1 NGS Data formats and QC

1.1 Introduction

There are several file formats for storing Next Generation Sequencing (NGS) data. In this tutorial we will look at some of the most common formats for storing NGS reads and variant data. We will cover the following formats:

FASTQ - This format stores unaligned read sequences with base qualities
SAM/BAM - This format stores unaligned or aligned reads (text and binary formats)
CRAM - This format is similar to BAM but has better compression than BAM
VCF/BCF - Flexible variant call format for storing SNPs, indels, structural variations (text and binary formats)

Following this, we will work through some examples of converting between the different formats.

Further to understanding the different file formats, it is important to remember that all sequencing platforms have technical limitations that can introduce biases in your sequencing data. Because of this it is very important to check the quality of the data before starting any analysis, whether you are planning to use something you have sequenced yourself or publicly available data. In the latter part of this tutorial we will describe how to perform a QC assessment for your NGS data.

1.2 Learning outcomes

On completion of the tutorial, you can expect to be able to:

- Describe the different NGS data formats available (FASTQ, SAM/BAM, CRAM, VCF/BCF)
- Perform a QC assessment of high throughput sequence data
- Perform conversions between the different data formats

1.3 Tutorial sections

This tutorial comprises the following sections:
1. Data formats
2. QC assessment

If you have time you can also complete:

3. File conversion

1.4 Authors

This tutorial was written by Jacqui Keane and Sara Sjunnebo based on material from Petr Danecek and Thomas Keane.

1.5 Running the commands from this tutorial

You can follow this tutorial by typing all the commands you see into a terminal window. This is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, open a new terminal on your computer and type the command below:

```
[ ]: cd ~/course_data/data_formats/
```

Now you can follow the instructions in the tutorial from here.

1.6 Let's get started!

This tutorial assumes that you have samtools, bcftools and Picard tools installed on your computer. These are already installed on the VM you are using. To check that these are installed, you can run the following commands:

```
[ ]: samtools --help
```

```
[ ]: bcftools --help
```

```
[ ]: picard -h
```

This should return the help message for samtools, bcftools and picard tools respectively.

To get started with the tutorial, go to the first section: Data formats

2 Data formats for NGS data

Here we will take a closer look at some of the most common NGS data formats. First, check you are in the correct directory.

```
[ ]: pwd
```

It should display something like:

```
/home/genomics/workshop_materials/file_types/data_formats
```

2.1 FASTA

The FASTA format is used to store both nucleotide data and protein sequences. Each sequence in a FASTA file is represented by two parts, a header line and the actual sequence. The header always starts with the symbol ">" and is followed by information about the sequence, such as a unique identifier. The following lines show two sequences represented in FASTA format:

```
>Sequence_1
CTTGACGACTTGAAAAATGACGAAATCACTAAAAAACGTGAAAAATGAGAAATG
AAAATGACGAAATCACTAAAAAACGTGACGACTTGAAAAATGACCAC
>Sequence_2
CTTGAGACGAAATCACTAAAAAACGTGACGACTTGAAGTGAAAAATGAGAAATG
AAATCATGACGACTTGAAGTGAAAAAGTGAAAAATGAGAAATGAACGTGACGAC
AAAATGACGAAATCATGACGACTTGAAGTGAAAAATAAATGACC
```

2.1.1 Exercises

Q1: How many sequences are there in the fasta file data/example.fasta? (Hint: is there a grep option you can use? Be VERY careful to quote the redirection sign '>', otherwise the file will be overwritten!!!)

```
[ ]:
```

2.2 FASTQ

FASTQ is a data format for sequencing reads. It is an extension to the FASTA file format, and includes a quality score for each base. Have a look at the example below, containing two reads:

```
@ERR007731.739 IL16_2979:6:1:9:1684/1
CTTGACGACTTGAAAAATGACGAAATCACTAAAAAACGTGAAAAATGAGAAATG
+
BBCBCBBBBBBBABBABBBBBBBABBBBBBBBBBBBBABAAAABBBBB=@>B
@ERR007731.740 IL16_2979:6:1:9:1419/1
AAAAAAAAAGATGTCATCAGCACATCAGAAAAGAAGGCAACTTTAAAACTTTTC
+
BBABBBABABAABABABBABBBAAA>@B@BBAA@4AAA>.>BAA@779:AAA@A
```

We can see that for each read we get four lines:

1. The read metadata, such as the read ID. Starts with `@` and, for paired-end Illumina reads, is terminated with /1 or /2 to show that the read is the member of a pair.
2. The read
3. Starts with `+` and optionally contains the ID again
4. The per base Phred quality score

The quality scores range (in theory) from 1 to 94 and are encoded as ASCII characters. The first 32 ASCII codes are reserved for control characters which are not printable, and the 33rd is reserved for space. Neither of these can be used in the quality string, so we need to subtract 33 from whatever the value of the quality character is. For example, the ASCII code of "A" is 65, so the corresponding quality is:

```
Q = 65 - 33 = 32
```

The Phred quality score `Q` relates to the base-calling error probability `P` as

$$P = 10^{-Q/10}$$

The Phred quality score is a measure of the quality of base calls. For example, a base assigned with a Phred quality score of 30 tells us that there is a 1 in 1000 chance that this base was called incorrectly.

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10,000	99.99%
50	1 in 100,000	99.999%
60	1 in 1,000,000	99.9999%

2.2.1 Exercises

Q2: How many reads are there in the file example.fastq? (Hint: remember that `@` is a possible quality score. Is there something else in the header that is unique?)

```
[ ]:
```

2.3 SAM

SAM (Sequence Alignment/Map) format is a unified format for storing read alignments to a reference genome. It is a standard format for storing NGS sequencing reads, base qualities, associated meta-data and alignments of the data to a reference genome. If no reference genome is available, the data can also be stored unaligned.

The files consist of a header section (optional) and an alignment section. The alignment section contains one record (a single DNA fragment alignment) per line describing the alignment between fragment and reference. Each record has 11 fixed columns and optional key:type:value tuples. Open the SAM/BAM file specification document `https://samtools.github.io/hts-specs/SAMv1.pdf` either in a web browser or you can find a copy in the QC directory as you may need to refer to it throughout this tutorial.

Now let us have a closer look at the different parts of the SAM/BAM format.

2.3.1 Header Section

The header section of a SAM file looks like:

```
@HD VN:1.0  SO:coordinate
@SQ SN:test_ref LN:17637
@RG ID:ERR003612 PL:ILLUMINA LB:g1k-sc-NA20538-TOS-1 PI:2000 DS:SRP000540 SM:NA20538 CN:SC
```

Each line in the SAM header begins with an `@`, followed by a two-letter header record type code as defined in the SAM/BAM format specification document. Each record type can contain meta-data captured as a series of key-value pairs in the format of 'TAG:VALUE'.

Read groups One useful record type is RG which can be used to describe each lane of sequencing. The RG code can be used to capture extra meta-data for the sequencing lane. Some common RG TAGs are:

- ID: SRR/ERR number
- PL: Sequencing platform
- PU: Run name
- LB: Library name
- PI: Insert fragment size
- SM: Individual/Sample
- CN: Sequencing centre

While most of these are self explanitory, insert fragment size may occasionally be negative. This simply indicates that the reads found are overlapping while its size is less than 2 x read length.

2.3.2 Exercises

Look at the following line from the header of a SAM file and answering the questions that follow:

```
@RG ID:ERR003612 PL:ILLUMINA LB:g1k-sc-NA20538-TOS-1 PI:2000 DS:SRP000540 SM:NA20538 CN:SC
```

Q3: What does RG stand for?

[ ]:

Q4: What is the sequencing platform?

[ ]:

Q5: What is the sequencing centre?

[ ]:

Q6: What is the lane identifier?

[ ]:

Q7: What is the expected fragment insert size?

[ ]:

2.3.3 Alignment Section

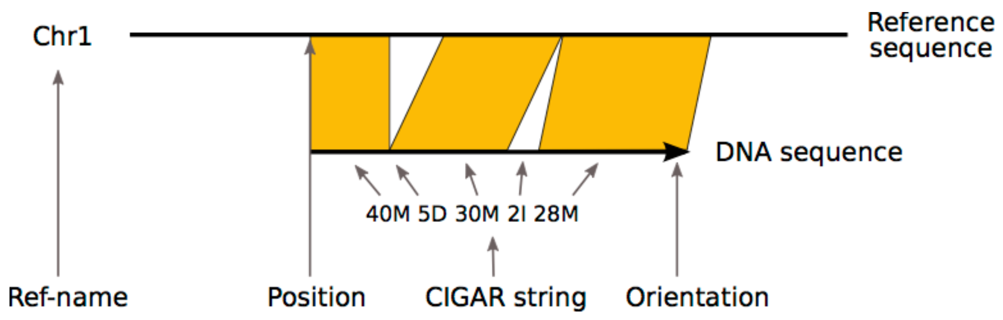
The alignment section of SAM files contains one line per read alignment, an example is

```
ERR005816.1408831 163 Chr1    19999970    23  40M5D30M2I28M   =
20000147    213 GGTGGGTGGATCACCTGAGATCGGGAGTTTGAGACTAGGTGG...
<=@A@??@=@A@A>@BAA@ABA:>@<>=BBB9@@2B3<=@A@...
```

Each of the lines are composed of multiple columns listed below. The first 11 columns are mandatory.

1. QNAME: Query NAME of the read or the read pair i.e. DNA sequence
2. FLAG: Bitwise FLAG (pairing, strand, mate strand, etc.)
3. RNAME: Reference sequence NAME
4. POS: 1-Based leftmost POSition of clipped alignment
5. MAPQ: MAPping Quality (Phred-scaled)
6. CIGAR: Extended CIGAR string (operations: MIDNSHPX=)
7. MRNM: Mate Reference NaMe ('=' if same as RNAME)
8. MPOS: 1-Based leftmost Mate POSition
9. ISIZE: Inferred Insert SIZE
10. SEQ: Query SEQuence on the same strand as the reference
11. QUAL: Query QUALity (ASCII-33=Phred base quality)
12. OTHER: Optional fields

The image below provides a visual guide to some of the columns of the SAM format.



2.3.4 Exercises

Let's have a look at example.sam. Notice that we can use the standard UNIX operations like **cat** on this file.

```
[ ]: cat data/example.sam
```

Q8: What is the mapping quality of ERR003762.5016205? (Hint: can you use grep and awk to find this?)

```
[ ]:
```

Q9: What is the CIGAR string for ERR003814.6979522? (Hint: we will go through the meaning of CIGAR strings in the next section)

```
[ ]:
```

Q10: What is the inferred insert size of ERR003814.1408899?

```
[ ]:
```

2.3.5 CIGAR string

Column 6 of the alignment is the CIGAR string for that alignment. The CIGAR string provides a compact representation of sequence alignment. Have a look at the table below. It contains the meaning of all different symbols of a CIGAR string:

Symbol	Meaning
M	alignment match or mismatch
=	sequence match
X	sequence mismatch
I	insertion into the read (sample sequenced)
D	deletion from the read (sample sequenced)
S	soft clipping (clipped sequences present in SEQ)
H	hard clipping (clipped sequences NOT present in SEQ)
N	skipped region from the reference
P	padding (silent deletion from padded reference)

Below are two examples describing the CIGAR string in more detail.

Example 1:
Ref: ACGTACGTACGTACGT
Read: ACGT- - - - ACGTACGA
Cigar: 4M 4D 8M

The first four bases in the read are the same as in the reference, so we can represent these as 4M in the CIGAR string. Next comes 4 deletions, represented by 4D, followed by 7 alignment matches and one alignment mismatch, represented by 8M. Note that the mismatch at position 16 is included in 8M. This is because it still aligns to the reference.

Example 2:
Ref: ACTCAGTG- - GT
Read: ACGCA- TGCAGTtagacgt
Cigar: 5M 1D 2M 2I 2M 7S

Here we start off with 5 alignment matches and mismatches, followed by one deletion. Then we have two more alignment matches, two insertions and two more matches. At the end, we have seven soft clippings, 7S. These are clipped sequences that are present in the SEQ (Query SEQuence on the same strand as the reference).

2.3.6 Exercises

Q11: What does the CIGAR from Q9 mean?

```
[ ]:
```

Q12: How would you represent the following alignment with a CIGAR string?

Ref: ACGT- - - - ACGTACGT
Read: ACGTACGTACGTACGT

```
[ ]:
```

2.3.7 Flags

Column 2 of the alignment contains a combination of bitwise FLAGs describing the alignment. The following table contains the information you can get from the bitwise FLAGs:

Hex	Dec	Flag	Description
0x1	1	PAIRED	paired-end (or multiple-segment) sequencing technology
0x2	2	PROPER_PAIR	each segment properly aligned according to the aligner
0x4	4	UNMAP	segment unmapped
0x8	8	MUNMAP	next segment in the template unmapped
0x10	16	REVERSE	SEQ is reverse complemented
0x20	32	MREVERSE	SEQ of the next segment in the template is reversed
0x40	64	READ1	the first segment in the template
0x80	128	READ2	the last segment in the template
0x100	256	SECONDARY	secondary alignment
0x200	512	QCFAIL	not passing quality controls
0x400	1024	DUP	PCR or optical duplicate
0x800	2048	SUPPLEMENTARY	supplementary alignment

For example, if you have an alignment with FLAG set to 113, this can only be represented by decimal codes `64 + 32 + 16 + 1`, so we know that these four flags apply to the alignment and the alignment is paired-end, reverse complemented, sequence of the next template/mate of the read is reversed and the read aligned is the first segment in the template.

Primary, secondary and supplementary alignments A read that aligns to a single reference sequence (including insertions, deletions, skips and clipping but not direction changes), is a **linear alignment**. If a read cannot be represented as a linear alignment, but instead is represented as a group of linear alignments without large overlaps, it is called a **chimeric alignment**. These can for instance be caused by structural variations. Usually, one of the linear alignments in a chimeric alignment is considered to be the **representative** alignment, and the others are called **supplementary**.

Sometimes a read maps equally well to more than one spot. In these cases, one of the possible alignments is marked as the **primary** alignment and the rest are marked as **secondary** alignments.

2.4 BAM

BAM (Binary Alignment/Map) format, is a compressed binary version of SAM. This means that, while SAM is human readable, BAM is only readable for computers. BAM files can be viewed using samtools, and will then have the same format as a SAM file. The key features of BAM are:

- Can store alignments from most mappers
- Supports multiple sequencing technologies
- Supports indexing for quick retrieval/viewing
- Compact size (e.g. 112Gbp Illumina = 116GB disk space)
- Reads can be grouped into logical groups e.g. lanes, libraries, samples
- Widely supported by variant calling packages and viewers

Since BAM is a binary format, we can't use the standard UNIX operations directly on this file format. **Samtools** is a set of programs for interacting with SAM and BAM files. Using the samtools view command, print the header of the BAM file:

[ ]:
```
samtools view -H data/NA20538.bam
```

2.4.1 Exercises

Q13: What version of the human assembly was used to perform the alignments? (Hint: Can you spot this somewhere in the @SQ records?)

[ ]:
```
```

Q14: How many lanes are in this BAM file? (Hint: Do you recall what RG represents?)

[ ]:
```
```

Q15: What programs were used to create this BAM file? (Hint: have a look for the program record, @PG)

[ ]:
```
```

Q16: What version of bwa was used to align the reads? (Hint: is there anything in the @PG record that looks like it could be a version tag?)

[ ]:
```
```

The output from running samtools view on a BAM file without any options is a headerless SAM file. This gets printed to STDOUT in the terminal, so we will want to pipe it to something. Let's have a look at the first read of the BAM file:

[ ]:
```
samtools view data/NA20538.bam | head -n 1
```

Q17: What is the name of the first read? (Hint: have a look at the alignment section if you can't recall the different fields)

[ ]:
```
```

Q18: What position does the alignment of the read start at?

[ ]:
```
```

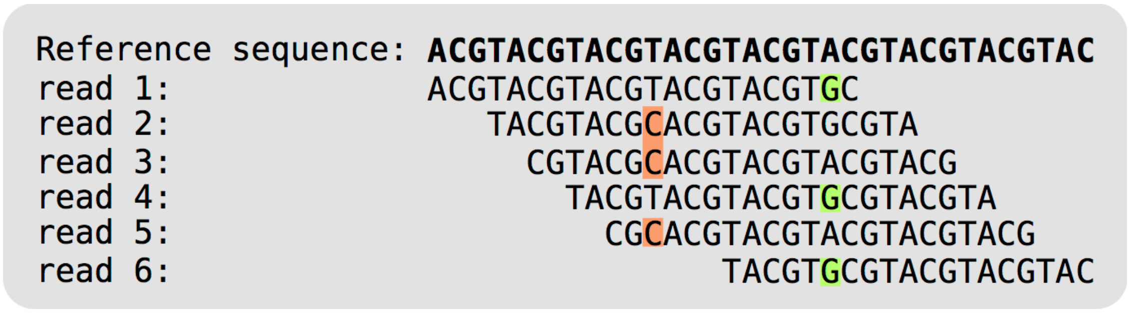
2.5 CRAM

Even though BAM files are compressed, they are still very large. Typically they use 1.5-2 bytes for each base pair of sequencing data that they contain, and while disk capacity is ever improving, increases in disk capacity are being far outstripped by sequencing technologies.

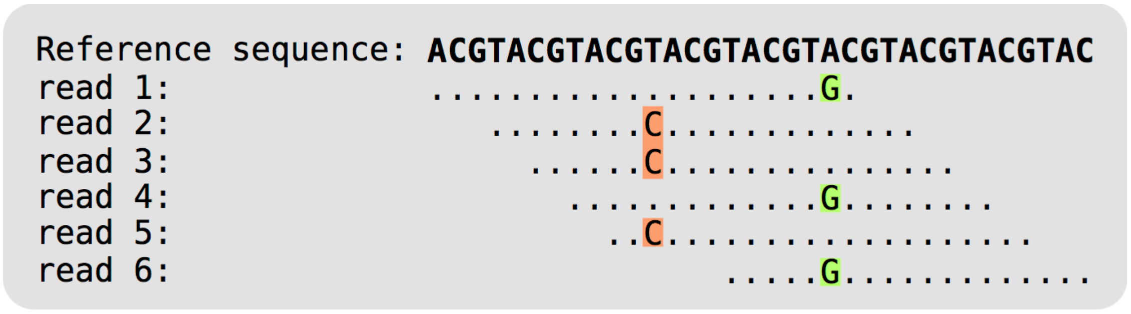
BAM stores all of the data, this includes every read base, every base quality, and it uses a single conventional compression technique for all types of data. CRAM was designed for better compression of genomic data than SAM/BAM. CRAM uses three important concepts:

- Reference based compression
- Controlled loss of quality information
- Different compression methods to suit the type of data, e.g. base qualities vs. metadata vs. extra tags

The figure below displays how reference-based compression works. Instead of saving all the bases of all the reads, only the nucleotides that differ from the reference, and their positions, are kept.



```
Reference sequence: ACGTACGTACGTACGTACGTACGTACGTACGTAC
read 1:             ACGTACGTACGTACGTACGTGC
read 2:                TACGTACGCACGTACGTGCGTA
read 3:                  CGTACGCACGTACGTACGTACG
read 4:                    TACGTACGTACGTGCGTACGTA
read 5:                      CGCACGTACGTACGTACGTACG
read 6:                            TACGTGCGTACGTACGTAC
```



```
Reference sequence: ACGTACGTACGTACGTACGTACGTACGTACGTAC
read 1:             ....................G.
read 2:                ........C.............
read 3:                  ......C...............
read 4:                    .............G........
read 5:                      ..C...................
read 6:                            .....G.............
```

In lossless (no information is lost) mode a CRAM file is 60% of the size of a BAM file, so archives and sequencing centres have moved from BAM to CRAM.

Since samtools 1.3, CRAM files can be read in the same way that BAM files can. We will look closer at how you can convert between SAM, BAM and CRAM formats in the next section.

2.6 Indexing

To allow for fast random access of regions in BAM and CRAM files, they can be indexed. The files must first be coordinate-sorted rather that sorted by read name. This can be done using **samtools sort**. If no options are supplied, it will by default sort by the left-most position of the reference.

[ ]:
```
samtools sort -o data/NA20538_sorted.bam data/NA20538.bam
```

Now we can use **samtools index** to create an index file (.bai) for our sorted BAM file:

[ ]:
```
samtools index data/NA20538_sorted.bam
```

To look for reads mapped to a specific region, we can use **samtools view** and specify the region we are interested in as: RNAME[:STARTPOS[-ENDPOS]]. For example, to look at all the reads mapped to a region called chr4, we could use:

```
samtools view alignment.bam chr4
```

To look at the region on chr4 beginning at position 1,000,000 and ending at the end of the chromosome, we can do:

```
samtools view alignment.bam chr4:1000000
```

And to explore the 1001bp long region on chr4 beginning at position 1,000 and ending at position 2,000, we can use:

```
samtools view alignment.bam chr4:1000-2000
```

2.6.1 Exercises

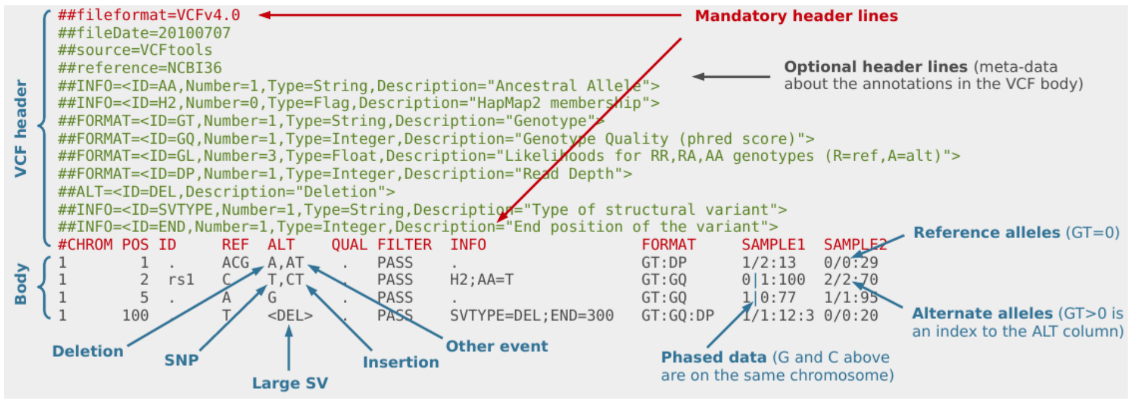
Q19: How many reads are mapped to region 20025000-20030000 on chromosome 1?

[ ]:
```

```

2.7 VCF

The VCF file format was introduced to store variation data. VCF consists of tab-delimited text and is parsable by standard UNIX commands which makes it flexible and user-extensible. The figure below provides an overview of the different components of a VCF file:



```
##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
```

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)
Deletion
SNP
Large SV
Insertion
Other event
Phased data (G and C above are on the same chromosome)

2.7.1 VCF header

The VCF header consists of meta-information lines (starting with `##`) and a header line (starting with `#`). All meta-information lines are optional and can be put in any order, except for *fileformat*. This holds the information about which version of VCF is used and must come first.

The meta-information lines consist of key=value pairs. Examples of meta-information lines that can be included are ##INFO, ##FORMAT and ##reference. The values can consist of multiple fields enclosed by `<>`. More information about these fields is available in the VCF specification `http://samtools.github.io/hts-specs/VCFv4.3.pdf`. This can be accessed using a web browser and there is a copy in the QC directory.

Header line The header line starts with `#` and consists of 8 required fields:

1. CHROM: an identifier from the reference genome
2. POS: the reference position
3. ID: a list of unique identifiers (where available)
4. REF: the reference base(s)
5. ALT: the alternate base(s)
6. QUAL: a phred-scaled quality score
7. FILTER: filter status
8. INFO: additional information

If the file contains genotype data, the required fields are also followed by a FORMAT column header, and then a number of sample IDs. The FORMAT field specifies the data types and order. Some examples of these data types are:

- GT: Genotype, encoded as allele values separated by either / or |
- DP: Read depth at this position for this sample
- GQ: Conditional genotype quality, encoded as a phred quality

2.7.2 Body

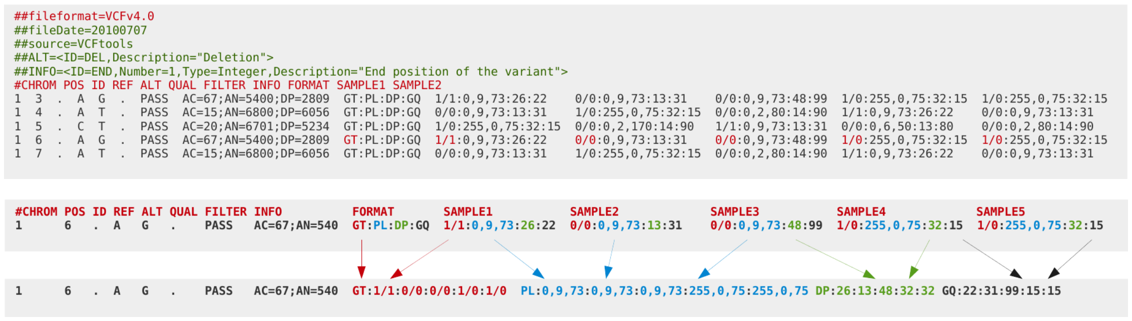
In the body of the VCF, each row contains information about a position in the genome along with genotype information on samples for each position, all according to the fields in the header line.

2.8 BCF

BCF is a compressed binary representation of VCF.

VCF can be compressed with BGZF (bgzip) and indexed with TBI or CSI (tabix), but even compressed it can still be very big. For example, a compressed VCF with 3781 samples of human data will be 54 GB for chromosome 1, and 680 GB for the whole genome. VCFs can also be slow to parse, as text conversion is slow. The main bottleneck is the “FORMAT” fields. For this reason the BCF format was developed.

In BCF files the fields are rearranged for fast access. The following images show the process of converting a VCF file into a BCF file.



```
##fileformat=VCFv4.0
##fileDate=20100707
##source=VCFtools
##ALT=<ID=DEL,Description="Deletion">
##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  3  .  A  G  .  PASS  AC=67;AN=5400;DP=2809  GT:PL:DP:GQ  1/1:0,9,73:26:22    0/0:0,9,73:13:31    0/0:0,9,73:48:99  1/0:255,0,75:32:15  1/0:255,0,75:32:15
1  4  .  A  T  .  PASS  AC=15;AN=6800;DP=6056  GT:PL:DP:GQ  0/0:0,9,73:13:31    1/0:255,0,75:32:15  0/0:0,2,80:14:90  1/1:0,9,73:26:22    0/0:0,9,73:13:31
1  5  .  C  T  .  PASS  AC=20;AN=6701;DP=5234  GT:PL:DP:GQ  1/0:255,0,75:32:15  0/0:0,2,170:14:90   1/1:0,9,73:13:31  0/0:0,6,50:13:80    0/0:0,2,80:14:90
1  6  .  A  G  .  PASS  AC=67;AN=5400;DP=2809  GT:PL:DP:GQ  1/1:0,9,73:26:22    0/0:0,9,73:13:31    0/0:0,9,73:48:99  1/0:255,0,75:32:15  1/0:255,0,75:32:15
1  7  .  A  T  .  PASS  AC=15;AN=6800;DP=6056  GT:PL:DP:GQ  0/0:0,9,73:13:31    1/0:255,0,75:32:15  0/0:0,2,80:14:90  1/1:0,9,73:26:22    0/0:0,9,73:13:31
```

```
#CHROM POS ID REF ALT QUAL FILTER INFO          FORMAT       SAMPLE1           SAMPLE2             SAMPLE3           SAMPLE4             SAMPLE5
1      6   .  A   G   .    PASS   AC=67;AN=540  GT:PL:DP:GQ  1/1:0,9,73:26:22  0/0:0,9,73:13:31    0/0:0,9,73:48:99  1/0:255,0,75:32:15  1/0:255,0,75:32:15
```

```
1      6   .  A   G   .    PASS   AC=67;AN=540  GT:1/1:0/0:0/0:1/0:1/0  PL:0,9,73:0,9,73:0,9,73:255,0,75:255,0,75 DP:26:13:48:32:32 GQ:22:31:99:15:15
```

Bcftools comprises a set of programs for interacting with VCF and BCF files. It can be used to convert between VCF and BCF and to view or extract records from a region.

2.8.1 bcftools view

Let’s have a look at the header of the file 1kg.bcf in the data directory. Note that bcftools uses **-h** to print only the header, while samtools uses **-H** for this.

[ ]:
```
bcftools view -h data/1kg.bcf
```

Similarly to BAM, BCF supports random access, that is, fast retrieval from a given region. For this, the file must be indexed:

[ ]:
```
bcftools index data/1kg.bcf
```

Now we can extract all records from the region 20:24042765-24043073, using the **-r** option. The **-H** option will make sure we don’t include the header in the output:

[ ]:
```
bcftools view -H -r 20:24042765-24043073 data/1kg.bcf
```

2.8.2 bcftools query

The versatile **bcftools query** command can be used to extract any VCF field. Combined with standard UNIX commands, this gives a powerful tool for quick querying of VCFs. Have a look at the usage options:

[ ]:
```
bcftools query -h
```

Let’s try out some useful options. As you can see from the usage, **-l** will print a list of all the samples in the file. Give this a go:

[ ]:
```
bcftools query -l data/1kg.bcf
```

Another very useful option is **-s** which allows you to extract all the data relating to a particular sample. This is a common option meaning it can be used for many bcftools commands, like `bcftools view`. Try this for sample HG00131:

[ ]:
```
bcftools view -s HG00131 data/1kg.bcf | head -n 50
```

The format option, **-f** can be used to select what gets printed from your query command. For example, the following will print the position, reference base and alternate base for sample HG00131, separated by tabs:

[ ]:
```
bcftools query -f'%POS\t%REF\t%ALT\n' data/1kg.bcf | head
```

Finally, let's look at the **-i** option. With this option we can select only sites for which a particular expression is true. For instance, if we only want to look at sites that have at least 2 alternate alleles across all samples, we can use the following expression (piped to **head** to only show a subset of the output):

[ ]:
```
bcftools query -f'%CHROM\t%POS\n' -i 'AC[0]>2' data/1kg.bcf | head
```

We use **-i** with the expression `AC[0]>2`. AC is an info field that holds the ___a___llele ___c___ount. Some fields can hold multiple values, so we use `AC[0]>2` to indicate that we are looking for the first value (this is zero indexed, and hence starts at 0 instead of 1), and that this value should be > 2. To format our output, we use **-f** to specify that we want to print the chromosome name and position.

There is more information about expressions on the bcftools manual page `http://samtools.github.io/bcftools/bcftools.html#expressions`

2.8.3 Exercises

Now, try and answer the following questions about the file 1kg.bcf in the data directory. For more information about the different usage options you can open the bcftools query manual page `http://samtools.github.io/bcftools/bcftools.html#query` in a web browser.

Q20: What version of the human assembly do the coordinates refer to?

[ ]:

Q21: How many samples are there in the BCF?

[ ]:

Q22: What is the genotype of the sample HG00107 at the position 20:24019472? (Hint: use the combination of -r, -s, and -f options)

[ ]:

Q23: How many positions are there with more than 10 alternate alleles? (Hint: use the -i filtering option)

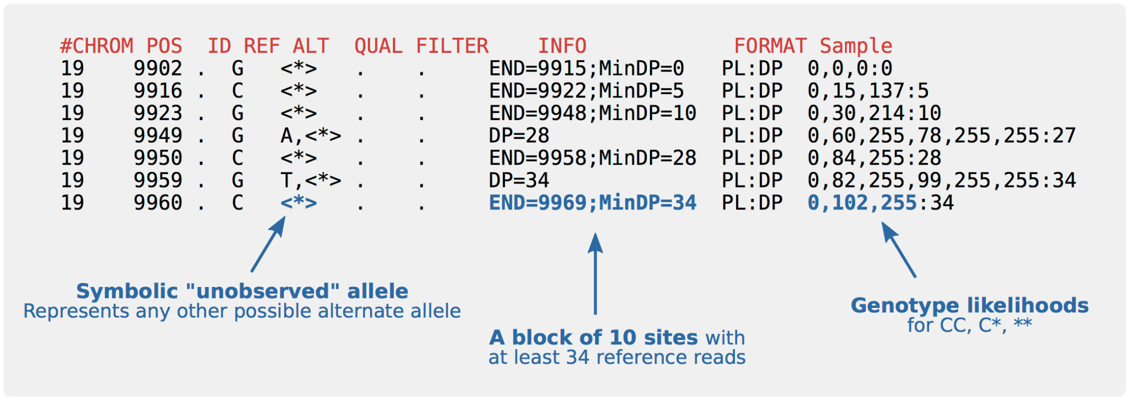
[ ]:

Q24: In how many positions does HG00107 have a non-reference genotype and a read depth bigger than 10? (Hint: you can use pipes to combine bcftools queries)

[ ]:

2.9 gVCF

Often it is not enough to know variant sites only. For instance, we don't know if a site was dropped because it matches the reference or because the data is missing. We sometimes need evidence for both variant and non-variant positions in the genome. In gVCF format, blocks of reference-only sites can be represented in a single record using the "INFO/END" tag. Symbolic alleles (<*>) are used for incremental calling:



```
#CHROM POS  ID REF ALT  QUAL FILTER    INFO            FORMAT Sample
19     9902 .  G   <*>   .    .     END=9915;MinDP=0   PL:DP  0,0,0:0
19     9916 .  C   <*>   .    .     END=9922;MinDP=5   PL:DP  0,15,137:5
19     9923 .  G   <*>   .    .     END=9948;MinDP=10  PL:DP  0,30,214:10
19     9949 .  G   A,<*> .    .     DP=28              PL:DP  0,60,255,78,255,255:27
19     9950 .  C   <*>   .    .     END=9958;MinDP=28  PL:DP  0,84,255:28
19     9959 .  G   T,<*> .    .     DP=34              PL:DP  0,82,255,99,255,255:34
19     9960 .  C   <*>   .    .     END=9969;MinDP=34  PL:DP  0,102,255:34
```

Symbolic "unobserved" allele
Represents any other possible alternate allele
A block of 10 sites with
at least 34 reference reads
Genotype likelihoods
for CC, C*, **

2.9.1 Exercises

Q25: In the above example, what is the size of the reference-only block starting at position 9923?

[ ]:

Q26: For the same block, what is the first base?

[ ]:

Q27: How many reference reads does the block have?

[ ]:

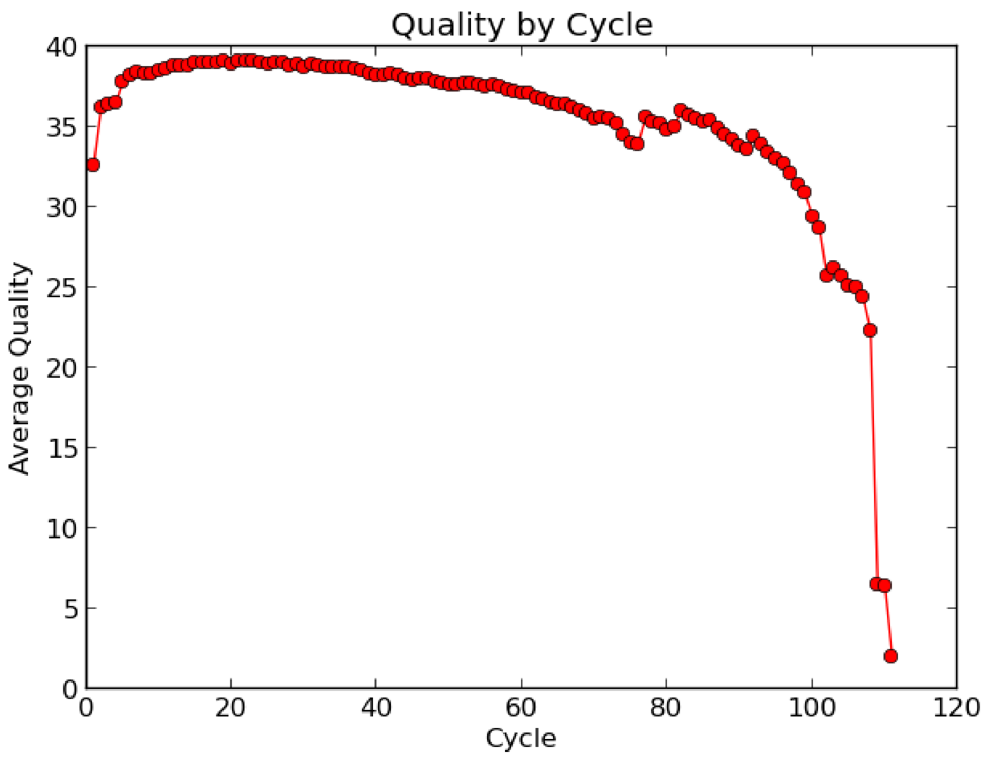
Now continue to the next section of the tutorial: QC assessment of NGS data.

3 QC assessment of NGS data

QC is an important part of any analysis. In this section we are going to look at some of the metrics and graphs that can be used to assess the QC of NGS data.

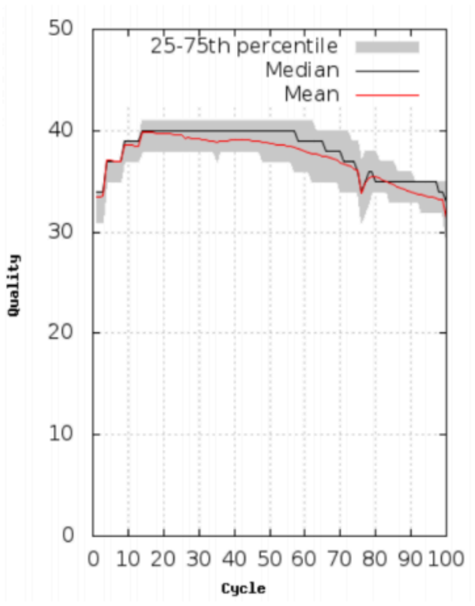
3.1 Base quality

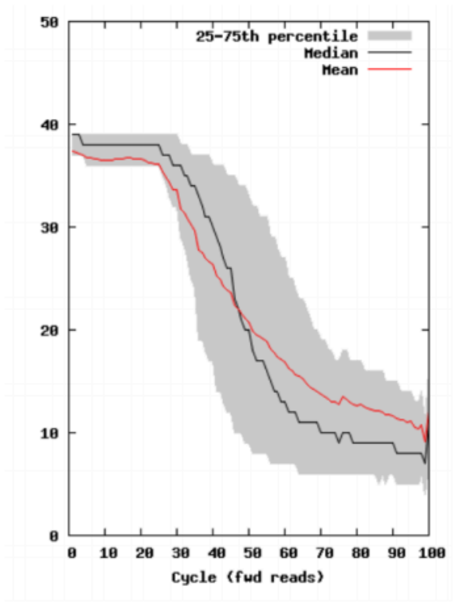
Illumina sequencing technology relies on sequencing by synthesis. One of the most common problems with this is **dephasing**. For each sequencing cycle, there is a possibility that the replication machinery slips and either incorporates more than one nucleotide or perhaps misses to incorporate one at all. The more cycles that are run (i.e. the longer the read length gets), the greater the accumulation of these types of errors gets. This leads to a heterogeneous population in the cluster, and a decreased signal purity, which in turn reduces the precision of the base calling. The figure below shows an example of this.



Because of dephasing, it is possible to have high-quality data at the beginning of the read but really low-quality data towards the end of the read. In those cases you can decide to trim off the low-quality reads, for example using a tool called Trimmomatic.

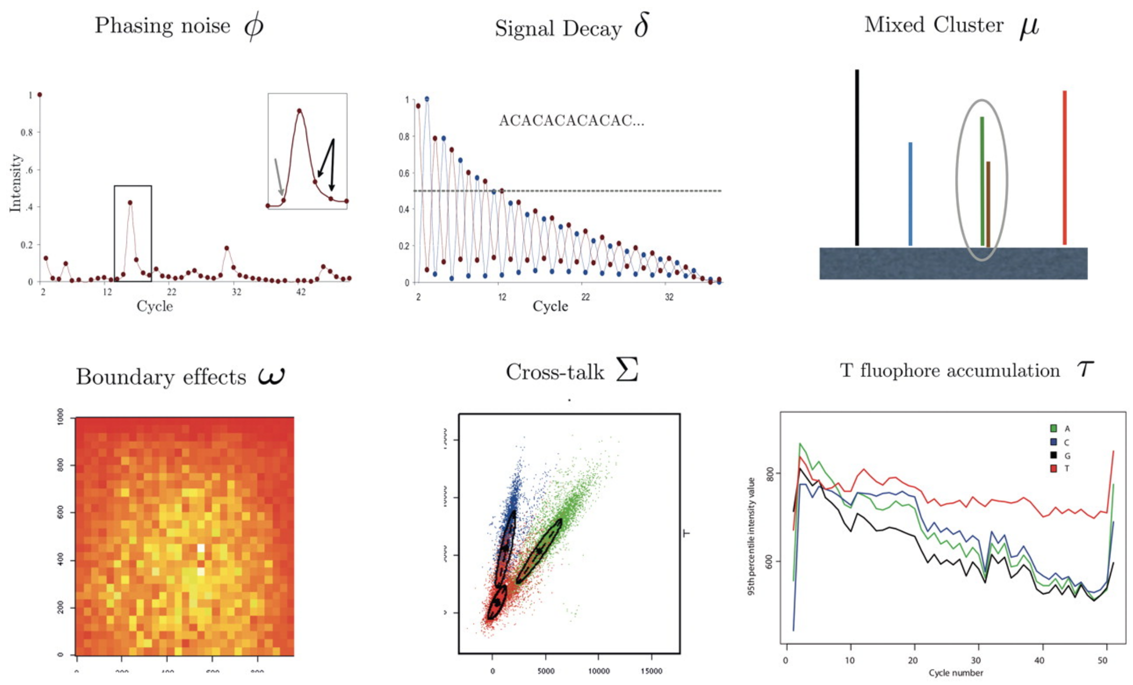
The figures below shows an example of a good sequencing run (left) and a poor sequencing run (right).





3.2 Other base calling errors

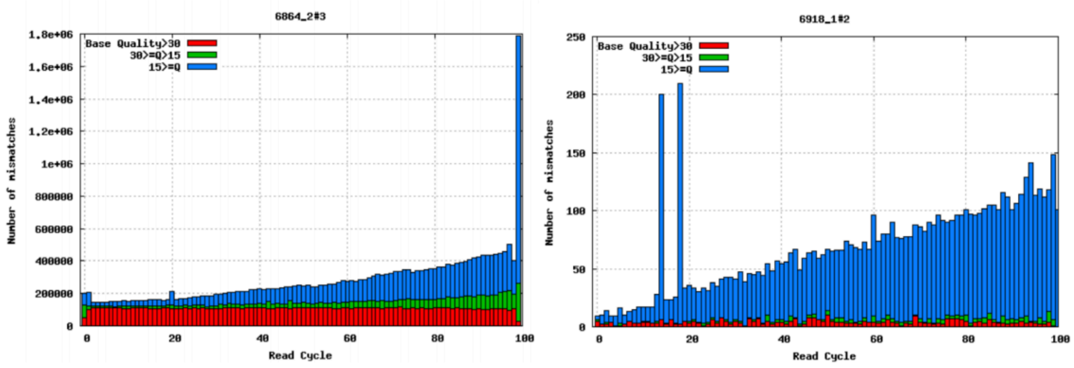
There are several different reasons for a base to be called incorrectly, as shown in the figure below. **Phasing noise** and **signal decay** is a result of the dephasing issue described above. During library preparation, **mixed clusters** can occur if multiple templates get co-located. These clusters should be removed from the downstream analysis. **Boundary effects** occur due to optical effects when the intensity is uneven across each tile, resulting in higher intensity found toward the center. **Cross-talk** occurs because the emission frequency spectra for each of the four base dyes partly overlap, creating uncertainty. Finally, for previous sequencing cycle methods **T fluorophore accumulation** was an issue, where incomplete removal of the dye coupled to thymine lead to an ambient accumulation the nucleotides, causing a false high Thymine trend.



Base-calling for next-generation sequencing platforms, doi: 10.1093/bib/bbq077

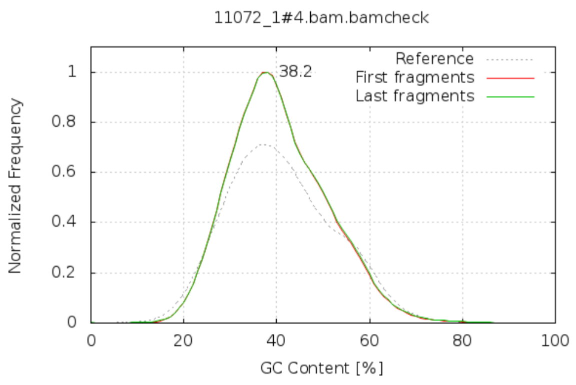
3.3 Mismatches per cycle

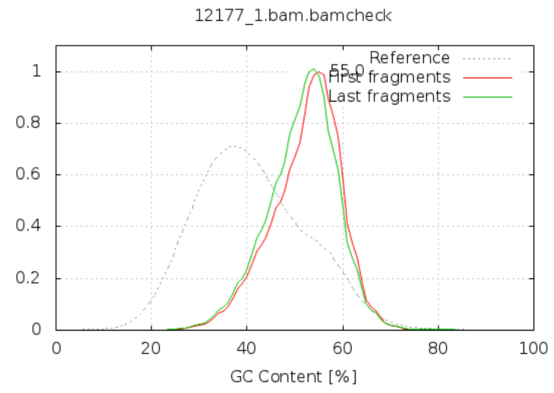
Aligning reads to a high-quality reference genome can provide insight to the quality of a sequencing run by showing you the mismatches to the reference sequence. This can help you detect cycle-specific errors. Mismatches can occur due to two main causes, sequencing errors and differences between your sample and the reference genome, which is important to bear in mind when interpreting mismatch graphs. The figure below shows an example of a good run (left) and a bad one (right). In the graph on the left, the distribution of the number of mismatches is even between the cycles, which is what we would expect from a good run. However, in the graph on the right, two cycles stand out with a lot of mismatches compared to the other cycles.



3.4 GC content

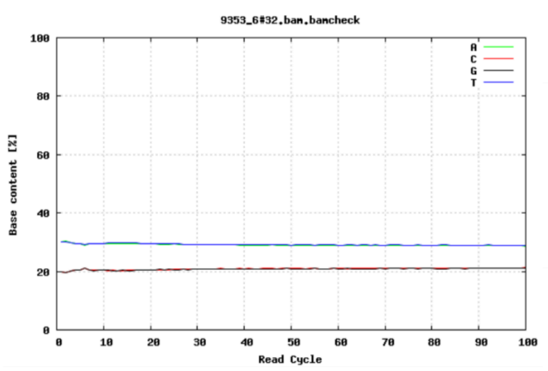
It is a good idea to compare the GC content of the reads against the expected distribution in a reference sequence. The GC content varies between species, so a shift in GC content like the one seen below could be an indication of sample contamination. In the left graph below, we can see that the GC content of the sample is about the same as for the reference, at ~38%. However, in the right graph, the GC content of the sample is closer to 55%, indicating that there is an issue with this sample.

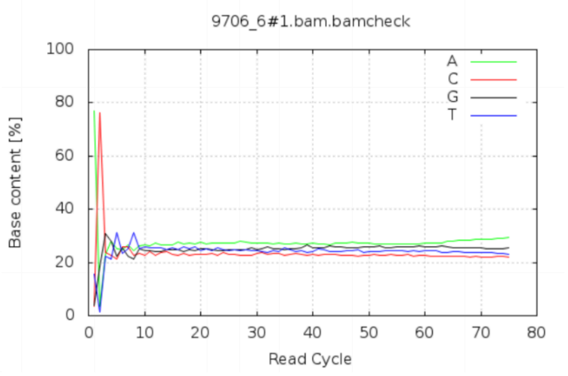




3.5 GC content by cycle

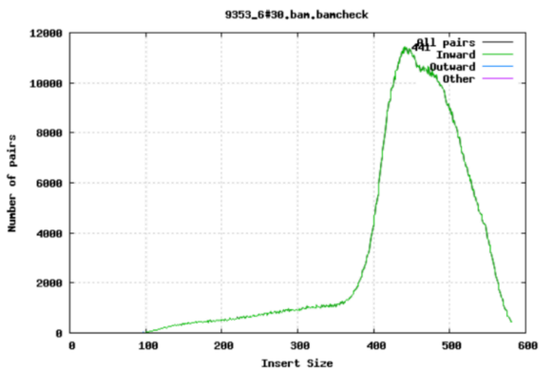
Looking at the GC content per cycle can help detect if the adapter sequence was trimmed. For a random library, it is expected to be little to no difference between the different bases of a sequence run, so the lines in this plot should be parallel with each other like in the graph on the left below. In the graph on the right, the initial spikes are likely due to adapter sequences that have not been removed.

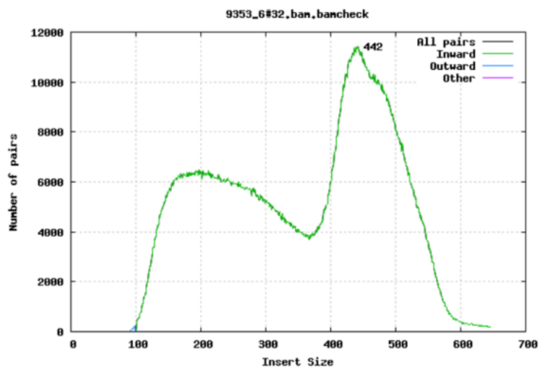




3.6 Fragment size

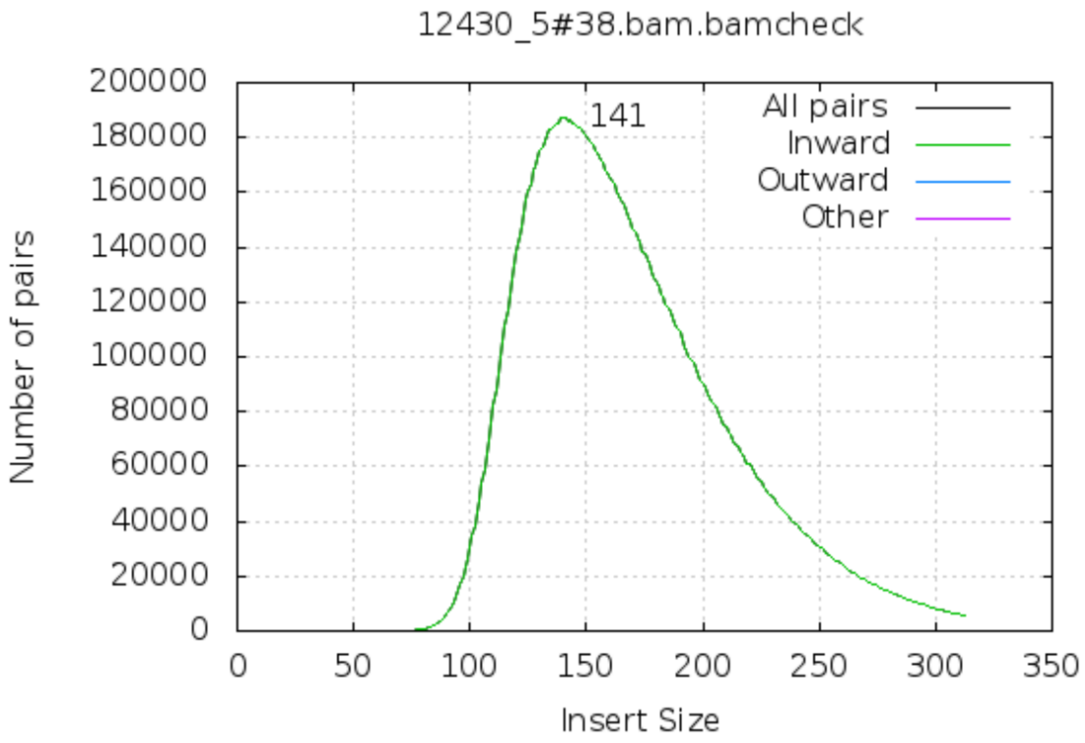
For paired-end sequencing the size of DNA fragments also matters. In the first of the examples below, the fragment size peaks around 440 bp. In the second however, there is also a peak at around 200 bp. This indicates that there was an issue with the fragment size selection during library prep.





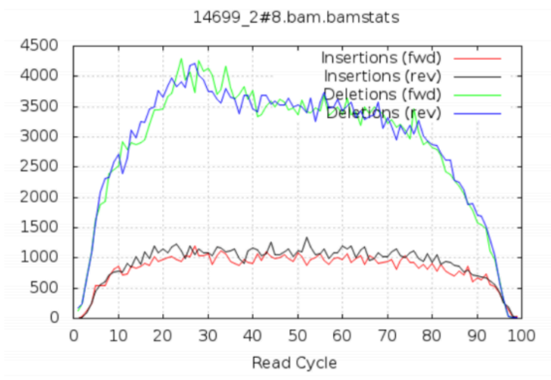
3.6.1 Exercises

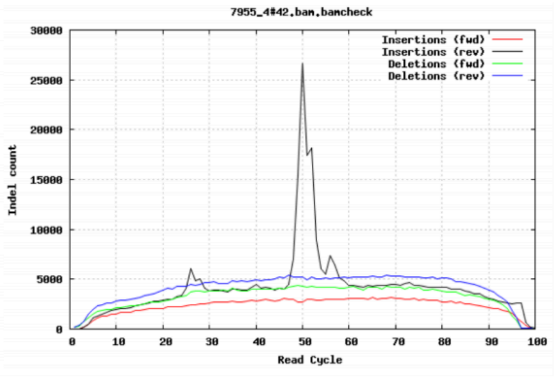
Q1: The figure below is from a 100bp paired-end sequencing. Can you spot any problems?



3.7 Insertions/Deletions per cycle

Sometimes, air bubbles occur in the flow cell, which can manifest as false indels. The spike in the right image provides an example of how this can look.





3.8 Generating QC stats

Now let's try this out! We will generate QC stats for two lanes of Illumina paired-end sequencing data from yeast. The reads have already been aligned to the Saccromyces cerevisiae reference genome to produce the BAM file lane1.sorted.bam.

Now we will use **`samtools stats`** to generate the stats for the primary alignments. The option **`-f`** can be used to filter reads with specific tags, while **`-F`** can be used to *filter out* reads with specific tags. The following command will include only primary alignments:

[ ]:
```
samtools stats -F SECONDARY data/lane1.sorted.bam > data/lane1.sorted.bam.bchk
```

Have a look at the first 47 lines of the statistics file that was generated:

[ ]:
```
head -n 47 data/lane1.sorted.bam.bchk
```

This file contains a number of useful stats that we can use to get a better picture of our data, and it can even be plotted with **`plot-bamstats`**, as you will see soon. First let's have a closer look at some of the different stats. Each part of the file starts with a **#** followed by a description of the section and how to extract it from the file. Let's have a look at all the sections in the file:

[ ]:
```
grep ^'#' data/lane1.sorted.bam.bchk | grep 'Use'
```

3.8.1 Summary Numbers (SN)

This initial section contains a summary of the alignment and includes some general statistics. In particular, you can see how many bases mapped, and how much of the genome that was covered.

3.8.2 Exercises

Now look at the output and try to answer the questions below.

Q2: What is the total number of reads?

[ ]:
```
```

Q3: What proportion of the reads were mapped?

[ ]:
```
```

Q4: How many pairs were mapped to a different chromosome?

[ ]:
```

```

Q5: What is the insert size mean and standard deviation?

[ ]:
```

```

Q6: How many reads were paired properly?

[ ]:
```

```

3.8.3 Generating QC plots

Finally, we will create some QC plots from the output of the stats command using the command **plot-bamstats** which is included in the samtools package:

[ ]:
```
plot-bamstats -p data/lane1-plots/ data/lane1.sorted.bam.bchk
```

Now in your web browser open the file lane1-plots/index.html to view the QC information.

Q7: How many reads have zero mapping quality?

[ ]:
```

```

Q8: Which read (forward/reverse) of the first fragments and second fragments are higher base quality on average?

[ ]:
```

```

Now continue to the next section of the tutorial: File conversion.

4 File conversion

In this section we are going to look at how to convert from one file format to another. There are many tools available for converting between file formats, and we will use some of the most common ones: samtools, bcftools and Picard.

4.1 SAM to BAM

To convert from SAM to BAM format we are going to use the **`samtools view`** command. In this instance, we would like to include the SAM header, so we use the **`-h`** option:

[ ]:
```
samtools view -h data/NA20538.bam > data/NA20538.sam
```

Now, have a look at the first ten lines of the SAM file. They should look like they did in the previous section when you viewed the BAM file header.

[ ]:
```
head data/NA20538.sam
```

Well that was easy! And converting SAM to BAM is just as straightforward. This time there is no need for the `-h` option, however we have to tell samtools that we want the output in BAM format. We do so by adding the `-b` option:

```
[ ]: samtools view -b data/NA20538.sam > data/NA20538_2.bam
```

Samtools is very well documented, so for more usage options and functions, have a look at the samtools manual http://www.htslib.org/doc/samtools-1.0.html.

4.2 BAM to CRAM

The samtools view command can be used to convert a BAM file to CRAM format. In the data directory there is a BAM file called yeast.bam that was created from S. cerevisiae Illumina sequencing data. There is also a reference genome in the directory, called Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa. For the conversion, an index file (.fai) for the reference must be created. This can be done using **`samtools faidx`**. However, as we will see, samtools will generate this file on the fly when we specify a reference file using the `-T` option.

To convert to CRAM, we use the **`-C`** option to tell samtools we want the output as CRAM, and the **`-T`** option to specify what reference file to use for the conversion. We also use the **`-o`** option to specify the name of the output file. Give this a try:

```
[ ]: samtools view -C -T data/Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa -o␣
     ↪data/yeast.cram data/yeast.bam
```

Have a look at what files were created:

```
[ ]: ls -l data
```

As you can see, this has created an index file for the reference genome called Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa.fai and the CRAM file yeast.cram.

4.2.1 Exercises

Q1: Since CRAM files use reference-based compression, we expect the CRAM file to be smaller than the BAM file. What is the size of the CRAM file?

```
[ ]:
```

Q2: Is your CRAM file smaller than the original BAM file?

```
[ ]:
```

To convert CRAM back to BAM, simply change `-C` to `-b` and change places for the input and output CRAM/BAM:

```
samtools view -b -T data/Saccharomyces_cerevisiae.EF4.68.dna.toplevel.fa -o data/yeast.bam data
```

4.3 FASTQ to SAM

SAM format is mainly used to store alignment data, however in some cases we may want to store the unaligned data in SAM format and for this we can use the picard tools **`FastqToSam`** application.

Picard tools is a Java application that comes with a number of useful options for manipulating high-throughput sequencing data. .

To convert the FASTQ files of lane 13681_1#18 to unaligned SAM format, run:

```
[ ]: picard FastqToSam F1=data/13681_1#18_1.fastq.gz F2=data/13681_1#18_2.fastq.gz␣
       ↪O=data/13681_1#18.sam SM=13681_1#18
```

From here you can go on and convert the SAM file to BAM and CRAM, as described previously. There are also multiple options for specifying what metadata to include in the SAM header. To see all available options, run:

```
[ ]: picard FastqToSam -h
```

4.4 CRAM to FASTQ

It is possible to convert CRAM to FASTQ directly using the `samtools fastq` command. However, for many applications we need the fastq files to be ordered so that the order of the reads in the first file match the order of the reads in the mate file. For this reason, we first use `samtools collate` to produce a collated BAM file.

```
[ ]: samtools collate data/yeast.cram data/yeast.collated
```

The newly produced BAM file will be called yeast.collated.bam. Let's use this to create two FASTQ files, one for the forward reads and one for the reverse reads:

```
[ ]: samtools fastq -1 data/yeast.collated_1.fastq -2 data/yeast.collated_2.fastq␣
       ↪data/yeast.collated.bam
```

For further information and usage options, have a look at the samtools manual page (http://www.htslib.org/doc/samtools.html).

4.5 VCF to BCF

In a similar way that samtools view can be used to convert between SAM, BAM and CRAM, **`bcftools view`** can be used to convert between VCF and BCF. To convert the file called 1kg.bcf to a compressed VCF file called 1kg.vcf.gz, run:

```
[ ]: bcftools view -O z -o data/1kg.vcf.gz data/1kg.bcf
```

The **`-O`** option allows us to specify in what format we want the output, compressed BCF (b), uncompressed BCF (u), compressed VCF (z) or uncompressed VCF (v). With the **`-o`** option we can select the name of the output file.

Have a look at what files were generated (the options `-lrt` will list the files in reverse chronological order):

```
[ ]: ls -lrt data
```

This also generated an index file, 1kg.bcf.csi.

To convert a VCF file to BCF, we can run a similar command. If we want to keep the original BCF, we need to give the new one a different name so that the old one is not overwritten:

[ ]:
```
bcftools view -O b -o data/1kg_2.bcf data/1kg.vcf.gz
```

Congratulations you have reached the end of the Data formats and QC tutorial!