Mapping and Variant Calling (June 2016)

Martin Jakt, Alexander Jueterbock[*]

PhD course: High throughput sequencing of non-model organisms

Contents

1 Mapping with Bowtie2 **2**

1.1 The mapping commands 2

1.2 Running the commands in a script for posterity 5

2 Filter mappings **6**

2.1 Removing duplicate reads 7

2.2 Re-alignment around indels 8

3 Visualizing alignments **9**

3.1 Samtools tview: command-line viewer 9

3.2 IGV: viewer with a graphical user interface 10

4 BONUS: SNP calling with samtools mpileup and bcftools **11**

Once you have assembled a draft genome, you can map your reads against it and identify potential read variants, like SNPs (Single Nucleotide Polymorphisms) and InDels (Insertions and Deletions). The library that you sequenced in this course is likely to represent a single diploid individual. Read variants, thus, occur at loci that differed between the mother and the father of this individual. Read variants become interesting when they are associated with certain environmental factors or phenotypes. This, however, requires sequencing several individuals of differing phenotypes or obtained from several environments.

[*]Nord University, Norway

1 Mapping with Bowtie2

Mapping next-generation sequencing data against genomes requires very high performance algorithms that must balance the accuracy of the mapping with the time taken. This is an interesting problem to work on and a rather large number of different applications have been developed. To map reads against the assembled draft genome, we will use Bowtie2. Bowtie2 defaults to finding global alignments (no soft-clipping of terminal bases) and it allows for gaps in the alignment, thereby increasing mapping accurracy (Schlotterer *et al.* (2014) *Nature Reviews Genetics*).

A number of aligners have been written to either compensate or make use of the specific properties of sequence data obtained using different technologies. For example the aligner TMAP was specifically designed to be used with Ion Torrent data which has problems with resolving homopolymer lengths accurately. Similarly there are specific aligners that handle SoliD color-space data which cannot be effectively aligned as nucleotide sequences. In this tutorial we will make use of Bowtie2 as it is one of the most commonly used aligners that balances speed, accuracy and memory requirements. My current favourite aligner is the STAR aligner, which was designed to align RNA sequences to genomic locations; it is rather accurate and very fast, but requires far more memory to run than bowtie/bowtie2. An alternative aligner that is also widely used is the BWA aligner. You can find an unpublished comparison of mapping performance here, a benchmarking of 9 mapping tools here and a more in-depth look at mapping RNA sequences to genomes here.

1.1 The mapping commands

What you need for mapping are two sequence files, one containing the quality-trimmed reads (fastq format) and the other containing the draft genome (in the fasta format).

Create a new folder named `Mapping` (with `mkdir`) and copy the two sequence files into it (with `cp`). For example for the assembly made using IonTorrent data:

```
mkdir Mapping

cp GenomeAssembly/IonTorrentDeNovoAssembly_assembly/IonTorrentDeNovoAssembly_d_results/\
IonTorrentDeNovoAssembly_out.unpadded.fasta \
Mapping/

cp RAWREADS_trimmed_trimmed.fq \
Mapping/

cd Mapping
```

Here the `RAWREADS_trimmed_trimmed.fq` refer to the IonTorrent reads after quality trimming using TrimGalore. Here we are simply making copies of the various files in different locations for performing different analyses. This is simple to describe and understand, but is not a particularly good way to organise your files, both

because it wastes disk space[1] and (more seriously) it makes it easy to end up with files with the

[1]In some modern file systems that make use of Copy-On-Write, there won't actually be any copying of the data unless one of the files is modified.

same names, but with slightly different content[2].

In the Mapping folder, we first create an index for the reference genome using the following command (enter the folder using `cd` before calling this command) :

```
bowtie2-build -f IonTorrentDeNovoAssembly_out.unpadded.fasta GENOMENAME
```

You should change `GENOMENAME` to something that is suitable for your data.

Now, to align the trimmed reads against the genome, use the following command:

```
nohup bowtie2 -p 1 \
-q \
--phred33 \
--sensitive \
--no-unal \
--al-gz FILE_Aligned.fq.gz \
--un-gz FILE_Unaligned.fq.gz \
--met-file MetricsFile.txt \
-x GENOMENAME \
-U RAWREADS_trimmed_trimmed.fq \
-S MAPPEDREADS.sam >Logfile.log &
```

The meaning of the options used:

`-p 1` Causes Bowtie 2 to use a single thread. Depending on the number of users and libraries we will probably increase this.

`-q` Informs the program that the reads to map are saved in fastq files.

`--phred33` Sets the quality encoding of the fastq files to "Phred+33".

`--sensitive` sets several options at once regarding the seeding and other adjustments.

`--no-unal` Suppress SAM records for reads that do not align.

`--al-gz` Write unpaired reads that align at least once to to the specified file.

`--un-gz` Write unpaired reads that failed to align to the specified file.

`--met-file` Write bowtie2 metrics to Metricsfile.txt.

`-x` Specifies the name of the genome.

`-U` Specify the unpaired reads to align (can contain a comma-separated list of several fq files).

`-S` Specify the sam file to which the alignment shall be saved.

You can't set the exact number of mismatches in the seed, but you can adjust the mismatch penalty.

[2]How to handle lots of files created by various versions of data flows or pipelines is not a simple problem and there are many systems that have been developed to address such problems. This comes under the general heading of version control systems, and is outside the scope of this course. In general though, it is wise to follow rule number one of database design: 'never store a piece of information in more than one location'. I.e. don't copy stuff around like we are doing here.

The program should run no longer than 10-20 mins. The resulting output file will be in the SAM format. For a detailed description of this format, see here.

To map the Illumina data we follow a similar procedure; however, we need to modify the call to `bowtie2` as the Illumina data contains paired reads. To find out how we can do this, we can run `bowtie2` without any arguments or specifying the `--help` option. This will print out the usage information. Knowing how to read usage information is one of the most important things you can do as you'll then be able to run most applications without relying on others. If you do this, you'll see something like this:

```
lmj@tej-X8DTG-QF:~$ bowtie2 --help
Bowtie 2 version 2.1.0 by Ben Langmead (langmea@cs.jhu.edu, www.cs.jhu.edu/~langmea)
Usage:
  bowtie2 [options]* -x <bt2-idx> {-1 <m1> -2 <m2> | -U <r>} [-S <sam>]

  <bt2-idx>  Index filename prefix (minus trailing .X.bt2).
             NOTE: Bowtie 1 and Bowtie 2 indexes are not compatible.
  <m1>       Files with #1 mates, paired with files in <m2>.
             Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).
  <m2>       Files with #2 mates, paired with files in <m1>.
             Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).
  <r>        Files with unpaired reads.
             Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).
  <sam>      File for SAM output (default: stdout)

  <m1>, <m2>, <r> can be comma-separated lists (no whitespace) and can be
  specified many times.  E.g. '-U file1.fq,file2.fq -U file3.fq'.

Options (defaults in parentheses):

 Input:
  -q                 query input files are FASTQ .fq/.fastq (default)
  --qseq             query input files are in Illumina's qseq format
.... more options
```

Let us consider the top lines first. This is the basic usage information that tells you the arguments you need to specify and their order.

```
Usage:
  bowtie2 [options]* -x <bt2-idx> {-1 <m1> -2 <m2> | -U <r>} [-S <sam>]
```

Things contained in square brackets `[stuff in square brackets]` denote optional arguments. So, the above (`bowtie2 [options] ...`) indicates that optional options (specified with `-` or `--`) should be specified before other arguments. After these options (of which there may be none) you should specify the value of the `-x` option. Looking down, you can see that `<bt2-idx>`, is a placeholder for the name of the index that you built using `bowtie2` in the previous section. If you have assembled a genome from the Illumina data on its own this will be a different index file based on a different assembly sequence, so we will need to change this value.

The next section of the usage line is contained in squiggly brackets (usually referred to as braces) indicating that you have a choice of two or more alternatives. These alternatives are seperated by the pipe (`|`) character which in computing languages is usually taken to mean 'or'. So the section `{-1 <m1> -2 <m2> | -U <r>}` reads as 'either specify the values of `-1` and `-2` or the value of `-U`'. Looking at the explanation further down, you can see that `<m1>` and `<m2>` refer to mate or paired sequences, whereas `<r>` refers to unpaired reads. The last section simply specifies to which file we wish to write the output; it's optional (inside `[]`) and if you don't specify this, the output will simply be written to the terminal (i.e. `STDOUT`). This is useful, because we can then pipe the data to other applications in a single command.

So reading the usage line (also known as the synopsis) we can design our command line. If our paired reads are in files `RAWREADS_fw_trimmed_trimmed.fq` and `RAWREADS_rv_trimmed_trimmed.fq`, and the index for our assembly genome is in `GENOMENAME.X.bt2`, the command without any of the optional options would be:

```
bowtie2 -x GENOMENAME -1 RAWREADS_fw_trimmed_trimmed.fq \
-2 RAWREADS_rv_trimmed_trimmed.fq -S MAPPED.sam
```

Here we haven't specifed any of the options we used for the IonTorrent data above and the program will simply use the default options. To see what the default options are you should read the rest of the help section that is printed out when you run `bowtie2` without any arguments. You can probably use most of the options as we used above, though you should not assume this.

Given that the Illumina data is paired end sequence data you should pay special attention to the Paired-end section of the help text. In particular consider the values of `-I` and `-X` and whether the default options are reasonable for your libraries.

1.2 Running the commands in a script for posterity

As was emphasised in the section on Unix tools for bioinformatics, you really shouldn't type these commands directly into a terminal window. It's too easy to make a mistake when you have to specify many options, and you will not have a record of the command that you actually used. Instead we will write the commands into a text file and ask the shell (in this case bash[3]) to run the commands non-interactively. In the simplest case you just make one file for each command, and run these seperately. However, it is much better to embed the full process into a single script as all the information will be in a single place. Here what we have done is:

- made a directory for our mapping (`mkdir`)
- copied the data files to that directory (`cp`)
- entered the directory (`cd`)
- run bowtie2 to make an index
- run bowtie2 to map the sequences

We can put all of those commands into a single shell script, or we can make the directories manually and only include the more complicated commands in the script. Which is better depends a little bit on the situation; if you have lots of different sequence files that you wish to map in different ways then you might want to put all the directory commands into the script; ideally doing this in an automated way using loops and assembling the directory names automatically. However, here I would suggest the simple option of manually making the directories and having simpler script files to avoid using more complex shell scripting.

[3]bash stands for Bourne Again Shell, and is a bit of a joke on the fact that Bash is an extension or enhancement of the Bourne shell. These days it's probably the most common shell used, but as always there are people who consider it an abomination.

Hence once you have created the appropriate directory and copied the sequence files (as above) you can write (eg: `nano pgm_map.sh`) a script (to map IonTorrent data) that looks a bit like:

```bash
#!/bin/bash

## here you can define some variables that specify the names of
## input and output files

RAWREADS=breiflabb_pgm
GENOMENAME=breiflabb_pgm
FILE="$GENOMENAME"_bt2

## note that when you use the variables you have to put a $
## sign in front of them
## and if you want to concatenate to words you need to
## to quote the variables so that the shell knows where
## the variable ends

## first build the index:
bowtie2-build -f IonTorrentDeNovoAssembly_out.unpadded.fasta $GENOMENAME

## then use that to map the sequences:
bowtie2 -p 1 -q -phred33 --sensitive --no-unal \
--al-gz "$FILE"_Aligned.fq.gz --un-gz "$FILE"_Unaligned.fq.gz \
--met-file MetricsFile.txt \
-x $GENOMENAME -U $RAWREADS_trimmed_trimmed.fq \
-S "$GENOMENAME"_bt2_mapped.sam > bt2_log.log

## here you can put some comments to indicate what the different
## options mean and why you have chosen them
```

To run this script (`pgm_map.sh`) you can manually invoke the bash interpreter:

```
bash pgm_map.sh
```

Or you can change the permissions of the file and run it directly as its own program:

```
chmod +x pgm_map.sh
./pgm_map.sh
```

Of course, as before, you should probably run it using `nohup`:

```
nohup bash pgm_map.sh > map_log &
```

1.3 The (lack of) directory structure

If you follow these procedures you will end up with one or more directories containing copies of the raw reads, the assembly sequence, genome index files, and as you keep following the instructions below, a whole load of other files. This really is a bit of an unholy mess. Don't do this at home. Instead try to set up directory structures that make sense for your project and try to follow some sort of rules as to what goes where. This is most easily established by running the analyses through scripts that you can reuse for new data. As usual though, there isn't that much point in trying to work out exactly how to structure your project before you started as you will almost certainly wish to change it as it develops. Learn the Unix basics well and this won't be a problem.

2 Filter mappings

To remove unmapped reads, reads below a mapping quality of 20, and reads that were not aligned uniquely (reads that were mapped to >1 places in the genome), use the python script Bowtie2Filtering.py:

```
Bowtie2Filtering.py -mq -u -a -s MAPPEDREADS.sam
```

Your filtered reads will be saved in `MAPPEDREADSfiltered.sam`

Alternatively, you can use samtools to filter out reads with a mapping quality <20:

```
samtools view -Sh -q 20 -o MAPPEDREADS_QualityAbove20.sam MAPPEDREADS.sam
```

Options:

`-S` Input is in the sam format

`-h` Include the samfile header in the output

`-q` Skip alignments with a mapping quality below 20

Note that it is usually possible to limit the alignments reported by the mapping program by adjusting the options; for at least some programs you can instruct the program to only report unique matches and so it might seem unnecessary to perform post-filtering steps like these. However, given that the mapping process takes far more time than the filtering process it often makes sense to map using permissive criteria and then to filter these depending on the questions being addressed.

2.1 Removing duplicate reads

After quality-trimming, we counted the fraction of duplicate reads. Duplicate reads have the same start and end coordinates and map to the same region. Duplicates result from primer or PCR bias towards these reads. As they can skew genotype estimates, they should be removed before SNP calling.

To remove duplicates, we will use 'MarkDuplicates' from the Picard command line tools. An alternative tool is samtools rmdup, which considers single-end reads to be duplicates when their mapping locations are the same - even if the base composition differs between the reads.

First, we need to convert our sam file to a bam file (a binary, compressed version of a sam file that is not human-readable) and sort the reads by the leftmost mapping coordinates.

```
samtools view -bSh MAPPEDREADS.sam  > MAPPEDREADS.bam
samtools sort -o MAPPEDREADS_sorted.bam MAPPEDREADS.bam
```

Meaning of the options:

`-b` output in bam format

`-S` input in sam format

`-h` include the header in the output

Then, you can use the 'MarkDuplicates' tool from Picard to remove the duplicates from the sorted bam file. Here we invoke the Picard tools using the command `picard-tools`. However, there isn't actually any such program. The Picard tools are implemented as a Java archive, and to be run should invoke the java virtual machine (or run time environment). If you look at the documentation on the Picard tool web site it will tell you to invoke the Picard tools using:

```
java jvm-args -jar picard.jar PicardToolName OPTION1=value1 OPTION2=value2...
```

That's a bit of a mouthful, and in order to use it you would also need to know where to find the picard.jar file. To simplify this, I've set up an alias to this command using the `alias` program. This has been added to the end of your `.bashrc` files which are read by `bash` when you log in to your account (try `more .bashrc` after logging in to see the contents of this file). At the end of the `.bashrc` file you will find following lines:

```
## in order to run picard-tools as picard tools we set up an alias
alias picard-tools='java -jar /usr/local/picard-tools/picard.jar'
```

As a result, when you type `picard-tools` into your terminal, the shell actually sees `java -jar /usr/local/picard-tools/picard.jar`, after which you can specify the tool and the options you wish to use. Note that this isn't standard in any way, and that you may need to use the Picard tools differently depending on your local setup. Doing it this way also doesn't let you specify any options to the java virtual machine (the `jvm-args` above) and if for whatever reason you need to do this you will have to use the full command as shown above.

To remove duplicates using the Picard MarkDuplicates tool we can then simply:

```
picard-tools MarkDuplicates \
INPUT=MAPPEDREADS_sorted.bam \
OUTPUT=MAPPEDREADS_dedup.bam \
METRICS_FILE=MAPPED_metricsfile \
ASSUME_SORTED=true \
VALIDATION_STRINGENCY=SILENT \
REMOVE_DUPLICATES=true
```

Duplication metrics will be written to the `MAPPED_metricsfile`. We again very strongly recommend that you put these commands into a shell file and run that rather than to run directly from the command line.

2.2 Re-alignment around indels

Reads that are spanning InDels are often misaligned and can result in false SNPs (see Schlotterer *et al.* (2014) *Nature Reviews Genetics*). These reads should be removed or re-aligned. We have not enough time to re-align the reads in this course but the required steps (using GATK) are described in detail here: `http://sfg.stanforde.edu/SFG.pdf`.

3 Visualizing alignments

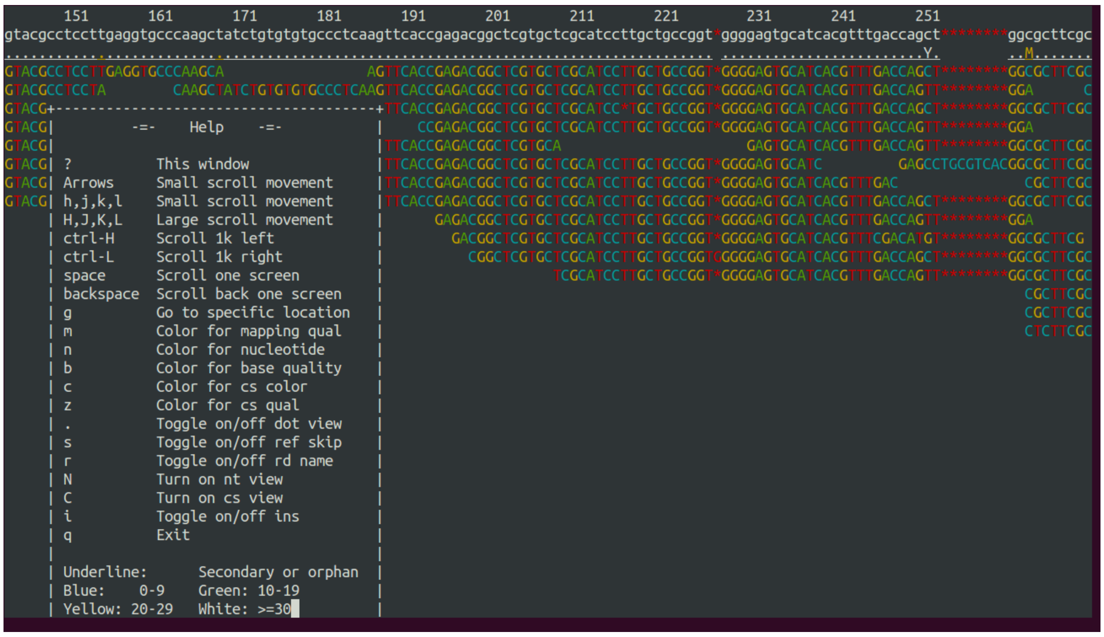
3.1 Samtools tview: command-line viewer

The command line tool samtools tview allows you to view your alignments directly in the command line window. What you need is the reference genome (fasta file) and the sorted and deduplicated alignment file (bam file). First, you need to index the bam file before using `samtools tview`:

```
samtools index MAPPEDREADS_dedup.bam

samtools tview MAPPEDREADS_dedup.bam \
IonTorrentDeNovoAssembly_out.unpadded.fasta
```

Fig. 1 shows a screenshot of tview. When you hit the `?` on your keyboard, you will see the range of options to navigate through the alignment. You can change the contig that you are looking at by hitting `g` and then enter in the Goto-window the name of the contig, like `IonTorrentDeNovoAssembly_c3`. You can exit the alignment viewer by hitting `q`.



```
       151       161       171       181       191       201       211       221        231       241       251
gtacgcctccttgaggtgcccaagctatctgtgtgtgccctcaagttcaccgagacggctcgtgctcgcatccttgctgccggt*ggggagtgcatcacgtttgaccagct********ggcgcttcgc
.................................................................................... ........................Y.        ..M.......
GTACGCCTCCTTGAGGTGCCCAAGCA                 AGTTCACCGAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGCT********GGCGCTTCGC
GTACGCCTCCTA        CAAGCTATCTGTGTGTGCCCTCAAGTTCACCGAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGTT********GGA      C
GTACG+--------------------------------------+TTCACCGAGACGGCTCGTGCTCGCATCC*TGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGCT********GGCGCTTCGC
GTACG|         -=-    Help    -=-           |    CCGAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGTT********GGA
GTACG|                                      |TTCACCGAGACGGCTCGTGCA                      GAGTGCATCACGTTTGACCAGTT********GGCGCTTCGC
GTACG| ?          This window               |TTCACCGAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATC         GAGCCTGCGTCACGGCGCTTCGC
GTACG| Arrows     Small scroll movement     |TTCACCGAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGAC               CGCTTCGC
GTACG| h,j,k,l    Small scroll movement     |TTCACCGAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGCT********GGCGCTTCGC
     | H,J,K,L    Large scroll movement     |      GAGACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGTT********GGA
     | ctrl-H     Scroll 1k left            |        GACGGCTCGTGCTCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTCGACATGT********GGCGCTTCG
     | ctrl-L     Scroll 1k right           |          CGGCTCGTGCTCGCATCCTTGCTGCCGGTGGGGAGTGCATCACGTTTGACCAGCT********GGCGCTTCGC
     | space      Scroll one screen         |                    TCGCATCCTTGCTGCCGGT*GGGGAGTGCATCACGTTTGACCAGTT********GGCGCTTCGC
     | backspace  Scroll back one screen    |                                                                            CGCTTCGC
     | g          Go to specific location   |                                                                            CGCTTCGC
     | m          Color for mapping qual    |                                                                            CTCTTCGC
     | n          Color for nucleotide      |
     | b          Color for base quality    |
     | c          Color for cs color        |
     | z          Color for cs qual         |
     | .          Toggle on/off dot view    |
     | s          Toggle on/off ref skip    |
     | r          Toggle on/off rd name     |
     | N          Turn on nt view           |
     | C          Turn on cs view           |
     | i          Toggle on/off ins         |
     | q          Exit                      |
     |                                      |
     | Underline:      Secondary or orphan  |
     | Blue:    0-9    Green: 10-19         |
     | Yellow: 20-29   White: >=30          |
```

Figure 1: Screenshot of tview

3.2 IGV: viewer with a graphical user interface

I bet that many of you prefer to look at the alignment in a graphical user interface. A decent free alignment viewer is igv, the Integrative Genomics Viewer (see Fig. 2 for a screenshot). Once you have registered, you can launch the program with Java Web Start. We can't promise that this works well in the course, since everything that relies on a graphical user interface can be quite slow when using a remote connection. Thus, you might want to download the required files (deduplicated SAM file and reference genome) and try out igv on your private computer. The interface is pretty much self-explanatory. To look at the alignment, you first need to load a genome and then add the mapped, sorted and indexed bam file.

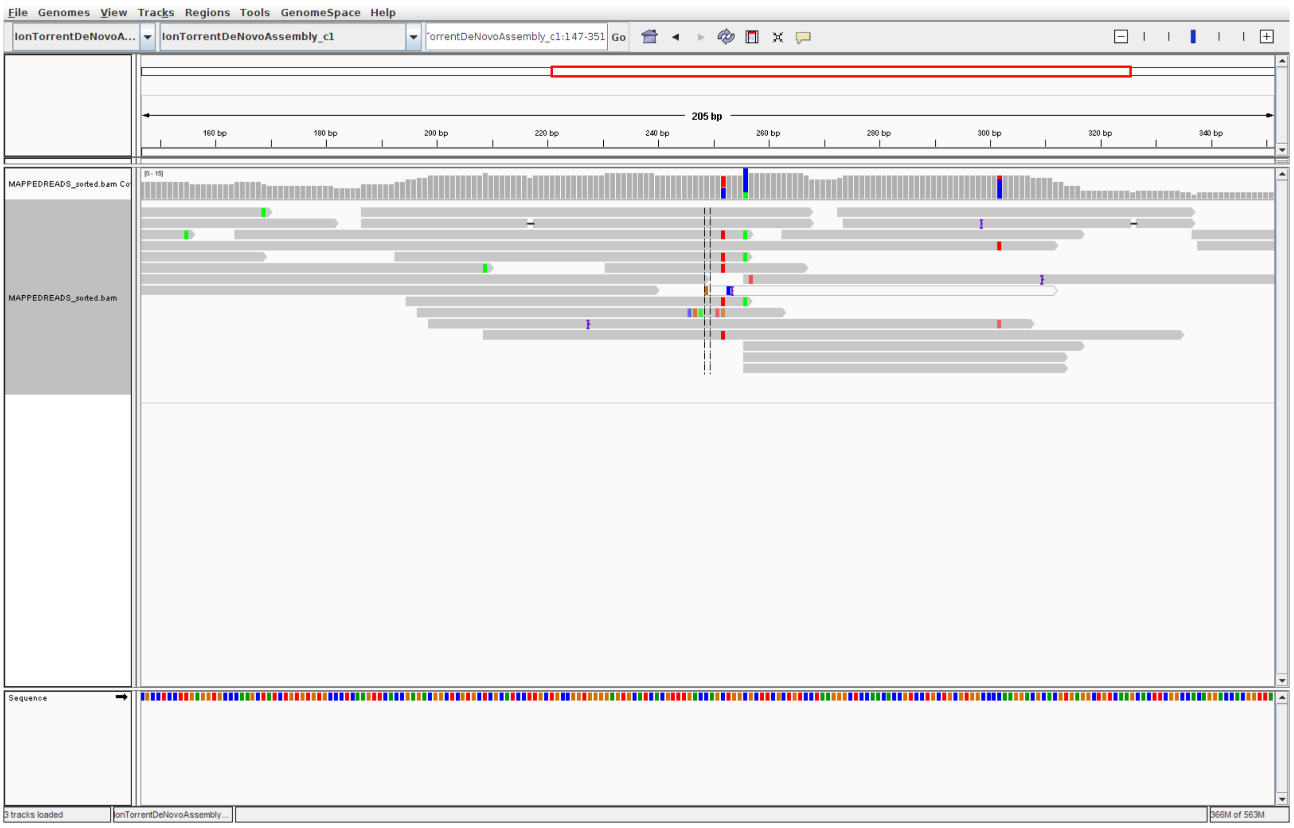

File Genomes View Tracks Regions Tools GenomeSpace Help
IonTorrentDeNovoA...
IonTorrentDeNovoAssembly_c1
orrentDeNovoAssembly_c1:147-351
Go
205 bp
160 bp
180 bp
200 bp
220 bp
240 bp
260 bp
280 bp
300 bp
320 bp
340 bp
MAPPEDREADS_sorted.bam Co
[0 - 15]
MAPPEDREADS_sorted.bam
Sequence
3 tracks loaded
IonTorrentDeNovoAssembly...
366M of 563M

Figure 2: Screenshot of igv with reads aligned to a reference and colored mismatches

4 BONUS: SNP calling with samtools mpileup and bcftools

Given sequences aligned to a reference it seems that it should be trivial to identify sequence variants. Surely any mismatches between the reference (in this case our assembly) and reads is evidence for the presence of a sequence variant. However, if the probability of observing a sequencing error is larger than the frequency of sequence variants within the population (an individual can be considered as a population of two haploid genomes) then most sequence mismatches will be caused by sequencing errors. This is usually the case (and overwhelmingly so) when looking at individuals from within a single species and in order to identify a position as a sequence variant we need to have more than one read diverging from the reference. How many reads are required depends on the total number of reads, the qualities of those reads and the expected variant frequency. If we are sequencing populations, then we also have to consider the rarity of a given allele; the rarer the allele one wishes to discover the larger the sequencing coverage required. This has led to the development of a rather large number of variant detection algorithms and programs (see table 3 of Schlotterer *et al* for a list), and the difficulty of balancing computation times, sensititivy and accuracy makes it likely that more methods and or implementations will be written.

Here we will use the `samtools mpileup` in conjuction with `bcftools`. Computationally these are some of the simplest ways to detect variants and are widely used. For more in depth analyses we would recommend that you consider using other tool sets that have the potential to provide more accurate variant detection at the cost of more processing time.

The tool `samtools mpileup` defaults to creating a pileup file, which summarizes aligned base calls in a text format (See here for an overview of its options, and here for a detailed characterization of a pileup file `http://samtools.sourceforge.net/pileup.shtml`). If you call `samtools mpileup` with the `-u` or `-g` option the output format is a vcf or bcf (compressed binary version of vcf) file; vcf stands for 'variant call format'. Its format specifications are described here and summarized in Fig. 3.

The first step for calling SNPs from your aligned and deduplicated reads is:

```
samtools mpileup -g \
-f \
IonTorrentDeNovoAssembly_out.unpadded.fasta \
-q 20 \
-Q 20 \
-t DP \
-t SP \
MAPPEDREADS_dedup.bam  > MAPPEDREADS_dedup.bcf
```

The chosen options are described on this page. By setting the `-t SP` and `-t DP` tags, samtools mpileup provides:

`-t SP` per-sample Phred-scaled strand bias P-value

`-t DP` per sample read depth

To call SNPs from the bcf file, we use bcftools:

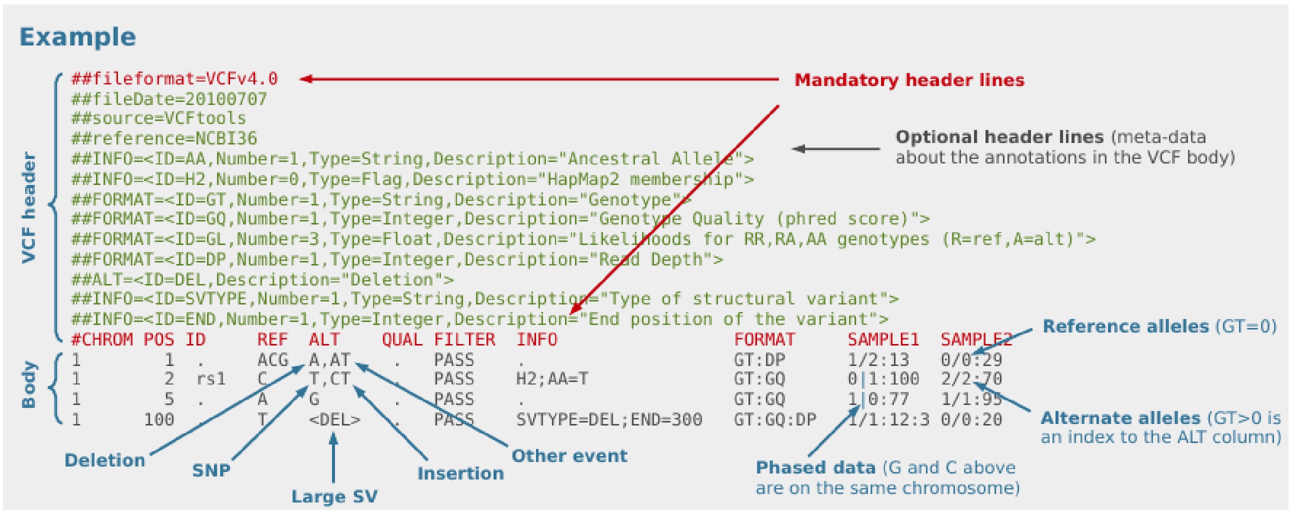
```
bcftools call -vm -V indels MAPPEDREADS_dedup.bcf >  MAPPEDREADS_variants.vcf
```

Options:

`-v` Output variant sites only

`-V indels` Skip indels

`-m` model for multiallelic and rare-variant calling


Example
VCF header
Body
Mandatory header lines
Optional header lines (meta-data about the annotations in the VCF body)
Reference alleles (GT=0)
Alternate alleles (GT>0 is an index to the ALT column)
Phased data (G and C above are on the same chromosome)
Deletion
SNP
Large SV
Insertion
Other event

```
##fileformat=VCFv4.0
##fileDate=20100707
##source=VCFtools
##reference=NCBI36
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality (phred score)">
##FORMAT=<ID=GL,Number=3,Type=Float,Description="Likelihoods for RR,RA,AA genotypes (R=ref,A=alt)">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##ALT=<ID=DEL,Description="Deletion">
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural variant">
##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the variant">
#CHROM POS ID     REF  ALT    QUAL FILTER  INFO                 FORMAT      SAMPLE1   SAMPLE2
1        1 .      ACG  A,AT   .    PASS    .                    GT:DP       1/2:13    0/0:29
1        2 rs1    C    T,CT   .    PASS    H2;AA=T              GT:GQ       0|1:100   2/2:70
1        5 .      A    G      .    PASS    .                    GT:GQ       1|0:77    1/1:95
1      100 .      T    <DEL>  .    PASS    SVTYPE=DEL;END=300   GT:GQ:DP    1/1:12:3  0/0:20
```

Figure 3: VCF file overview from Petr Danecek

To count how many SNPs were found, use the following command:

```
grep -v -c '^#' MAPPEDREADS_variants.vcf
```

The option `-v` in combination with `^#` excludes all header lines that start with (`^`) the `#`-sign. With the `-c` option, grep counts the lines instead of writing them out.

To filter out SNPs that are low quality or covered by low depth, we can use the `vcfutils.pl varFilter` that comes with samtools:

```
vcfutils.pl varFilter -d 5 -w 3 -Q 20  MAPPEDREADS_variants.vcf > MAPPEDREADS_variants_filtered.vcf
```

Options used:

`-d 5` minimum read depth of 5

`-w 3` SNP within 3 bp around a gap to be filtered. This may be an alternative solution to re-alignment around indels

`-Q 20` minimum mapping quality of 20

Another useful option can be:

`-1 0.0001` min P-value for strand bias (given the PV4-tag in the vcf file). We obtained the PV4-tag by setting the `-t SP` tag in `samtools mpileup`. This option filters out the SNPs that have a strong strand-bias: SNPs that are supported by one strand and not the other.

Count how many SNPs are left after filtering

```
grep -v -c '^#' MAPPEDREADS_variants_filtered.vcf
```

The SNPs can be visualized with IGV. For this, we first need to compress and index the vcf files:

```
bgzip -c \
MAPPEDREADS_variants_filtered.vcf \
> MAPPEDREADS_variants_filtered.vcf.gz

tabix \
-p vcf \
MAPPEDREADS_variants_filtered.vcf.gz
```

Open IGV and load the indexed bam file and the indexed vcf file. Emacs 24.3.1 (Org mode 8.3.4)