

Calling variants with simulated reads using **BCFtools**

Determine **concordance** between “**ground truth**” data set and variants based in BWA and bowtie2 alignments

```
$ vcf-compare ../variants.bowtie2.vcf.gz ../output.mutations.vcf.gz > bowtie2_vs_truth.txt # compare  
bowtie2 derived variants with ground truth data set  
$ cat bowtie2_vs_truth.txt
```

Compare the results, which aligner worked better? What are the conclusions we can draw from this analysis?

- ✓ Other ways to look at concordance between variants **bedtools jaccard distance** and **SnpSift**
- ✓ **bedtools** is a software suite that operates in genomic intervals
<https://bedtools.readthedocs.io/en/latest/>

Calling variants

Calling variants with simulated reads using **BCFtools**

Determine **concordance** between “**ground truth**” data set and variants based in BWA and bowtie2 alignments

Use bedtools to calculate **Jaccard distance** between vcf files

\$ `bedtools jaccard -a ../variants.bwa.vcf.gz -b ../output.mutations.vcf.gz` # prints out Jaccard distance and some other data, higher Jaccard distance indicates higher similarity between interval sets

\$ `bedtools jaccard -a ../variants.bowtie2.vcf.gz -b ../output.mutations.vcf.gz`

Explanation of Jaccard distance

Read the docs <https://bedtools.readthedocs.io/en/latest/content/tools/jaccard.html>

Command syntax

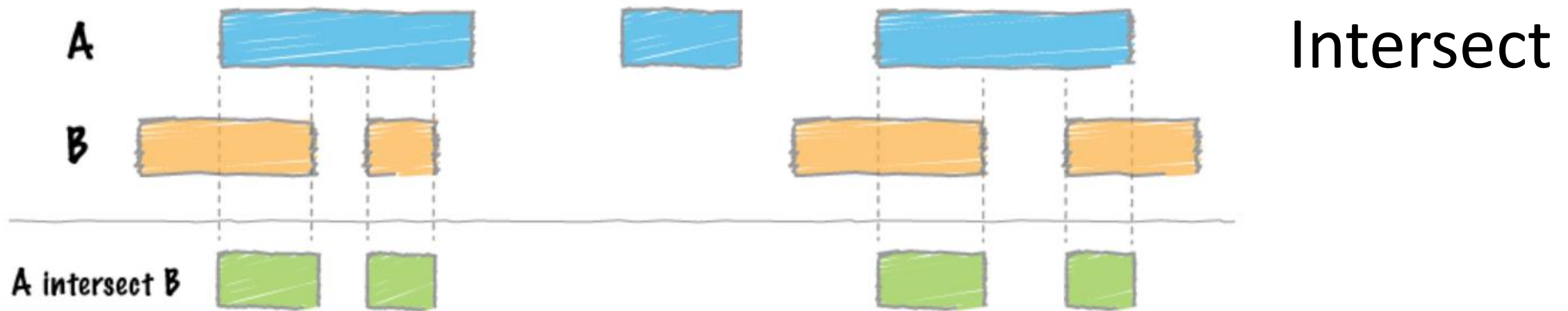
`bedtools jaccard [options] -a <BED,VCF,GFF> -b < BED,VCF,GFF >`

Calling variants

Calling variants with simulated reads using **BCFtools**

Determine **concordance** between “**ground truth**” data set and variants based in BWA and bowtie2 alignments

Explanation of Jaccard distance (continued)



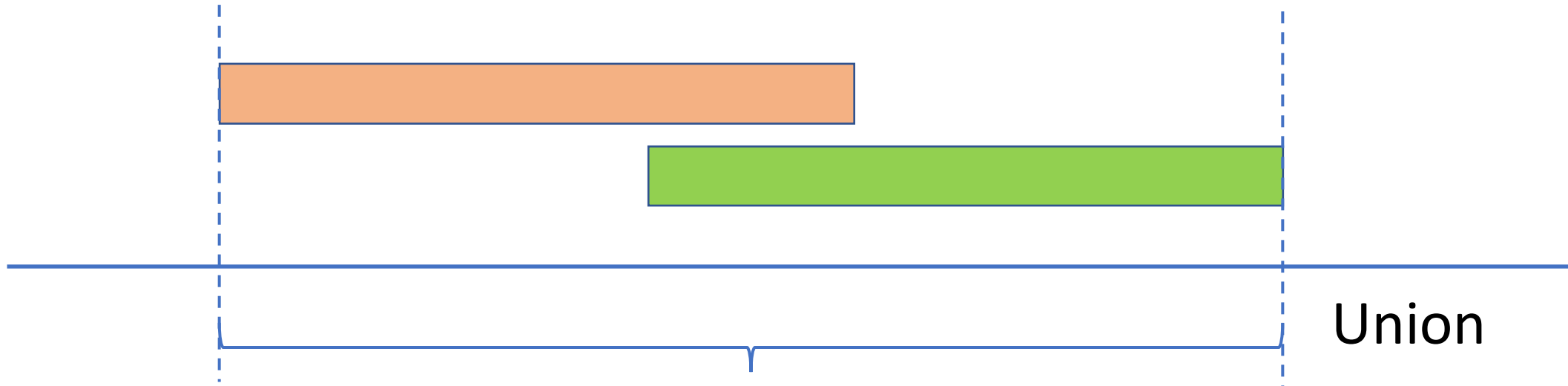
Example from bedtools manual: <https://bedtools.readthedocs.io/en/latest/content/tools/intersect.html>

Calling variants

Calling variants with simulated reads using **BCFtools**

Determine **concordance** between “**ground truth**” data set and variants based in BWA and bowtie2 alignments

Explanation of Jaccard distance (continued)



Jaccard distance = $\text{Intersect} / \text{Union}$

Bedtools Jaccard distance = $\text{Intersect} / \text{Union} - \text{Intersect}$

Calling variants

Calling variants with simulated reads using **Freebayes**

Freebayes link: <https://github.com/freebayes/freebayes>

\$ **freebayes --help** # Get familiar with the options

- **Freebayes** was designed to call SNPs, indels, multi-nucleotide polymorphisms, and composite insertion and substitution events that are shorter than the alignment length
- Freebayes uses statistics that relies on frequently observed haplotypes to call variants, it is expected that haplotypes that result from sequencing errors are rare

		Variant Region		Variant Region		
Reads	Ref	TACCGAT	CATTGGATCA	CGATTCC...GCATTGC	AAAAAAA-	GACCGCA
	TACCGAT	CATTGGATCA	CGATTCC...GCATTGC	-AAAAAA-	GACCGCA	
	ACCGAT	TATTGCATCG	CGATTCC...GCATTGC	-AAAAAA-	GACCGCA	
	ACCGAT	CATTGGATCA	CGATTCC...GCATTGC	AAAAAA-A	GACCGCA	
	ACCGAT	TATTGGATCG	CGATTCC...GCATTGC	-AAAAAAA	GACCGCA	
	CCGAT	C- TTGGATCA	CGATTCC...GCATTGC	AAAAAAA-	GACCGCA	
	CCGAT	CAT G GGATCA	CGATTCC...GCATTGC	AAAAAAA A	GACCGCA	
		• • •	• • •	• • •	• • •	
Observed Haplotypes	CATTGGATCA	x8	(A) ₇	x10		
	TATTGGATCG	x9	(A) ₆	x7		
	CTTGGATCA	x1	(A) ₅	x1		
	CAT G GGATCA	x1	(A) ₈	x1		
	• • •		• • •			

Calling variants

Calling variants with simulated reads using **Freebayes**

```
$ freebayes -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -p 1 simulated.bwa.sorted.bam > freebayes.bwa.vcf # -p 1 – set ploidy to 1
```

Previous command may take a long time, let's restrict variant calling to small region of the genome

```
$ freebayes -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -p 1 -r NC_000962.3:20000-80000 simulated.bwa.sorted.bam > freebayes.bwa.vcf # -r chrQ:start-end will restrict variant calling to a specified region
```

NOTE: Freebayes requires bam file with alignment quality encoded on Phred33 scale

Print the summary of alignments

```
$ bcftools stats freebayes.bwa.vcf > freebayes.bwa.stats
```

How many variant sites are in common between freebayes call and “ground truth” data

```
$ bgzip freebayes.bwa.vcf # we need to bgzip files for bcftools isec to work
```

```
$ bgzip output.mutations.vcf
```


Calling variants

Calling variants with simulated reads using *Freebayes*

Index compressed VCF files

```
$ bcftools index freebayes.bwa.vcf.gz
```

```
$ bcftools index output.mutations.vcf.gz
```

```
$ bcftools isec -r NC_000962.3:20000-80000 -p freebayes_vs_truth freebayes.bwa.vcf.gz  
output.mutations.vcf.gz # we will restrict the comparison to the region of freebayes call
```

```
$ cd freebayes_vs_truth
```

```
$ ls -lh
```

```
$ cat README.txt
```

Check how many variants are in common between freebayes and simulated mutations data set

```
$ bcftools stats 0003.vcf | less # 2358 sites are in common – “true positive” calls
```

```
$ bcftools stats 0000.vcf | less # 1608 sites called by freebayes are not on “ground truth” data, they are  
“false positives”
```

```
$ bcftools stats 0001.vcf # 3628 of mutation sites were simulated, but not called by freebayes, these are  
“false negatives”
```

Calling variants

Calling variants with simulated reads using **Freebayes**

Calculate Jaccard distance between freebayes calls and simulated mutations

Let's extract variants from a specific region and save it to a separate file

```
$ bcftools view -r NC_000962.3:20000-80000 freebayes.bwa.vcf.gz > freebayes.sub.vcf
```

Now we need to extract variants from the same regions in simulated data set

```
$ bcftools view -r NC_000962.3:20000-80000 output.mutations.vcf.gz > simulated.sub.vcf
```

Use **bedtools** to calculate Jaccard distance

```
$ bedtools jaccard -a freebayes.sub.vcf -b simulated.sub.vcf
```

Variant normalization

SNPs involve a single nucleotide and can be represented in a single way, i.e. **A → T**

Indels, multi-nucleotide polymorphisms, and other structural variants can be represented in different ways:


- GAC → AC
 - TGAC → AC
 - GACC → ACC
-
- ✓ Researchers realized that there is need to develop a unified representation of variants in order to facilitate comparison and integration of variant calls
 - ✓ The process of bringing the variants to unified representation is called “variant normalization”

Variant normalization

Variant normalization was originally described in the following paper:

JOURNAL ARTICLE

Unified representation of genetic variants

Adrian Tan, Gonçalo R. Abecasis, Hyun Min Kang  [Author Notes](#)

Bioinformatics, Volume 31, Issue 13, July 2015, Pages 2202–2204,

<https://doi.org/10.1093/bioinformatics/btv112>

Published: 19 February 2015 **Article history** ▼

Variant normalization has two conditions – the variant must be **parsimonious** and **left aligned**

Parsimony means representing a variant with as few nucleotides as possible without reducing the length of any allele to 0

Left alignment means shifting the start position of that variant to the left until it is impossible to do so

Variant normalization

- ✓ Visual explanation of parsimonious representation in the context of multi-nucleotide variant
- ✓ The example is found here: https://genome.sph.umich.edu/wiki/Variant_Normalization

Reference and alternative alleles of a multi nucleotide polymorphism (MNP)

REF
ALT

GGGCATGGG
GGG**TGCG**GGG

Genome Reference

GGGGCATGGGG

REF

GCAT

ALT

GTGC

REF

CATG

ALT

TGCG

REF

GCATG

ALT

GTGCG

REF

CAT

ALT

TGC

Variant Call Format

POS

REF

ALT

4

GCAT

GTGC

5

CATG

TGCG

4

GCATG

GTGCG

5

CAT

TGC

Not left trimmed

Not right trimmed

Not left and right trimmed

Normalized

Alleles represented against the human genome reference. Allele pairs are colored the same, all are representations of the same variant.

Alleles represented in Variant Call Format, all are representations of the same variant.

Variant normalization

- ✓ Visual explanation of parsimonious representation and left alignment in the context of a deletion

Reference and alternative alleles of a CA short tandem repeat (STR)

REF
ALT

GGGCACACACAGGG
GGGCACACAGGG

← CA deletion from the reference

Genome Reference		Variant Call Format			
GGGCACACACAGGG		POS	REF	ALT	
REF	CA	8	CA	.	Not left aligned and alternate allele is empty
ALT	.				
REF	CAC	6	CAC	C	Not left aligned but parsimonious
ALT	C				
REF	GCACA	3	GCACA	GCA	Not right trimmed
ALT	GCA				
REF	GGCA	2	GGCA	GG	Not left trimmed
ALT	GG				
REF	GCA	3	GCA	G	Normalized (left aligned & parsimonious)
ALT	G				
Alleles represented against the human genome reference. Allele pairs are colored the same, all are representations of the same variant.		Alleles represented in Variant Call Format, all are representations of the same variant.			

Variant normalization

Normalization of variants with **bcftools norm**

Variant calls derived from different software must be normalized in order to compare and integrate them

Some variant callers, such as GATK already produces normalized variant calls

Consult a variant caller's manual understand its behavior regarding variant normalization

Print **bcftools norm** help and try to understand the options

```
$ bcftools norm -h
```

Normalize freebayes calls

```
$ bcftools norm -O v -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -d all freebayes.sub.vcf -o  
freebayes.norm.vcf # -O v – output is an uncompressed VCF file; -d – removed duplicate variants (SNPs  
and indels)
```

```
$ bcftools norm -O v -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -d all simulated.sub.vcf -o  
simulated.norm.vcf
```

Variant normalization

Normalization of variants with **bcftools norm**

Now we can do a proper comparison using normalized file

```
$ bgzip freebayes.norm.vcf # bgzip both files
```

```
$ bgzip simulated.norm.vcf
```

```
$ bcftools index freebayes.norm.vcf.gz # index zipped files with bcftools
```

```
$ bcftools index simulated.norm.vcf.gz
```

```
$ bcftools isec -p comp_norm freebayes.norm.vcf.gz simulated.norm.vcf.gz # compare with isec
```

NOTE: I would recommend to always normalize VCF files making them comparable between projects

Additional reading about variant normalization: <http://www.cureffi.org/2014/04/24/converting-genetic-variants-to-their-minimal-representation/>

Variant normalization

Multi-sample variant calling

It is possible to call variants in multiple samples at the same time

There are two ways to call variants in more than one sample:

1. Use multiple BAM as input into the variant caller
2. Call variants on individual BAM and then merge resulting VCF files into a single file (tool: **bcftools merge**)

Likely it is better to create multi-sample files using method 1 if possible.

Merging VCF files may be problematic especially when different versions of the variant caller were used

Variant normalization

Multi-sample variant calling

Let's create a multi-sample VCF file

First, we will generate 3 simulated read files with different coverage

Create a new directory **multi/** in the **sandbox/** folder and change into it

```
$ mkdir multi
```

```
$ cd multi
```

Simulate 3 samples with different coverages and error rates, all reads will be single-end

Sample	Coverage	Error rate
Sample_1	3	0.00001
Sample_2	10	0.0001
Sample_3	25	0.003

Multi-sample analysis

Multi-sample variant calling

Simulate 3 samples with **dwgsim** from **M. tuberculosis** genomes, the reads will be single-end, 100 bp
\$ dwgsim -H -C 3 -1 100 -2 0 -e 0.00001 ../GENOMES/M_tuber/M_tuber_H37Rv.fasta sample_1 # -H – simulate in haploid mode; -C 3 – mean coverage is 3; -1 100 – length of read 1 is 100 bp; -2 0 – length of read 2 is 0, the reads are single-end; -e 0.00001 – set the error rate

Sample 2 reads – error rate 0.0001 and with a mean coverage of 10

```
$ dwgsim -H -C 10 -1 100 -2 0 -e 0.0001 ../GENOMES/M_tuber/M_tuber_H37Rv.fasta sample_2
```

Sample 3 – error rate 0.003; mean coverage - 25

```
$ dwgsim -H -C 25 -1 100 -2 0 -e 0.003 ../GENOMES/M_tuber/M_tuber_H37Rv.fasta sample_3
```

```
$ ls -lh
```

Remove interleaved fastq files and read 2 files we don't need

```
$ rm *bfast* *read2.*
```

```
$ ls -lh
```

Multi-sample analysis

Multi-sample variant calling

Map the reads bwa

```
$ bwa mem -t 2 -R '@RG\tID:sample_1\tSM:sample_1'  
../GENOMES/M_tuber/bwa_index/M_tuber_H37Rv.fasta sample_1.bwa.read1.fastq.gz > sample_1.sam  
$ bwa mem -t 2 -R '@RG\tID:sample_2\tSM:sample_2'  
../GENOMES/M_tuber/bwa_index/M_tuber_H37Rv.fasta sample_2.bwa.read1.fastq.gz > sample_2.sam  
$ bwa mem -t 2 -R '@RG\tID:sample_3\tSM:sample_3'  
../GENOMES/M_tuber/bwa_index/M_tuber_H37Rv.fasta sample_3.bwa.read1.fastq.gz > sample_3.sam  
$ samtools view -H sample_1.sam | grep '@RG' # check that read groups are present
```

Sort and index SAM files

```
$ samtools view -@ 2 -b sample_1.sam > sample_1.bam  
$ samtools view -@ 2 -b sample_2.sam > sample_2.bam  
$ samtools view -@ 2 -b sample_3.sam > sample_3.bam  
  
$ samtools sort -@ 2 sample_1.bam -o sample_1.sorted.bam  
$ samtools sort -@ 2 sample_2.bam -o sample_2.sorted.bam  
$ samtools sort -@ 2 sample_3.bam -o sample_3.sorted.bam
```

Multi-sample analysis

Multi-sample variant calling

Sort and index SAM files [continued]

```
$ samtools index sample_1.sorted.bam
```

```
$ samtools index sample_2.sorted.bam
```

```
$ samtools index sample_3.sorted.bam
```

```
$ ls -lh # check the results
```

Remove SAM files, they take up space and contain the same information as BAM

```
$ rm *.sam
```

Remove unsorted BAM files

```
$ ls sample_[[:digit:]].bam # test regex pattern first
```

```
$ rm sample_[[:digit:]].bam # remove unsorted bam files according to the pattern match
```

Check the quality of the alignments using **bamqc qualimap**

```
$ bamqc qualimap bamqc -nt 2 -bam sample_1.sorted.bam -outdir sample_1_qualimap -outfile  
sample_1_qualimap.html -gff
```

```
../GENOMES/M_tuber/annotation/GCF_000195955.2_ASM19595v2_genomic.gff # -nt <NUM_THREADS>
```

Multi-sample analysis

Multi-sample variant calling

Check the quality of the alignments using ***qualimap bamqc***

```
$ qualimap bamqc -nt 2 -bam sample_2.sorted.bam -outdir sample_2_qualimap -outfile  
sample_2_qualimap.html -gff  
../GENOMES/M_tuber/annotation/GCF_000195955.2_ASM19595v2_genomic.gff # -nt <NUM_THREADS>
```

```
$ qualimap bamqc -nt 2 -bam sample_3.sorted.bam -outdir sample_3_qualimap -outfile  
sample_3_qualimap.html -gff  
../GENOMES/M_tuber/annotation/GCF_000195955.2_ASM19595v2_genomic.gff
```

Integrate quality control data into a single report using ***mutiqc***

```
$ mutiqc .
```

Multi-sample analysis

Multi-sample variant calling

Call the variants according to the recommendations in *freebayes* manual

- Add **readgroups** to bam files with ***bamaddrg*** (our alignments already have read groups added by ***bwa***)
- Mark duplicates with ***sambamba markdups***
- Run ***freebayes*** on alignment files and a multi-sample VCF, usually defaults should work, but you will need to adjust **--ploidy** and **--min-alternate-fraction** for your analysis case
- Filter the outputs based on variant quality and/or read depth or observation count
- Interpret the variants
- Possibly, repeat filtering based on variant interpretation

Multi-sample analysis

Multi-sample variant calling

The read groups are already present in our **bam file** and we can skip directly to marking the duplicates

Mark duplicates using *sambamba markdup*

```
$ sambamba markdup -h # Check the help file
```

```
$ sambamba markdup -t 2 sample_1.sorted.bam sample_1.sorted.dedup.bam # -t 2 – use 2 threads
```

```
$ sambamba markdup -t 2 sample_2.sorted.bam sample_2.sorted.dedup.bam
```

```
$ sambamba markdup -t 2 sample_3.sorted.bam sample_3.sorted.dedup.bam
```

Call variants with *freebayes using* deduplicated files

```
$ freebayes -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta --ploidy 1 --min-alternate-fraction 0
```

```
sample_1.sorted.dedup.bam sample_2.sorted.dedup.bam sample_3.sorted.dedup.bam > multi.vcf # --
```

```
ploidy 1 - set ploidy to 1; --min-alternate-fraction 0 – minimum frequency of the alternative allele is set to 0
```

Examine VCF statistics

```
$ bcftools stats multi.vcf | less
```


Multi-sample analysis

Multi-sample variant calling

Normalize the variants in the vcf files we had produced

```
$ bcftools norm -O v -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -d all multi.vcf -o multi.norm.vcf
```

Normalize simulated variants

```
$ bcftools norm -O v -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -d all sample_1.mutations.vcf -o sample_1.mutations.norm.vcf
```

```
$ bcftools norm -O v -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -d all sample_2.mutations.vcf -o sample_2.mutations.norm.vcf
```

```
$ bcftools norm -O v -f ../GENOMES/M_tuber/M_tuber_H37Rv.fasta -d all sample_3.mutations.vcf -o sample_3.mutations.norm.vcf
```

Bgzip normalized files

```
$ bgzip multi.norm.vcf
```

```
$ bgzip sample_1.mutations.norm.vcf
```

```
$ bgzip sample_2.mutations.norm.vcf
```

```
$ bgzip sample_3.mutations.norm.vcf
```

Multi-sample analysis

Multi-sample variant calling

Index bgzipped files with tabix

```
$ tabix multi.norm.vcf.gz
```

```
$ tabix sample_1.mutations.norm.vcf.gz
```

```
$ tabix sample_2.mutations.norm.vcf.gz
```

```
$ tabix sample_3.mutations.norm.vcf.gz
```

Once VCF files are indexed with tabix we can compare them using **vcf-compare** from vcftools

```
$ vcf-compare -s sample_1 multi.norm.vcf.gz sample_1.mutations.norm.vcf.gz # -s <sample_name> will  
select variants from the sample
```

```
$ vcf-compare -s sample_2 multi.norm.vcf.gz sample_2.mutations.norm.vcf.gz
```

```
$ vcf-compare -s sample_3 multi.norm.vcf.gz sample_3.mutations.norm.vcf.gz
```

Record the numbers of **True positives (TP)**, **False positives (FP)**, and **False negatives (FN)** for these comparisons.

How much of an effect increasing coverage from 3 to 25 has on the results?

Multi-sample analysis

Multi-sample variant calling

Benchmarks to evaluate variant calling workflows

Precision – ratio of correctly called variants to all variants detected

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Sensitivity (Recall) – ratio of correctly called variants to all variants in “ground truth” dataset

$$\text{Sensitivity} = \text{TP} / (\text{TP} + \text{FN})$$

F1 score (F1 measure)

F1 score – harmonious means of precision and sensitivity

$$\text{F1 Score} = 2 * (\text{Sensitivity} * \text{Precision}) / (\text{Sensitivity} + \text{Precision})$$

F1 score ranges between 0 and 1 , with 0 being the worst possible score and 1 is the best meaning that the all events were predicted correctly.

Rules of thumb for F1 scores > **0.9** – excellent; **0.8 – 0.9** – good; **0.5 – 0.8** – OK; < **0.5** - poor

Filtering variants

- ✓ After a caller software is used to detect variants, we always need to filter in order to exclude low quality calls
- ✓ Filtering criteria differ depending on the properties of the dataset, but frequently it includes selecting the variant calls based on variant quality and reads depth
- ✓ Please, do not try to filter variants using *ad hoc* scripts and commands, use specialized software
- ✓ Popular and well-established filtering software includes:
 - **bcftools** - <https://samtools.github.io/bcftools/bcftools.html>
 - **vcftools** - https://vcftools.sourceforge.net/man_latest.html
 - **Snpsift** - https://pcingola.github.io/SnpEff/ss_introduction/
 - **rtg tools** - <https://github.com/RealTimeGenomics/rtg-tools>
 - **vcflib** - <https://github.com/vcflib/vcflib>

Filtering variants

Practice variant filtering using **vcftools**

Read **vcftools** manual to understand its options and capabilities:

https://vcftools.sourceforge.net/man_latest.html

We will extract and filter sample_3 from multi VCF file, this file has a mean coverage of 25

```
$ bcftools view -s sample_3 -O v multi.norm.vcf.gz > sample_3.norm.vcf
```

Did we extract a right sample? Sample_3 is supposed to have a mean coverage close to 25

`vcftools --vcf sample_3.norm.vcf --depth && cat out.idpth` # the first part of the command will create a file with **.idpth** extension, and the second part after ' && ' will open it

```
$ vcftools --vcf sample_3.norm.vcf --minQ 30 --recode --recode-INFO-all --out sample_3.Q30 # --gzvcf – the input file was compressed; --minQ 30 – keep variants with quality over 30; --recode – will create a new filtered VCF file and a log file with “recode” suffix; --recode-INFO-all – keep INFO fields from the original file, they are removed by default
```

```
$ bcftools stats sample_3.Q30.recode.vcf | less # check the resulting file
```

Filtering variants

Practice variant filtering using ***vcftools***

Compare filtered file to the “ground truth” data set

`vcf-compare -s sample_3 multi.norm.vcf.gz sample_3.mutations.norm.vcf.gz` # Compare original VCF file first

```
$ bgzip sample_3.Q30.recode.vcf
```

```
$ tabix sample_3.Q30.recode.vcf.gz
```

```
$ vcf-compare sample_3.Q30.recode.vcf.gz sample_3.mutations.norm.vcf.gz
```

Filtering based on quality > 30 improved the results, now are detecting 98% of true variants, however we are still generating about 60% of false positives

Filtering variants

Practice variant filtering using ***vcftools***

Let's calculate an F-score:

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Sensitivity} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{F1 Score} = 2 * (\text{Sensitivity} * \text{Precision}) / (\text{Sensitivity} + \text{Precision})$$

We should get F1 ~ **0.56**, that's not bad, but not good enough!

Let's try to improve a filter by read depth to the filtering parameters

First let's visualize true positives and false positives using IGV in attempt to understand what separates them

Use ***bcftools isec*** to create separate files for true and false positives

Visualize variants

Practice visualizing variants

Use **bcftools isec** to create separate files for true and false positives

```
$ mkdir igv_vis
```

```
$ cd igv_vis # for convenience create igv_vis/ directory and change into it
```

```
$ cp ../sample_3.Q30.recode.vcf.gz* ../sample_3.mutations.norm.* . # copy the files will be working with
```

```
$ bcftools index sample_3.Q30.recode.vcf.gz # we will need to index this file with bcftools to enable isec comparison
```

```
$ bcftools isec sample_3.Q30.recode.vcf.gz sample_3.mutations.norm.vcf.gz -p comparison # compare filtered file
```

```
$ ls -lh comparison
```

```
$ mv comparison/* .
```

```
$ mv comparison/* .
```

```
$ cat README.txt # which files do we need to use?
```

0000.vcf – false positives

0002.vcf or **0003.vcf** – true positives

Visualize variants

Practice visualizing variants

Bgzip comparison files

```
$ bgzip 0000.vcf
```

```
$ bgzip 0002.vcf
```

```
$ tabix 0000.vcf.gz
```

```
$ tabix 0002.vcf.gz
```

Fire up IGV

- ✓ Load M. tuberculosis genome
- ✓ Load sorted, indexed and deduplicated files
- ✓ Load 0000.vcf.gz – TRUE positives
- ✓ Load 0002.vcf.gz – FALSE positives

There is something really odd about false positive variants!

Visualize variants

Practice visualizing variants

Let's verify our observation about allele counts

```
$ zcat 0000.vcf.gz | grep -v '^#' | head -n 3 # Print some lines from the body of the VCF file
```

```
NC_000962.3      336      .      C      A      345.443      .      AB=0;ABP=0,AC=0;AF=0.333333
```

How many variants in “false positives” file have allele count of 0?

```
$ zcat 0000.vcf.gz | grep -v '^#' | grep ';AC=0;' | wc -l
```

How many variant in “false positives” file have allele count of 1?

```
$ zcat 0000.vcf.gz | grep -v '^#' | grep ';AC=1;' | wc -l
```

Now count variants with allele count 0 or 1 in “true positive” file

```
$ zcat 0002.vcf.gz | grep -v '^#' | grep ';AC=0;' | wc -l
```

```
$ zcat 0002.vcf.gz | grep -v '^#' | grep ';AC=1;' | wc -l
```

Visualize variants

Practice visualizing variants

Filter out the variants with **AC=0** and re-calculate F1 score

```
$ cp ../sample_3.norm.vcf . # copy the original file to our working directory
```

Keep only the variants with allele count 1

```
$ bcftools view --min-ac 1 -O v sample_3.norm.vcf > sample_3.AC1.vcf
```

Compress and index with tabix

```
$ bgzip sample_3.AC1.vcf
```

```
$ tabix sample_3.AC1.vcf.gz
```

Compare AC filtered VCF and “ground truth” data set

```
$ vcf-compare sample_3.AC1.vcf.gz sample_3.mutations.norm.vcf.gz
```

Visualize variants

Practice visualizing variants

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Sensitivity} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{F1 Score} = 2 * (\text{Sensitivity} * \text{Precision}) / (\text{Sensitivity} + \text{Precision})$$

$$\text{F1 Score} \sim 0.984$$

Variant annotation

Predict biological effects of detected variants

- Variant annotation means predicting the consequences of genetic changes on the function of genes.
- Changes in the sequence of the gene itself or regulatory regions may alter rate of transcription, change the structure of transcripts, change the stability of transcripts, and change the stability and structure of proteins. This list is not exhaustive.
- Genetic changes may be silent.
- The software that predicts variant effects is called a variant annotator
- Variant annotators can rely both on mathematical models and on external data to predict variant effects
- Normally, they take VCF file as an input and output an updated VCF files annotations together with more detailed html or pdf reports

Variant annotation

Predict biological effects of detected variants

What kind of variant annotations we can generate?

- Relate variants to genomic regions (coding or non-coding regions, regulatory elements, transcription start or termination sites, etc.)
- List which genes or transcripts are affected by the variant
- Determine the effect of the variant on protein sequence: synonymous mutations, missense mutation, stop-gained, stop-lost, frameshift
- Match detected variants with known variants from an external data set, like snpDB, Clinvar, ExAc or custom data sets

Variant annotation

Predict biological effects of detected variants

Variant annotation software

- ✓ snpEff : <https://pcingola.github.io/SnpEff/>
- ✓ Variant Effect Predictor : <https://useast.ensembl.org/info/docs/tools/vep/script/index.html>
- ✓ AnnoVar : <https://annovar.openbioinformatics.org/en/latest/>
- ✓ Vaast2 : <https://www.yandell-lab.org/software/vaast.html>

Here we will use ***SnpEff*** to annotate the final set of variants detected with the use of ***BWA – freebayes*** workflow

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with *snpEff*

Familiarize yourself with *snpEff* manual : https://pcingola.github.io/SnpEff/se_introduction/

Create a new directory **annotation/** in the **sandbox/** folder

```
$ mkdir annotation
```

```
$ cd annotation
```

List prebuilt annotation data sets that come with *snpEff*

```
$ snpEff databases > snpeff_builds.txt
```

```
$ wc -l snpeff_builds.txt # how many organisms are available?
```

```
$ head snpeff_builds.txt
```

Search for Mycobacterium tuberculosis H37Rv

```
$ grep --color="auto" 'Mycobacterium_tuberculosis_h37' snpeff_builds.txt | head
```


Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with *snpEff*

Copy filtered VCF file to **annotation/** directory

```
$ cp ../multi/igv_vis/sample_3.AC1.vcf.gz .
```

```
$ gunzip sample_3.AC1.vcf.gz
```

Annotate the VCF file

```
$ snpEff -t Mycobacterium_tuberculosis_h37rv sample_3.AC1.vcf > sample_3.AC1.annot.vcf # -t option
```

instruct the command to run on multiple threads

Take a look at the body of the resulting VCF file

```
$ bcftools view -H sample_3.AC1.annot.vcf | head
```

```
ANN=A| |MODIFIER| | | | | | | | | |ERROR_CHROMOS
```

SnpEff added an **ANN** tag to the INFO field on the VCF file, the entries in the ANN tag are separated with |

However, the ANN fields are empty since the chromosome names in the database in in the VCF file did not match!

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with *snpEff*

- Annotation data sets used by snpEff were build based on bacterial genomes from Ensembl <https://bacteria.ensembl.org/index.html>
- We downloaded our reference sequence from NCBI genomes, as the result, the sequence id between snpEff database and our genome does not match
- We must use the same genome build from the same source throughout the project
- Luckily snpEff allows us to build custom databases from using the genomes sequence and gene annotation file

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with *snpEff*

Build a custom database for the reference genome

Create a directory called **data/** in the **annotation/** folder

```
$ mkdir data
```

In **data/** create a subfolder names after the accession number of our reference sequence

```
$ mkdir -p data/NC_000962.3
```

Copy the reference genome and gene annotation file to **data/NC_000962.3/** directory

```
$ cp ../GENOMES/M_tuber/M_tuber_H37Rv.fasta data/NC_000962.3/
```

```
$ cp ../GENOMES/M_tuber/annotation/GCF_000195955.2_ASM19595v2_genomic.gff  
data/NC_000962.3/
```

Rename these files to **sequences.fa** and **genes.gff** correspondignly and build snpEff database

```
$ snpEff build -gff3 -v NC_000962.3
```

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with ***snpEff***

snpEff build command will create snpEffectPredictor.bin file in the database directory

```
$ ls -lh data/NC_000962.3/
```

Now we can annotate the filtered VCF file

```
$ snpEff -t -canon NC_000962.3 sample_3.AC1.vcf > sample_3.AC1.annot.vcf # -canon option forces the use of only canonical transcripts
```

snpEff will add an **ANN** tag to the INFO field that contains information about predicted variant effects

The ANN format specifications are described here:

https://pcingola.github.io/SnpEff/se_inputoutput/

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with ***SnpEff***

Filtered variants based on annotation using ***SnpSift***

Install ***SnpSift*** and go through the manual: https://pcingola.github.io/SnpEff/ss_introduction/

Look through ***SnpSift*** help file to become familiar with the syntax and options

```
$ SnpSift -h
```

SnpSift is a universal variant filtering software with added ability to filter based on ANN fields

For example, we can filter VCF file by quality

```
$ cat sample_3.AC1.annot.vcf | SnpSift filter "( QUAL >= 400 )" | grep -v '^#' | head
```

Filter by multiple fields, for example QUAL and DP

```
$ cat sample_3.AC1.annot.vcf | SnpSift filter "( QUAL >= 400 && DP > 25 )" | grep -v '^#' | head
```

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with ***SnpEff***

Filtered variants based on annotation using ***SnpSift***

Extract variants based on genomic intervals

```
$ cat sample_3.AC1.annot.vcf | SnpSift filter "( POS > 10000) & (POS < 20000 )" | grep -v '^#' | head
```

Filter by reference and alternative alleles

```
$ cat sample_3.AC1.annot.vcf | SnpSift filter "( REF = 'A' && ALT = 'C') " | grep -v '^#' | head # Get A/C variants
```

Keep deletions with quality over 500

```
$ cat sample_3.AC1.annot.vcf | SnpSift filter "(TYPE = 'del' && QUAL > '500') " | grep -v '^#' | head
```

Keep SNPs only

```
$ cat sample_3.AC1.annot.vcf | SnpSift filter "(TYPE = 'snp') " | grep -v '^#' | head
```

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with ***SnpEff***

Filtered variants based on annotation using ***SnpSift***

Specification for ANN format: https://pcingola.github.io/SnpEff/adds/VCFannotationformat_v1.0.pdf

We can filter by ANN fields, but we must remember about some specific properties of this format

For example, we could try to select **missense** variants

```
$ SnpSift filter "ANN[0].EFFECT = 'missense_variant'" sample_3.AC1.annot.vcf
```

However, there can be more than one effect for the same variant separated by &, for example

missense_variant&splice_region_variant. In this case specifying '=' would mean that we will select missense variants, but not missense variants combined with splice region variants.

It makes sense to use 'has' instead of '=', the command below will find lines where any of the effects are missense variants

```
$ SnpSift filter "ANN[*].EFFECT has 'missense_variant'" sample_3.AC1.annot.vcf
```

Variant annotation

Predict biological effects of detected variants

Annotate the filtered set of variants with ***SnpEff***

Filtered variants based on annotation using ***SnpSift***

Extract missense variant from a specific gene

```
$ SnpSift filter "(ANN[*].EFFECT has 'missense_variant') && (ANN[*].GENE = 'pstP')"  
sample_3.AC1.annot.vcf
```

Extract **high** impact variants

```
$ SnpSift filter "(ANN[*].IMPACT = 'HIGH')" sample_3.AC1.annot.vcf
```


Complete workflow

Putting it all together

Let's plan a complete variant calling workflow based on what we've learned so far!

1. Use **FastQC** to evaluate the quality of sequencing reads
2. Trim adapters if necessary (**Trimmomatic**, **TrimGalore!**); trimming low quality bases is probably unnecessary if you are using a variant caller that relies on haplotype data, like freebayes, see discussion here: <https://www.biostars.org/p/85400/>
3. Map the reads to the reference genome (**BWA**, **bowtie2**, **mosaic**, **etc**), if resulting file is in SAM format – convert to BAM. Sort and index BAM files.
4. Evaluate the quality of alignments (**qualimap**, **bamQC**)
5. Mark PCR duplicates with **sambamba** or **picard tools**, so they won't be used in variant calling
6. Call variants using **bcftools**, **freebayes**, **varscan2** or any other suitable variant caller

Complete workflow

Putting it all together (continued)

7. Filter the variants using **bcftools**, **vcftools**, **rtg** or other similar packages. Normally the variants are filtered based on variant quality (QUAL) and read depth (DP). Sometime filtering is fine-tuned by using allele balance (AB) and strand bias counts (SRF, SRR, SAF, SAR, SAP). In many cases, filtering strategies can become quite sophisticated.

9. Evaluate filtering against simulated data set, your experiment may call for higher precision or sensitivity depending on the application. Explore the variants together with alignments in genomic browser. Use visualizations, comparisons and analysis of VCF statistics to gain better understanding of the sources of false positives and false negatives.

10. Annotate the final set of variants (**SnpEff**, **Variant Effect Predictor**, **AnnoVar**)