Realization of variant calling from exome sequencing data

Author: Tomàs Montserrat Ayuso

August 29, 2022

Introduction

In this post we will show how we can perform variant calling from data from an exome sequencing experiment. Having one million random single-end reads of chromosome 22, we will discover the variants present and annotate those found in the "common all" subset of the *dbSNP database*.

The content we will present is the following:

- Basic quality control of the reads with *fastqc* and filtering of bases depending on their quality with *trimmomatic*.
- Alignment of the short sequences against chromosome 22 of the hg19 genome version with *BWA*.
- Removing duplicate sequences from the *bam* file using *Picard*.
- Running variant calling using *FreeBayes*.
- Filter the variants present in the APOBEC3H gene and annotate them.

Creation of directories and pipeline

Once we are in the working directory, it is convenient to create the folders where we will save the different files that will be generated during the analysis:

```
$ mkdir alignment data quality reference results scripts
```

In the *alignment* folder we will save all the *bam* files of the project. To *date* we will save the *fastq* files with the reads from the sequencing. We will use the quality directory to store the files with information about the quality of the data, both *fastq* and *bam* files. In the *reference* folder we will save all the files related to the reference genome, both *fasta* and those necessary to generate the index. In *results* we will save the outputs from the variant calling, that is, the different *vcf* files with the variants found. Finally, in the *scripts* folder we will save all the commands necessary to reproduce the analysis automatically.

The analysis pipeline is as follows:

- Carry out a quality control of the reads. If necessary, filter out low-quality bases and sequences.
- Align the reads with the reference genome to obtain a *bam* file with the aligned sequences.

- Carry out variant calling with a suitable program. Filter the variants found according to quality and/or any other attribute that interests us.
- Annotate the variants according to the biological effect they cause and identify those present in any database of our interest.

Quality control of the reads

As we did in the post on how to obtain a count matrix, the first step is to check that the quality of the sequences is adequate for our analysis. We will do it with the *fastqc* program, which requires us to specify the path to the *fastq* file and the directory where we want the results:

```
$ fastqc data/exomeSeq_chr22.fastq -o quality/
```

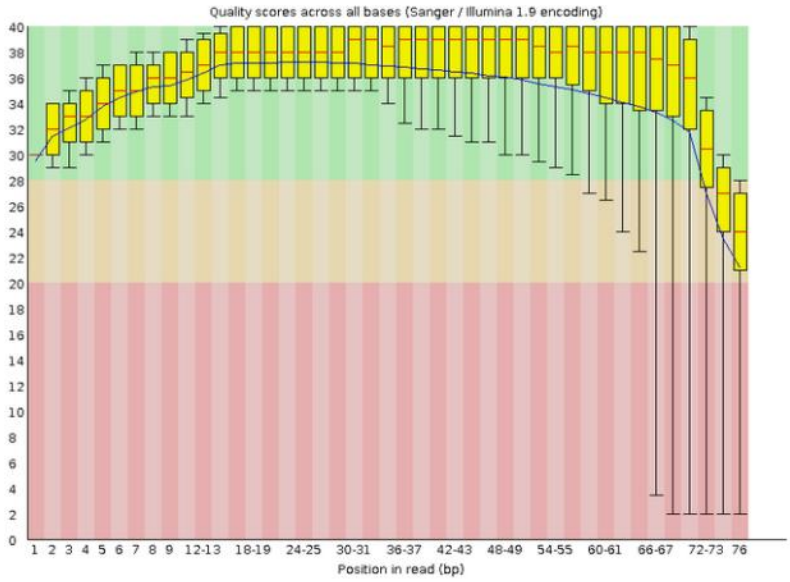
This program will generate a report in *html* format of the results obtained. As we did in the post mentioned before, we will focus on the most important aspects (a detailed explanation of each section can be seen on this webpage):

- Basic statistics:

Measure	Value
Filename	exomeSeq_chr22.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	1000000
Sequences flagged as poor quality	0
Sequence length	76
%GC	53

In this section we can find information such as the total number of reads, their length or the average GC content.

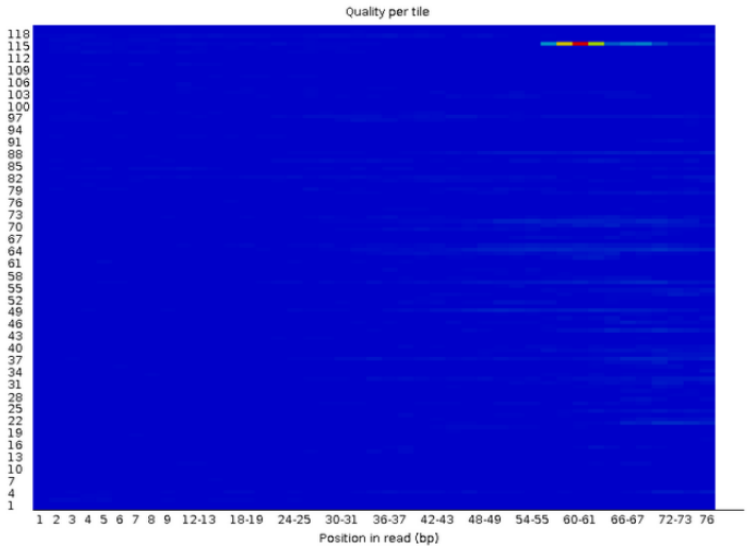
- Per base quality:



In this section of the report, we can study the distribution of the quality (confidence that the reported base is correct) of the sequencing of each base at each position of the sequence. The interpretation of the quality score of each base can be read in the post referenced above.

The data we have shows the typical pattern in which quality increases from the top positions to the middle positions and decreases at the bottom. However, we can see how the mean and median quality decreases greatly from position 66. So does the variability, with many more sequences showing low quality in the final positions.

The rest of the sections show no problems, except for the quality in one part of the flow cell, which has reported lower qualities:



Filtering of positions with low quality

In order to improve the average and median quality of the sequences and make it constant in all positions, we will eliminate the bases at the end of the reads that do not have a minimum Phred score of 15. We will also eliminate those sequences that, once trimmed, do not have a minimum length of 50 bases. We will do this using *trimmomatic*:

```
$ java -jar ${HOME}/trimmomatic/Trimmomatic-0.39/trimmomatic-0.39.jar SE \
> -threads 4 \
> data/exomeSeq_chr22.fastq data/exomeSeq_chr22_trimmed.fastq \
> TRAILING:15 \
> MINLEN:50 \
> -phred33
```

The different options we have specified are:

- `SE`: single-end reads.
- `threads 4`: number of cores to use.
- `data/exomeSeq_chr22.fastq`: *fastq* file containing the sequences.

- `data/exomeSeq_chr22_trimmed.fastq`: filtered file name.
- `TRAILING:15`: trim sequences by tail as long as base score is less than 15.
- `MINLEN:50`: remove sequences shorter than 50 base pairs.
- `phred33`: base quality coding system.

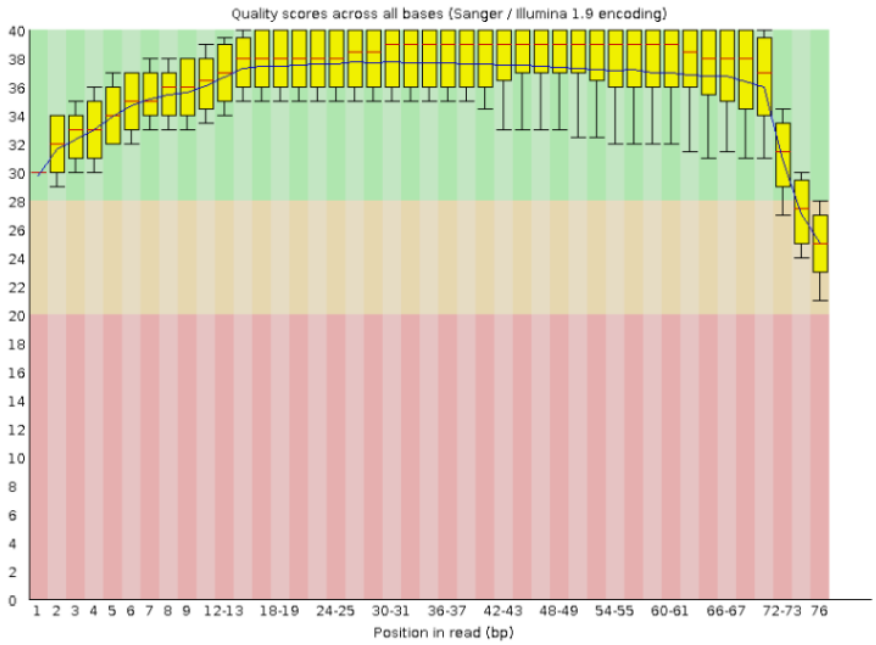
Quality control of post-trimmed reads

After performing the trimming, it is advisable to do the quality control again to make sure that the reads with which we will continue the analysis have the desired quality:

```
$ fastqc data/exomeSeq_chr22_trimmed.fastq -o quality/
```

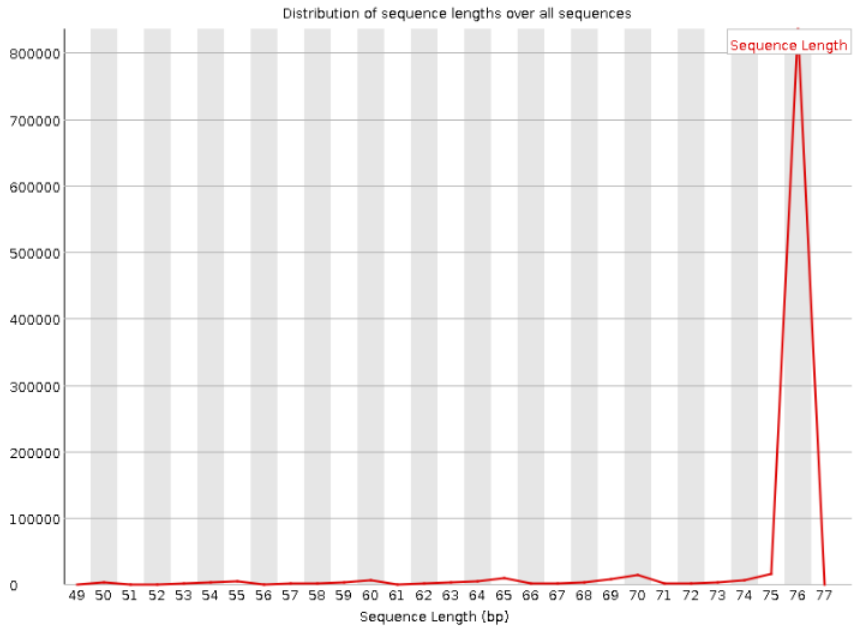
The result, as can be seen, is a significant improvement of the qualities in the last positions of the sequences. We can also see that its length is now between 50 and 76:

Measure	Value
Filename	exomeSeq_chr22_trimmed.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	958996
Sequences flagged as poor quality	0
Sequence length	50-76
%GC	52



In the same way, the number of sequences available to us has been slightly reduced: from the initial million reads, we now have 958,996.

An interesting aspect to consider after trimming is to see the distribution of the length of the sequences, as it is an attribute to consider when choosing the alignment algorithm against the reference genome. This information is also found in the report generated by *fastqc*:



The vast majority of reads are still 76 nucleotides long.

Alignment of the sequences with the reference genome

Once we are satisfied with the quality of our sequences, we can now align them to the reference genome. Since we know the reads are from chromosome 22, we will align them only against it. Specifically, we will use the sequence of chromosome 22 from the hg19 version of the human genome. We can do it using the following command:

```
$ wget 'ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr22.fa.gz' \
> -O chr22.fa.gz
```

After downloading the file, we need to unzip it. We can do it with the following command:

```
$ gunzip chr22.fa.gz
```

To align the reads we will use the *BWA* program, specifically the *BWA-MEM* algorithm, which is optimized for short sequences of between 70 and 100 nucleotides. In our case, the length of the vast majority of reads is 76. Before proceeding with the alignment, we need to create an index of our reference genome. We will do this with the following command:

```
$ bwa index -p chr22 chr22.fa
```

The `-p` option indicates the prefix that all generated files will have.

Finally, we can now align the short sequences. The output of an alignment is a *sam* (or *bam*) file which contains the original sequences and information about their alignment. We will redirect the output to *samtools* in order to sort the sequences based on their alignment coordinates and generate a *bam* file:

```
$ bwa mem -t 4 \
> reference/chr22 \
> data/exomeSeq_chr22_trimmed.fastq \
> | samtools sort -o alignment/exomeSeq_chr22.bam
```

Marking of duplicate sequences

The next step in the pipeline is to mark the reads that align in the same position, that is, those that have the same initial and final coordinates. These sequences are usually the result of duplications of the same cDNA fragment during sequencing and should be marked so as not to be taken into account during variant calling as they could cause false positives.

For this task we will use *MarkDuplicates* program from *Picard* suite:

```
$ java -jar ${HOME}/picard/picard.jar MarkDuplicates \
> INPUT=alignment/exomeSeq_chr22.bam \
> OUTPUT=alignment/exomeSeq_chr22_marked.bam \
> METRICS_FILE=metrics.txt \
> ASSUME_SORTED=true
```

The program generates a new *bam* file with duplicate reads marked and a text file with numerical information from the *bam* file:

```
$ grep -v "#" metrics.txt | cut -f 2,5,6,9
> UNPAIRED_READS_EXAMINED UNMAPPED_READS UNPAIRED_READ_DUPLICATES PERCENT_DUPLICATION
943184 15812 291896 0.309479
```

Checkpoint of the project: statistics of the reads and visualization of the alignment

We are now ready to carry out the variant calling with suitable software. However, at this point it is worth checking that everything has worked as we expected. We will use *samtools stats* to extract basic statistics of some properties of our reads and visualize the alignment in *IGV*. Let's start by generating the basic statistics:

$ samtools stats alignment/exomeSeq_chr22_marked.bam > quality/bam_file_stats.txt

The program generates statistics of all the reads. Now, we are interested in the first section, where the general statistics of the *bam* file appear:

```
$ grep -v "#" bam_file_stats.txt | grep -v "CHK" | head -n 26
SN raw total sequences: 958996
SN filtered sequences: 0
SN sequences: 958996
SN is sorted: 1
SN 1st fragments: 958996
SN last fragments: 0
SN reads mapped: 943184
SN reads unmapped: 15812
SN reads QC failed: 0
SN non-primary alignments: 0
SN bases trimmed: 0
SN bases duplicated: 21699486
SN average length: 74
SN average first fragment length: 75
SN average last fragment length: 0
SN maximum length: 76
SN maximum first fragment length: 76
SN maximum last fragment length: 0
SN average quality: 36.3
SN insert size average: 0.0
SN insert size standard deviation: 0.0
SN inward oriented pairs: 0
SN outward oriented pairs: 0
SN pairs with other orientation: 0
SN pairs on different chromosomes: 0
SN percentage of properly paired reads (%): 0.0
```

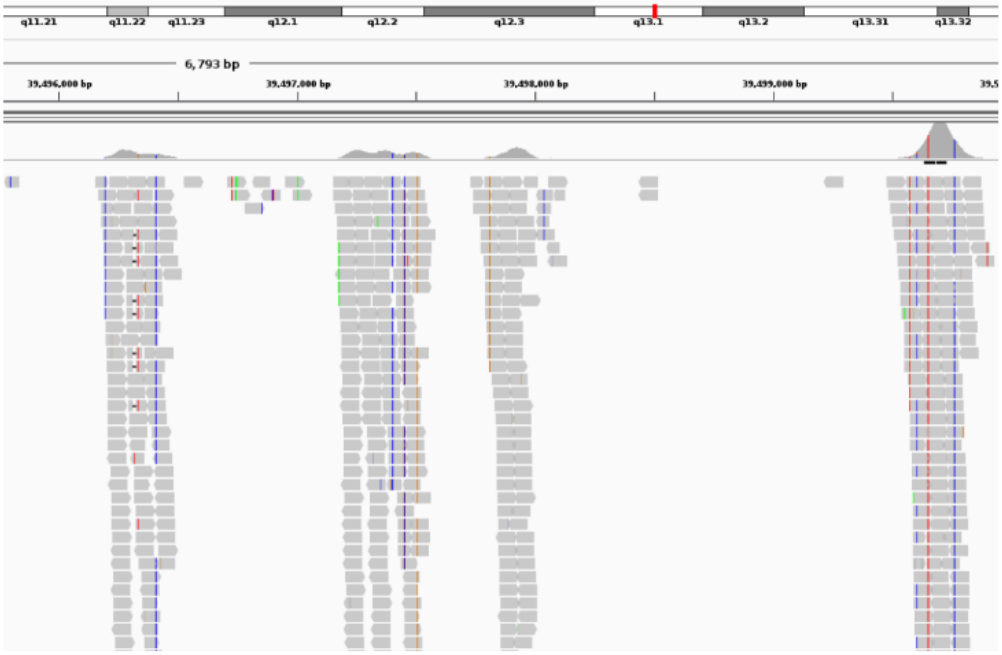
We can see the total number of sequences, how many could not be aligned, the number of reads marked as duplicates or the average quality (Phred score) of these, among other characteristics.

Viewing the alignment allows us to check that everything went well: since the data comes from exome sequencing, the exons are the regions where the most aligned reads should accumulate, although we also expect, according to the literature, that the border regions of exons and introns have high coverage. In addition, we will take the opportunity to see the differences between the alignment keeping duplicate reads and excluding them.

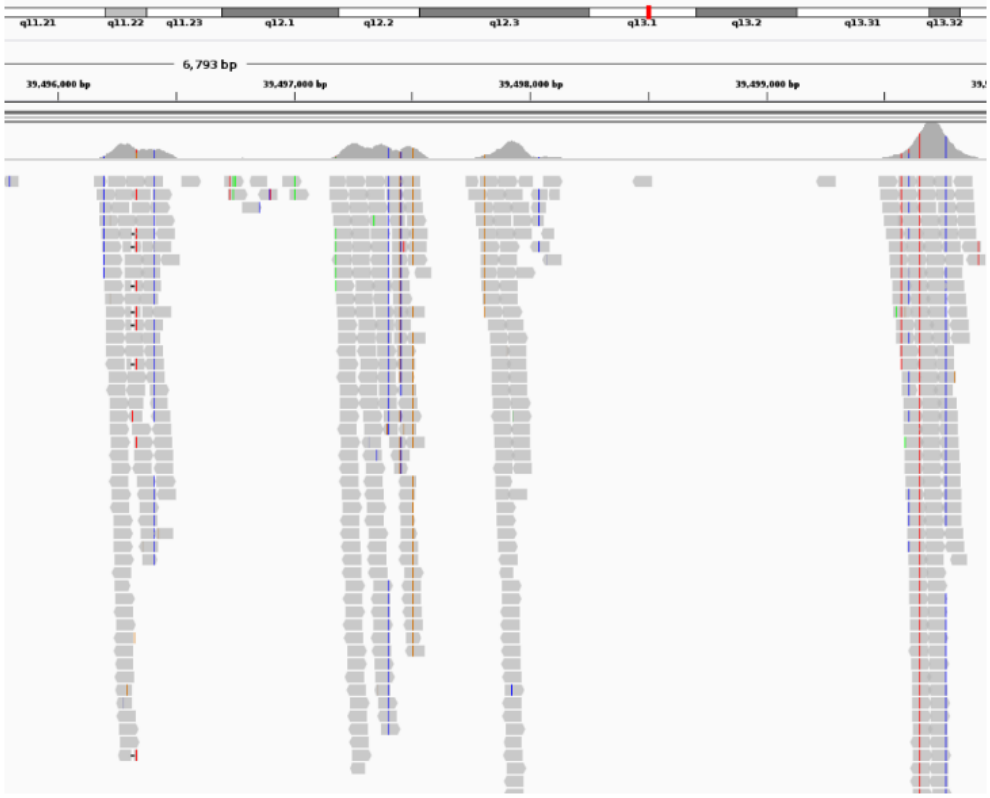
In order to upload *bam* files into *IGV*, we need to create an index of them. We can do this with *samtools index*:

```
$ samtools index -b alignment/exomeSeq_chr22.bam
$ samtools index -b alignment/exomeSeq_chr22_marked.bam
```

If we go to an arbitrary region of chromosome 22, we can observe the alignment without excluding duplicates:



And excluding them:



A significant reduction in the number of reads aligned in some of the positions is observed.

Search for differences between the aligned sequences and the reference genome (variant calling)

We will use *FreeBayes* to detect the small variants (SNVs and indels) of the problem sample with respect to the hg19 version of chromosome 22 of the human genome. This software looks for significant differences, using Bayesian statistical procedures, between the aligned sequences and the reference genome. It is a haplotype-based variant caller, which means that, instead of studying individual positions, it does so in windows of dynamically determined length. In addition, *FreeBayes* automatically performs realignment of insertions and deletions, necessary because reads aligned near indels sometimes align incorrectly during the first try.

We proceed to make the calling variant:

```
$ freebayes -f reference/chr22.fa \
> --min-coverage 7 alignment/exomeSeq_chr22_marked.bam > results/variants.vcf
```

The program receives as input a *bam* file and the reference genome. The `-f` option allows us to specify the file where the reference genome is located. The `--min-coverage` option indicates the minimum coverage required to process a locus.

The program determines the genotype of the individual for each position of the reference genome and returns a *vcf* file (Variant Call File) that includes, among others:

- The positions of the variants found.
- The change associated with these variants.
- The genotype of the sample for each variant.
- Measures of the reliability of variants and genotypes.

The full specification of this file type can be found on the *Samtools* suite web page.

By default, the program only considers variants supported by at least two observations and at least 20% of the reads at that position. We have also modified the number of observations, increasing the minimum coverage of that position to 7.

To each detected variant, *FreeBayes* has assigned a quality score, specifically a Phred score. Using *vcffilter* we will eliminate those variants that have a probability of having been erroneously found greater than 1%. This means removing those discovered variants with a quality score below 20:

```
$ java -jar ${HOME}/vcffilter/vcffilter-assembly-0.2.jar \
> -I results/variants.vcf \
> -o results/variants_filtered.vcf \
> --minQualScore 20
```

Variants in the APOBEC3H gene

At this point, we already have the variants discovered and we could proceed to annotate and study them. As an example, we will focus on the APOBEC3H gene. First, we will filter the discovered variants to keep those located in the coordinates of our gene. Then, we will predict the biological effect they cause, and we will annotate them based on the mutations collected in a database.

To filter the variants according to the position we will use *SnpSift*:

```
$ cat results/variants_filtered.vcf | \
> java -jar ${HOME}/snpEff/SnpSift.jar filter \
> "(CHROM = 'chr22') & (POS > 39493228) & (POS < 39500073)" \
> results/variants_apobec3h.vcf
```

This program takes as input a *vcf* file using the pipe operator and applies all the filters we want to it. In our case, it is on chromosome 22 (for illustrative purposes only, since we only studied variants on this chromosome) and between the positions where the gene is located. We redirected the output to a new *vcf* file.

Next, we will predict the biological effect (such as amino acid changes) that these variants can cause with the *SnpEff* program:

```
$ java -jar ${HOME}/snpEff/snpEff.jar -v hg19 \
> -stats results/snpEff_apobec3h.html \
> results/variants_apobec3h.vcf > results/variants_apobec3h_eff.vcf
```

The program needs to specify the version of the genome to be used (option `-v`) and generates a report of the results in *html* format. We redirected the output of the program to a new *vcf* file, which contains a new field with the annotations made by *SnpEff*.

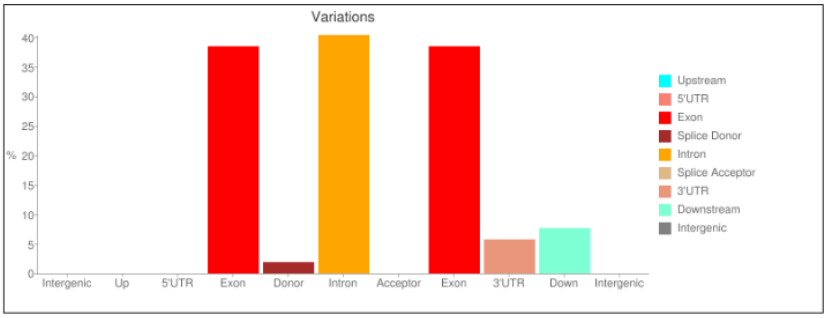
The report contains a summary of the biological effects that have been found, such as counts of the type of mutations (missense, synonymous...) or the region they affect (introns, exons...):

Type

Type (alphabetical order)	Count	Percent
3_prime_UTR_variant	3	5.357%
disruptive_inframe_deletion	4	7.143%
downstream_gene_variant	4	7.143%
intron_variant	25	44.643%
missense_variant	12	21.429%
splice_donor_variant	1	1.786%
splice_region_variant	3	5.357%
synonymous_variant	4	7.143%

Region

Type (alphabetical order)	Count	Percent
DOWNSTREAM	4	7.692%
EXON	20	38.462%
INTRON	21	40.385%
SPLICE_SITE_DONOR	1	1.923%
SPLICE_SITE_REGION	3	5.769%
UTR_3_PRIME	3	5.769%



The final step is to annotate the discovered variants found in the *dbSNP* database. Specifically, we will annotate those that are in the "common all" subset, which are those variants with a germinal origin and that the frequency of the allele is greater than 1%. The *vcf* file of the variants in the *dbSNP* database can be downloaded from this link.

In order to reduce the size of the database *vcf* file, we preprocessed it to work only with those variants found in the APOBEC3H gene. First, we filtered the file to keep only the variants of chromosome 22 and thus reduce its size:

```
$ grep ^# -v dbsnp.vcf | sed -e 's/^/chr/' | grep ^"chr22" > dbsnp_chr22.vcf
```

Then, with the following Python script, we were left with only the variants of the genomic region containing our gene:

```
import numpy as np
import pandas as pd

# Read VCF file
dbsnp = pd.read_table("dbsnp_chr22.vcf",
                      names=["chr", "pos", "id",
                             "ref", "alt", "qual",
                             "filter", "info"])

# Keep the variants near the genomic region that contains the gene APOBEC3H
filtered_dbsnp_hg19 = dbsnp[(dbsnp["pos"] > 39486385) & (dbsnp["pos"] < 39506916)]

# Export a new VCF file with the variants of interest
filtered_dbsnp_hg19.to_csv("filtered_dbsnp_chr22_hg19.vcf",
      sep="\t", header=False, index=False)
```

Now we can identify our variants present in this database. To do this we will use *SnpSift Annotate*:

```
$ java -jar ${HOME}/snpEff/SnpSift.jar annotate \
> data/filtered_dbsnp_chr22_hg19.vcf \
> results/variants_apobec3h_eff.vcf > results/variants_apobec3h_ann.vcf
```

This program takes as input our *vcf* file with the discovered variants, and the *vcf* file with the annotated variants from the database. We redirected the output to a new annotated *vcf* file. If we study the first fields of this file we can see which variants were in the database used and which were not:

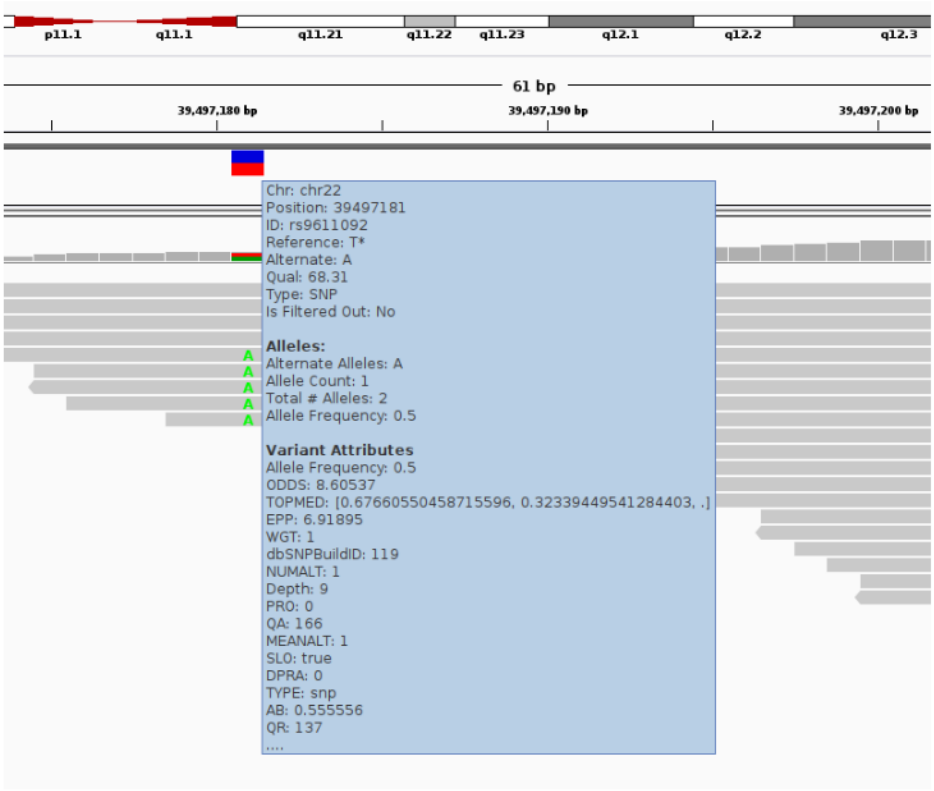
```
$ grep -v "#" results/variants_apobec3h_ann.vcf | cut -f 1-3
chr22 39496199 rs139291
chr22 39496322 .
chr22 39496336 rs139293
chr22 39496412 rs139294
chr22 39497181 rs9611092
```

```
chr22 39497404 rs139297
chr22 39497452 .
chr22 39497509 .
chr22 39497811 rs139301
chr22 39499575 rs139313
chr22 39499604 rs139314
chr22 39499653 rs139315
chr22 39499763 rs139316
```

Only three variants were not found in the database we used.

Finally, we can visualize this genomic region with the discovered variants and the alignment in *IGV*:

If we zoom in on one of the positions that contains a variant, we can study it in more detail. For example, we can see if the individual is homozygous or heterozygous for that variant or what the difference is with respect to the reference genome:



In this example we see that the individual is heterozygous for this variant. One of the alleles matches the reference one and has a thymine in this position, while the alternative has an adenine.

Conclusion

In this post we have seen how to carry out a variant calling from the reads of the sequencing of an exome. We aligned the short reads, performed quality control and finally performed variant calling using *FreeBayes*.

We have seen how we can filter the variants found according to their quality and location. Once we have found and filtered the variants, we focused on the APOBEC3H gene, annotating the variants by the biological effect they cause and identifying those present in a subset of the *dbSNP* database.

To finish, we have seen how we can visualize the variants in *IGV* and study in more depth those that interest us most for our project.

Once the analysis is done, it is convenient to save all the commands we used in a Bash script, creating a pipeline and automating the process in case you want to reproduce it on another occasion. The script with all the commands to replicate this analysis can be found in the following link.