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)

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, and also suggest how to identify possible contamination.

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
- Identify possible contamination in high throughput sequence data

1.3 Tutorial sections

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

1.4 Authors

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

1.5 Running the commands from this tutorial

You can follow this tutorial by running all the commands provided on the `bioinfsrv` server. To get started, connect to the `bioinfsrv` server using the instructions provided earlier and type all the commands you see into the terminal window. Remember, the terminal window is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

Start by typing the command below:

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

1.6 Let's get started!

This tutorial assumes that you have samtools, bcftools, fastqc and bactinspector on your computer. The easiest way to do this is by creating a Conda environment called `qc` and install the listed software using conda. This has already been setup on the server for you. To activate the environment use

[ ]:
```
conda activate qc
```

After your environment is activated you can run the following commands:

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

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

[ ]:
```
fastqc --help
```

[ ]:
```
bactinspector -h
```

This should return the help message for these tools.

To get started with the tutorial, head 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 right directory

[ ]:
```
pwd
```

It should display something like

```
/home/username/course_data/data_formats/data
```

Where username will be your username for the server.

2.1 FASTA

The FASTA format is one of the most basic ways to store sequence data, and it can be 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?)

[ ]:
```

```

If you get stuck here, do not spend too much time trying to figure this out and move on. A solution will be provided during the practical session.

2.2 FASTQ

FASTQ is a data format for raw unaligned sequencing reads. It is an extension to the FASTA file format, and includes a quality score for each base. For paired-end sequencing, two FASTQ files are produced. 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 - see https://en.wikipedia.org/wiki/Phred_quality_score

The quality scores range (in theory) from 1 to 94 and are encoded as ASCII characters - see https://en.wikipedia.org/wiki/ASCII). 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%

The following simple perl command will print the quality score value for an ASCII character. Try changing the "A" to another character, for example one from the quality strings above (e.g. @, = or B).

[ ]:
```
perl -e 'printf "%d\n",ord("A")-33;'
```

Something to be aware of is that two different ways of calculating the quality scores have historically been in use. The standard Sanger variant uses the Phred score, while the Solexa pipeline earlier used a different mapping.

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?)

[ ]:
```

```

Again, don't worry if you cannot solve this the solution will be provided during the practical session.

2.3 SAM/BAM

SAM (Sequence Alignment/Map) format was developed by the 1000 Genomes Project group in 2009 and is a unified format for storing read alignments to a reference genome. BAM is the compressed version of SAM. SAM/BAM format is the accepted 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 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

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 unit of sequencing, this can be a lane of sequencing data. 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

From reading section 1.3 of the SAM specification, look at the following line from the header of the SAM/BAM file:

```
@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 expected fragment insert size?

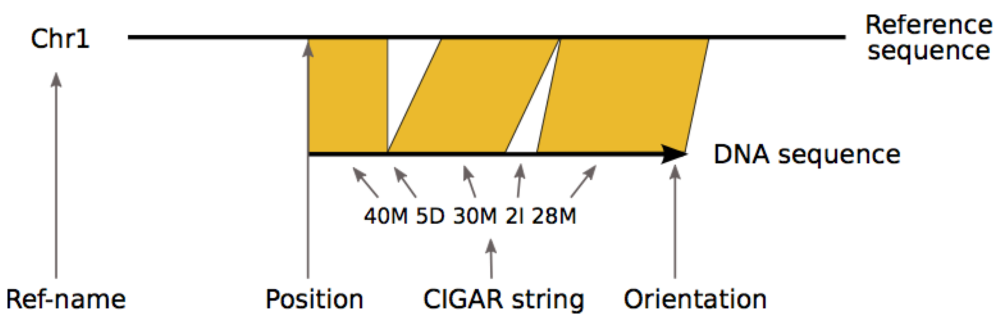
```
[ ]:
```

2.3.3 Alignment Section

The alignment section of SAM files contains one line per fragment alignment, which in turn contains the columns listed below. The first 11 columns are mandatory.

1. QNAME: Query NAME of the read or the read pair
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.



In a SAM file, this image representation could for instance translate to the following entry of 100 bases:

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

2.3.4 Exercises

Let's have a look at example.sam. This file contains only a subset of the alignment section of a BAM-file that we will look closer at soon. Notice that we can use the standard UNIX operations like **cat** on this file.

[ ]:
```
cat 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? (We will go through the meaning of CIGAR strings in the next section)

[ ]:
```

```

Q10: What is the inferred insert size?

[ ]:
```

```

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 to the reference
D	deletion from the reference
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.

2.3.8 BAM

BAM (Binary Alignment/Map) format, is a binary version of SAM. This means that, while SAM is human readable, BAM is only readable for computers. BAM was developed for fast processing and random access. To achieve this, BGZF (Block GZIP) compression is used for indexing. 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

2.3.9 Exercises

Since BAM is a binary format, we can't use the standard Linux operations (cat, less, head, grep etc.) 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 NA20538.bam
```

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 sequencing runs/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 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)

[ ]:
```

```

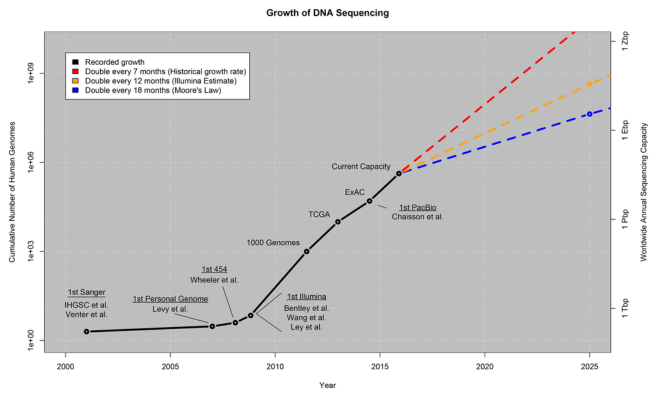
Q18: What position does the alignment of the read start at?

[ ]:
```

```

2.4 CRAM

Even though BAM files are compressed, they are still too 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. Therefore, 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 are now moving 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.

2.5 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. This can be done using **samtools sort**. If no options are supplied, it will by default sort by the left-most position.

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

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

[ ]:
```
samtools index 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, if we wanted 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
```

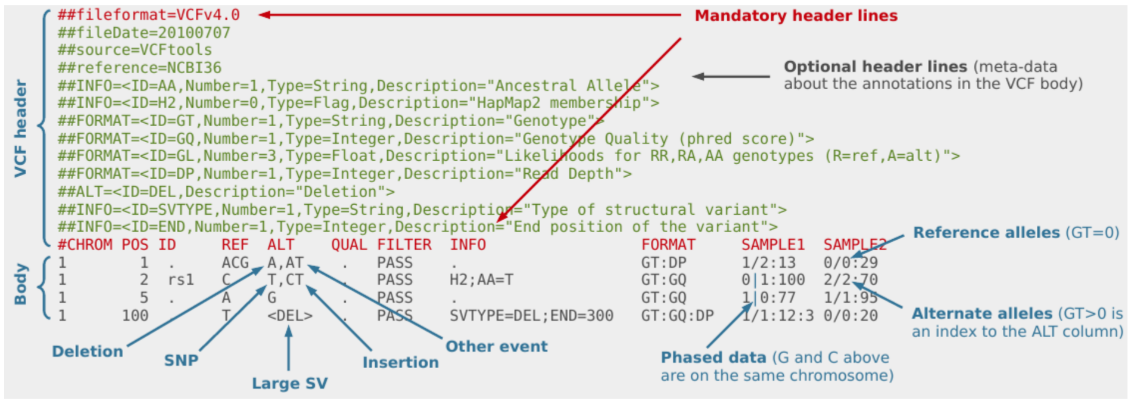
2.5.1 Exercises

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

[ ]:

2.6 VCF/BCF

The VCF file format and its binary version BCF were 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)
Phased data (G and C above are on the same chromosome)
Deletion
SNP
Large SV
Insertion
Other event

2.6.1 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.

2.6.2 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 of / or |
- DP: Read depth at this position for this sample
- GQ: Conditional genotype quality, encoded as a phred quality

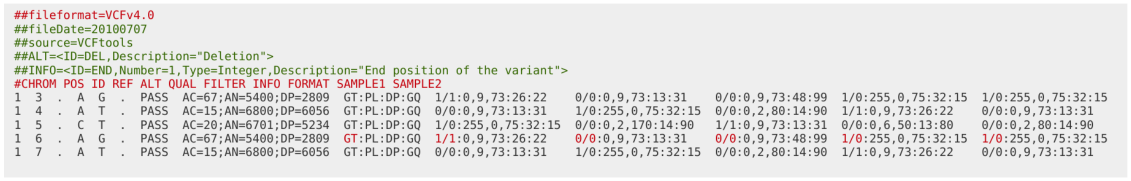
2.6.3 Body

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.

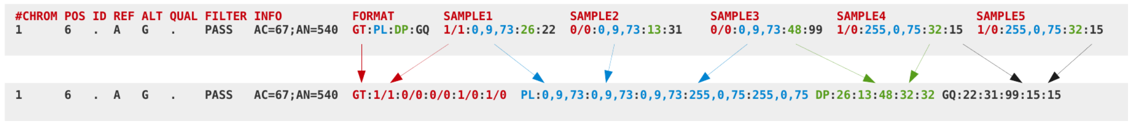
2.6.4 BCF

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, a binary representation of VCF, 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.

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 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 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 1kg.bcf
```

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 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 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' -s HG00131 1kg.bcf | head
```

2.6.5 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 new tab.

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)

[ ]:
```

```

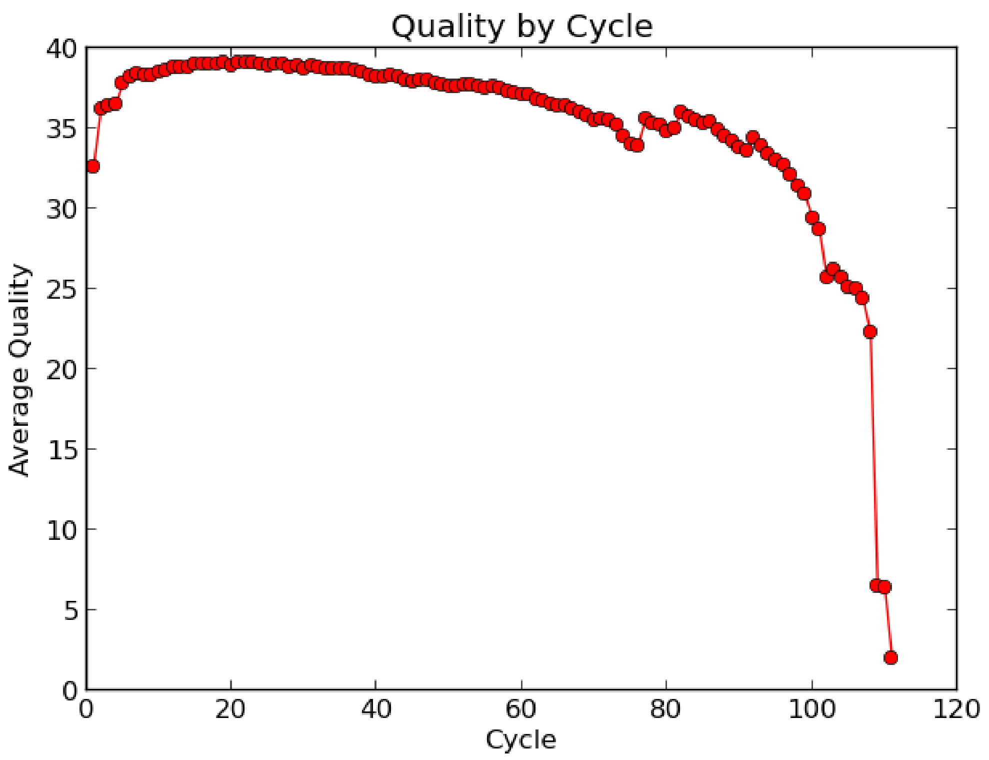
Now continue to the next section of the tutorial: QC assement.

3 QC assessment of NGS data

As mentioned previously, 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.

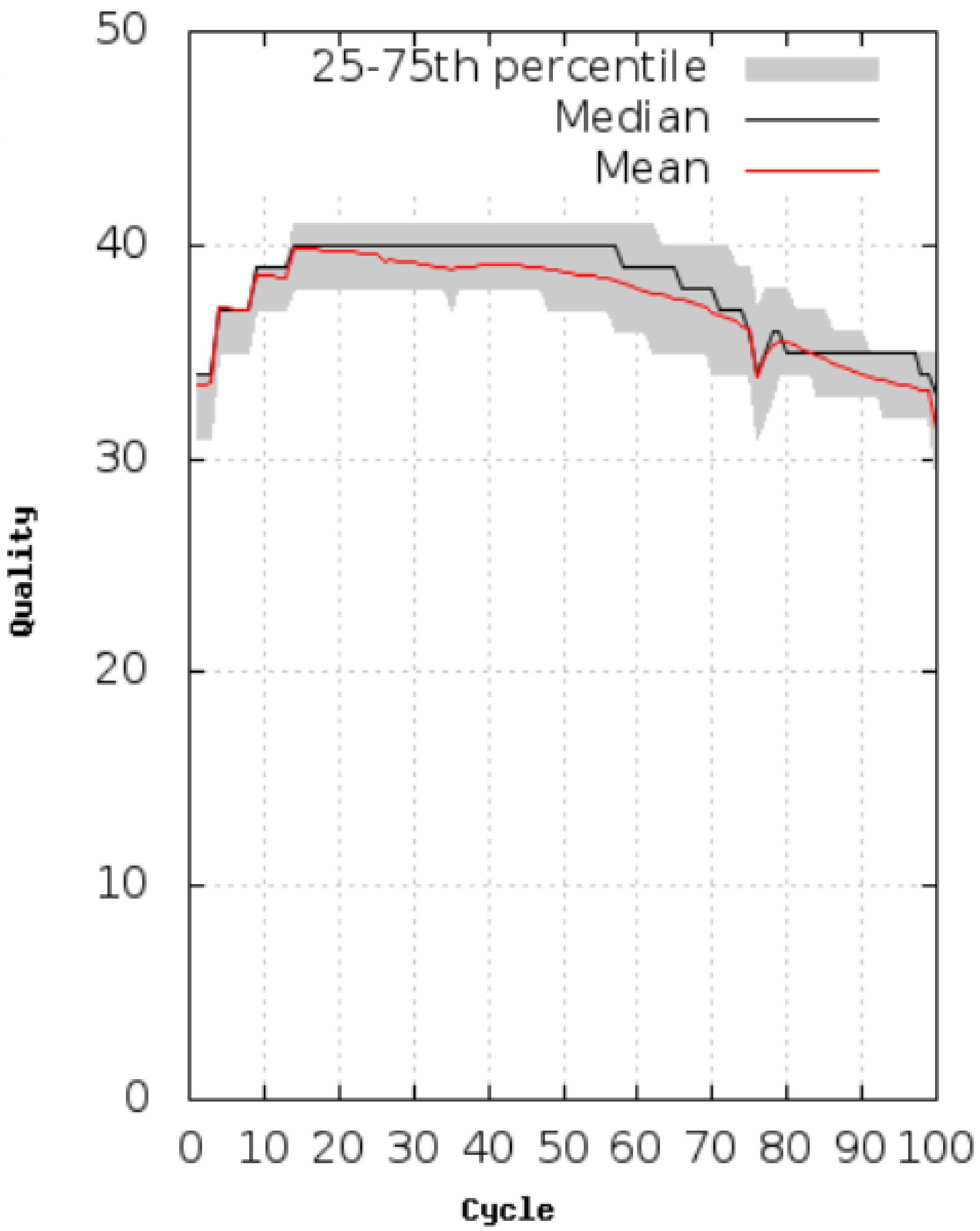
3.1 Base quality

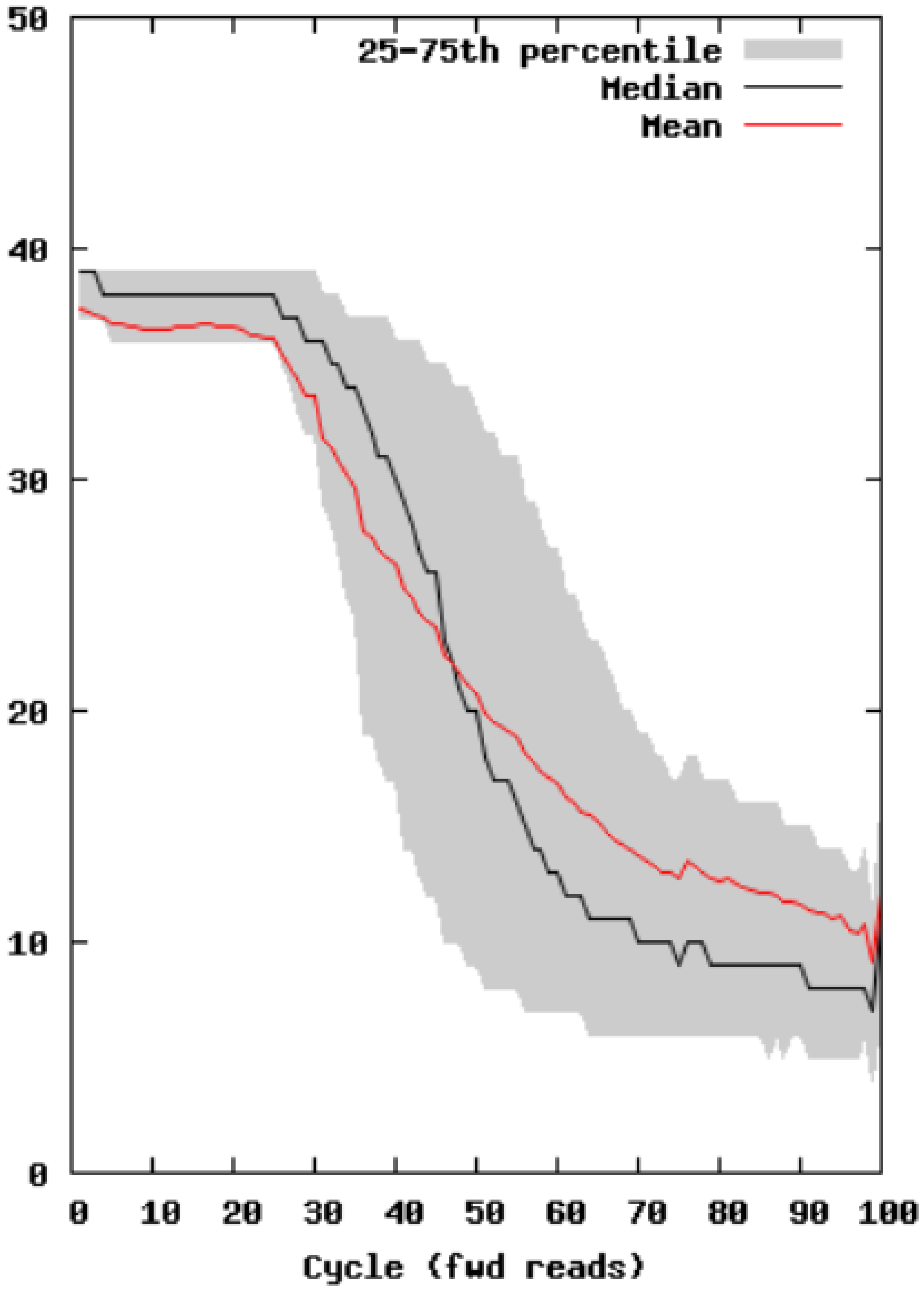
Illumina sequencing technology relies on sequencing by synthesis. One of the most common problems with this is **dephasing**. For each sequencing cycle, there is a possibility that the replication machinery slips and either incorporates more than one nucleotide or perhaps misses to incorporate one at 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.

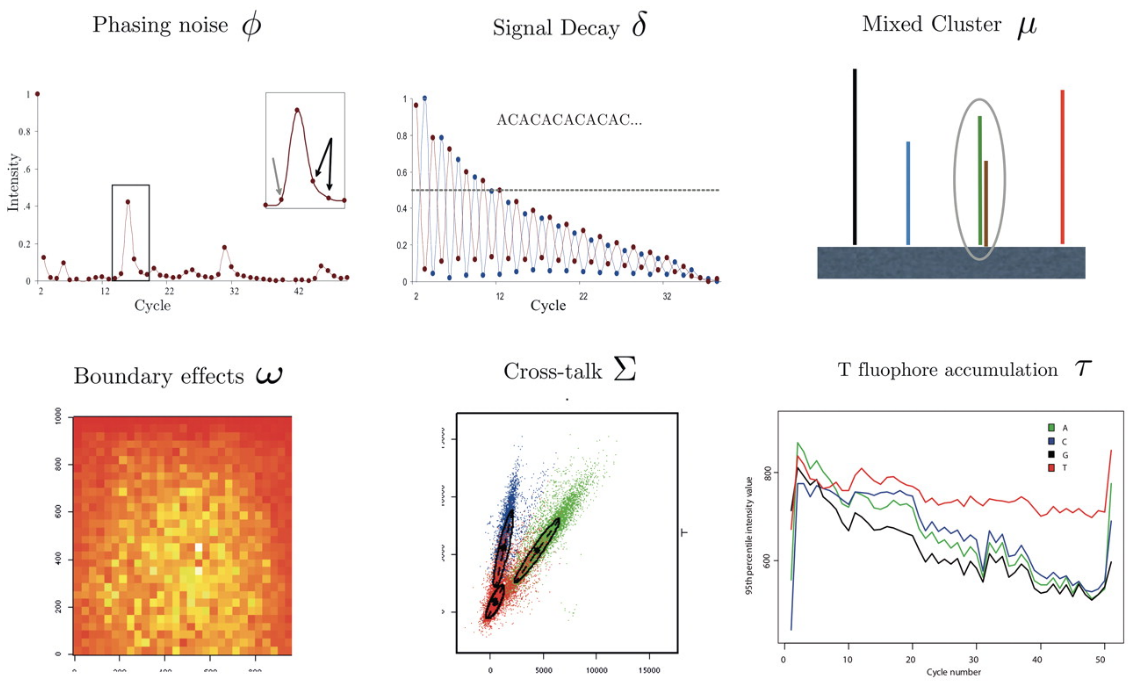
The figures below shows an example of a high-quality read data (top) and a poor quality read data (bottom).


25-75th percentile
Median
Mean
50
40
30
20
10
0
Quality
0
10
20
30
40
50
60
70
80
90
100
Cycle



3.2 Other base calling errors

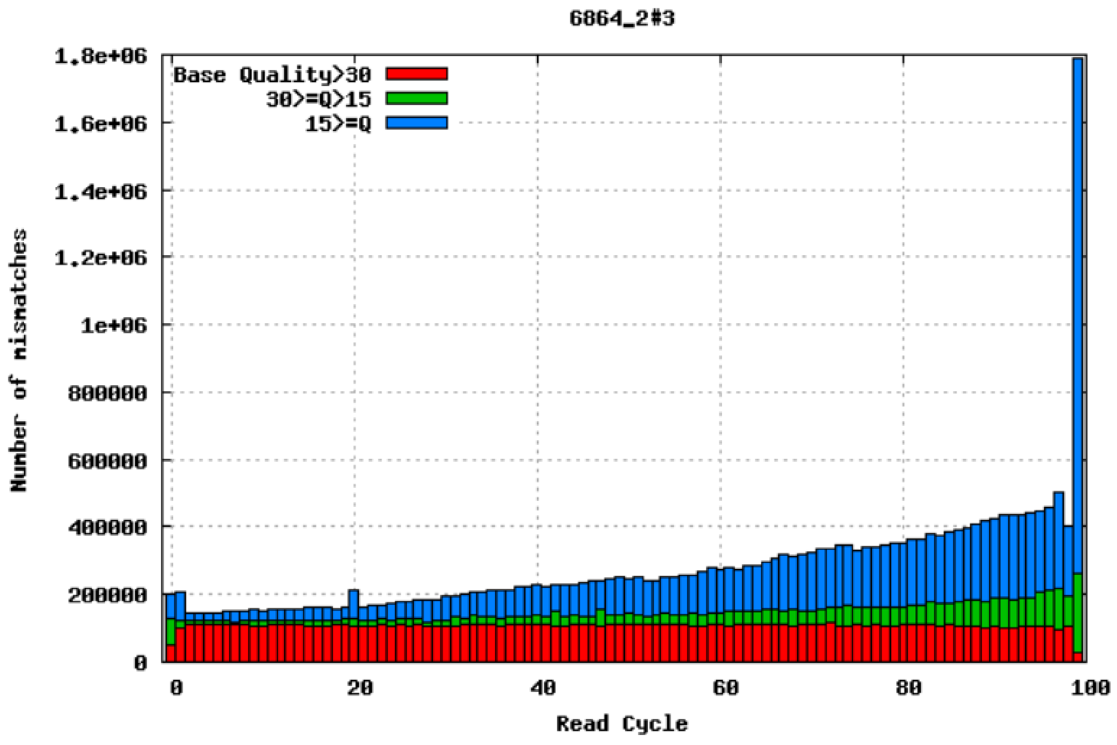
There are several different reasons for a base to be called incorrectly, as shown in the figure below. **Phasing noise** and **signal decay** is a result of the dephasing issue described above. During library preparation, **mixed clusters** can occur if multiple templates get co-located. These clusters 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.

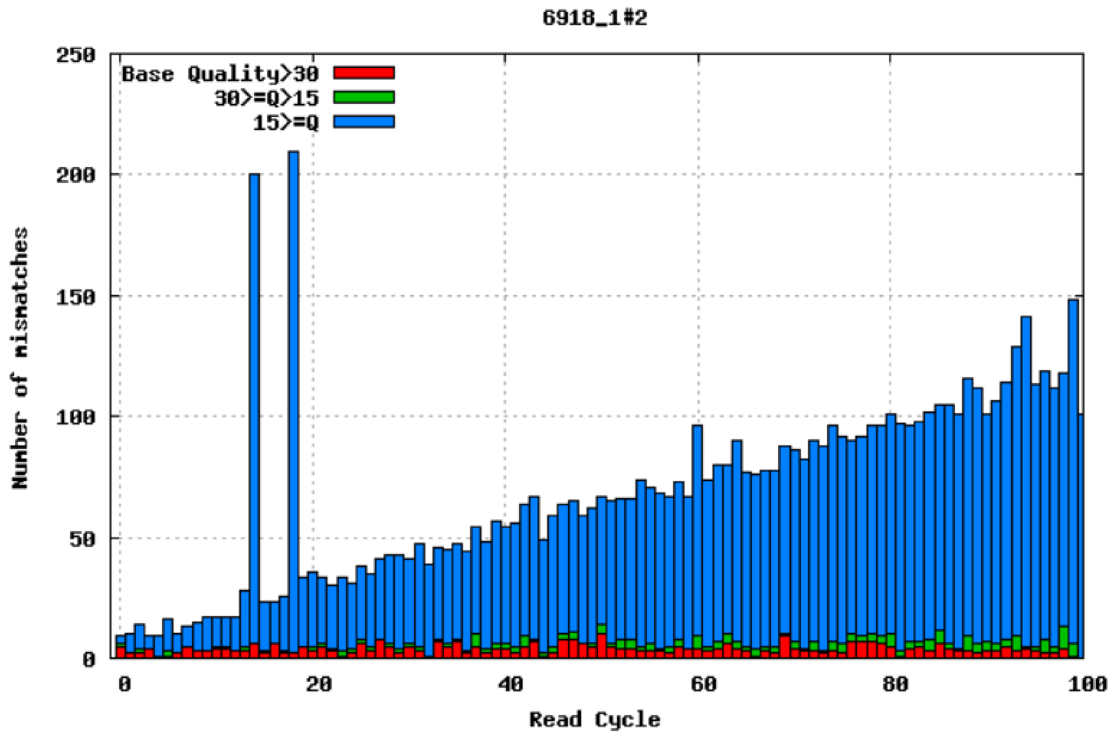


Base-calling for next-generation sequencing platforms, doi: 10.1093/bib/bbq077

3.3 Mismatches per cycle

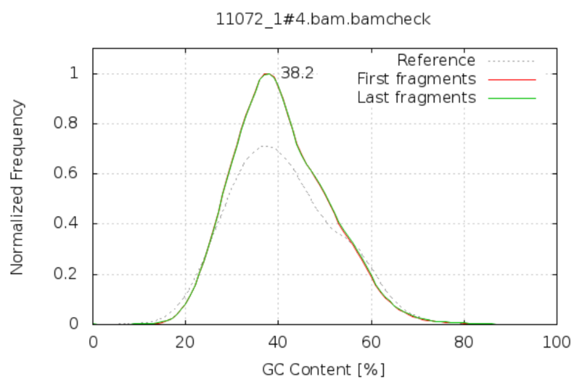
Aligning reads to a high-quality reference genome can provide insight to the quality of a sequencing run by showing you the mismatches to the reference sequence. This can help you detect cycle-specific errors. Mismatches can occur due to two main causes, sequencing errors and differences between your sample and the reference genome, which is important to bear in mind when interpreting mismatch graphs. The figures below show an example of a good run (top) and a bad one (bottom). In the first figure, 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 second figure, two cycles stand out with a lot of mismatches compared to the other cycles.

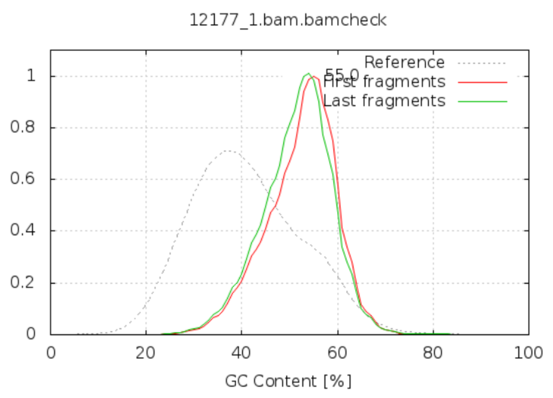

6864_2#3
Base Quality>30
30>=Q>15
15>=Q
Number of mismatches
Read Cycle
1.8e+06
1.6e+06
1.4e+06
1.2e+06
1e+06
800000
600000
400000
200000
0
0
20
40
60
80
100


6918_1#2
Base Quality>30
30>=Q>15
15>=Q
Number of mismatches
Read Cycle
250
200
150
100
50
0
0
20
40
60
80
100

3.4 GC content

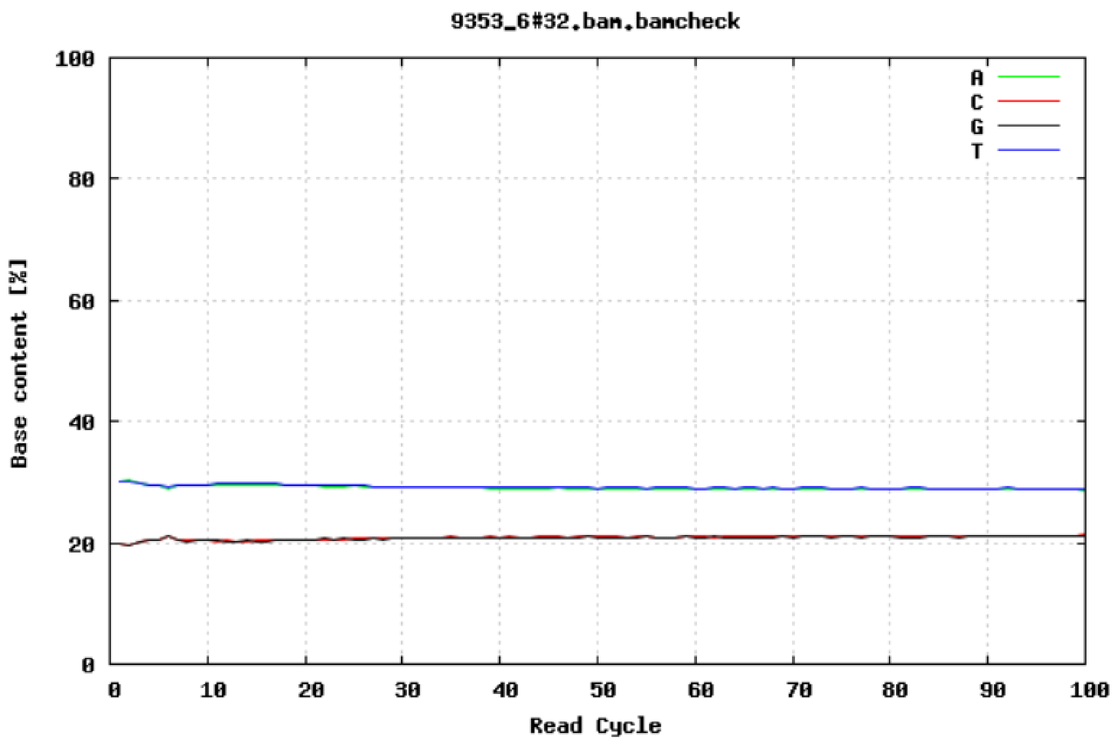
It is a good idea to compare the GC content of the reads against the expected distribution in a reference sequence. The GC content varies between species, so a shift in GC content like the one seen below could be an indication of sample contamination. In the left image below, we can see that the GC content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

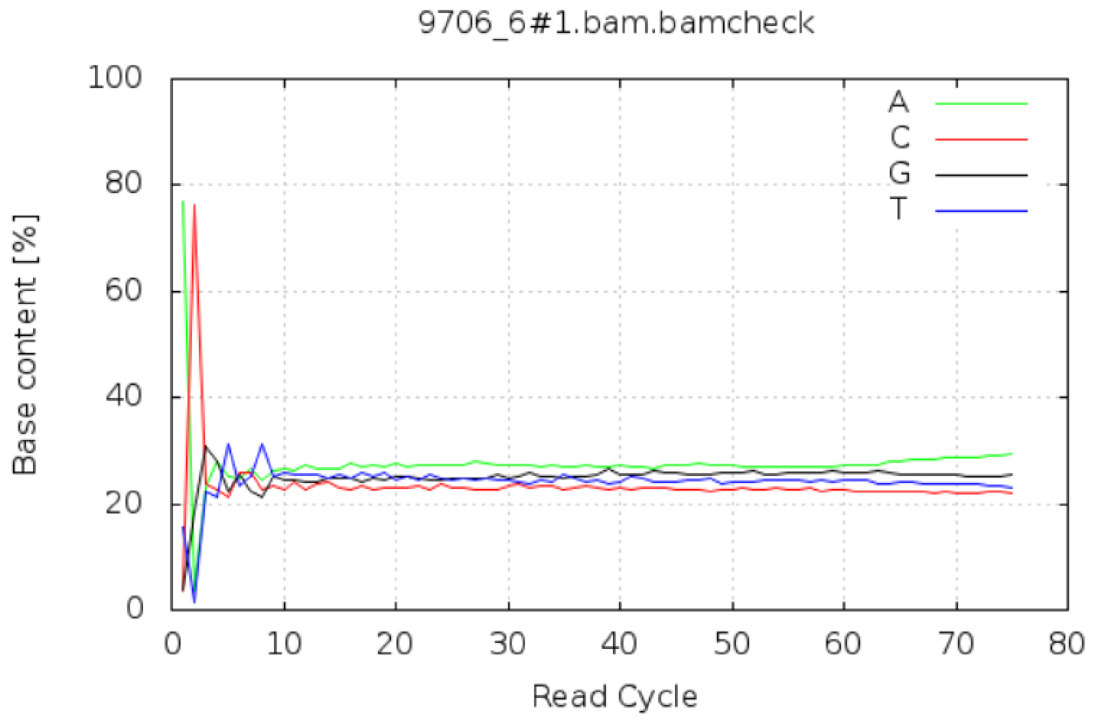




3.5 GC content by cycle

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 first of the two figures below. In the second of the figures, the initial spikes are likely due to adapter sequences that have not been removed.

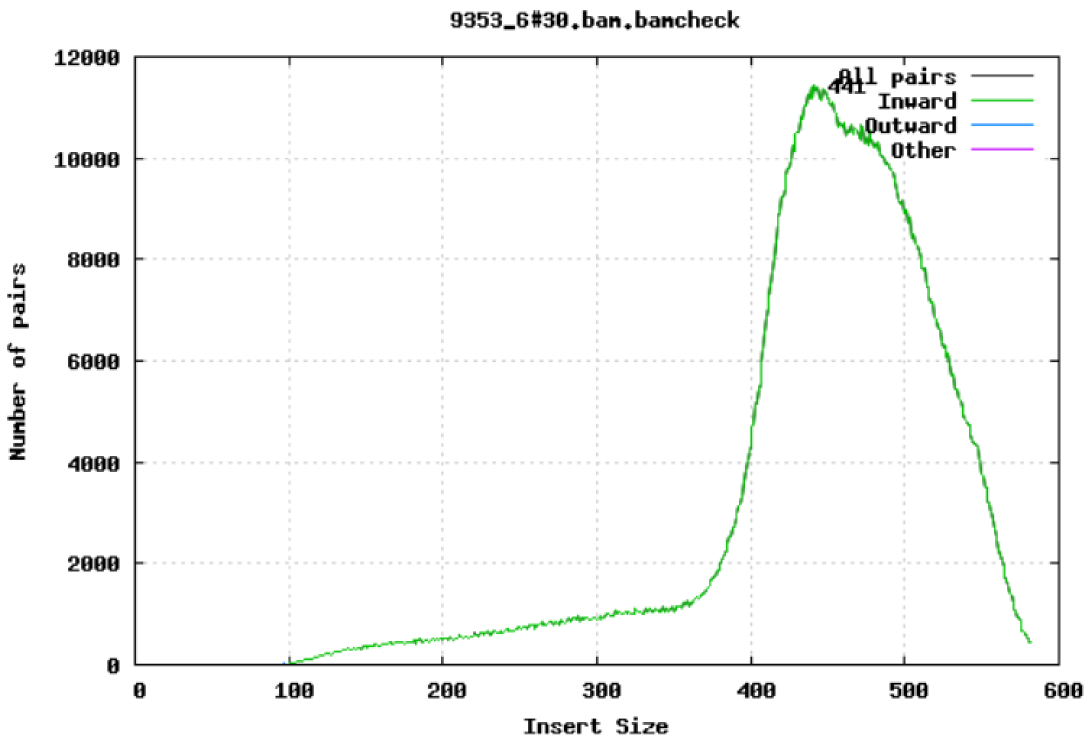


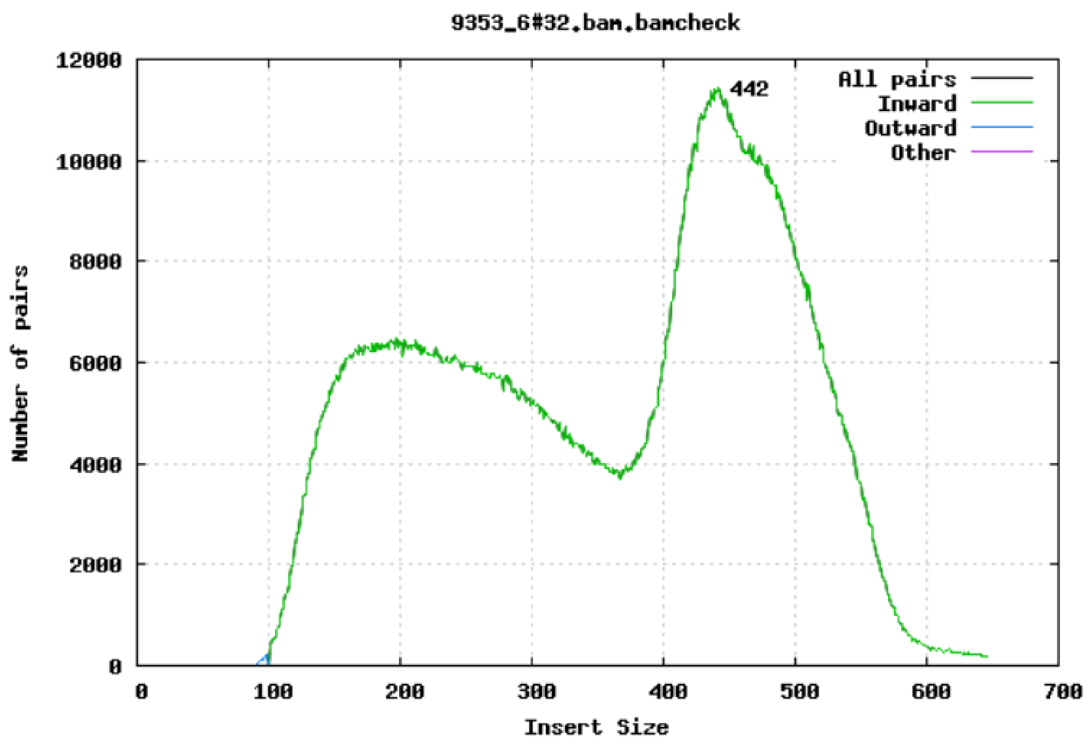


```
[ ]: ## GC Bias/Depth
```

3.6 Insert size

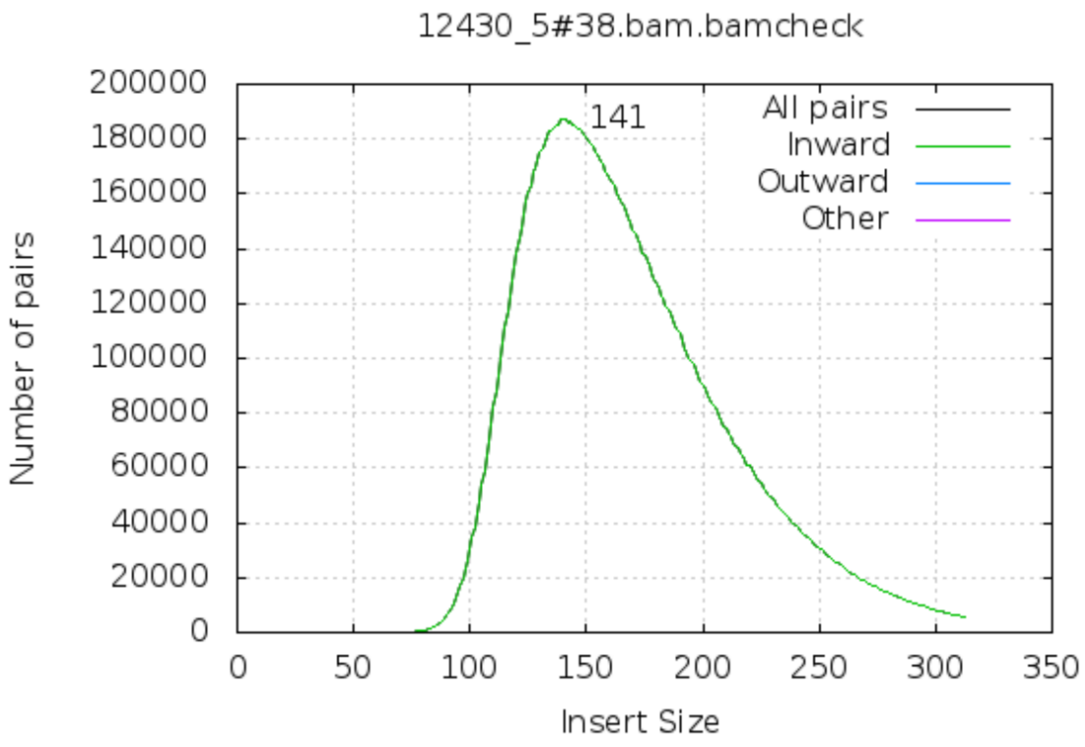
For paired-end sequencing the size of DNA fragments also matters. In the first of the examples below, the insert size peaks around 440 bp. In the second however, there is also a peak at around 200 bp. This indicates that there was an issue with the fragment size selection during library prep.





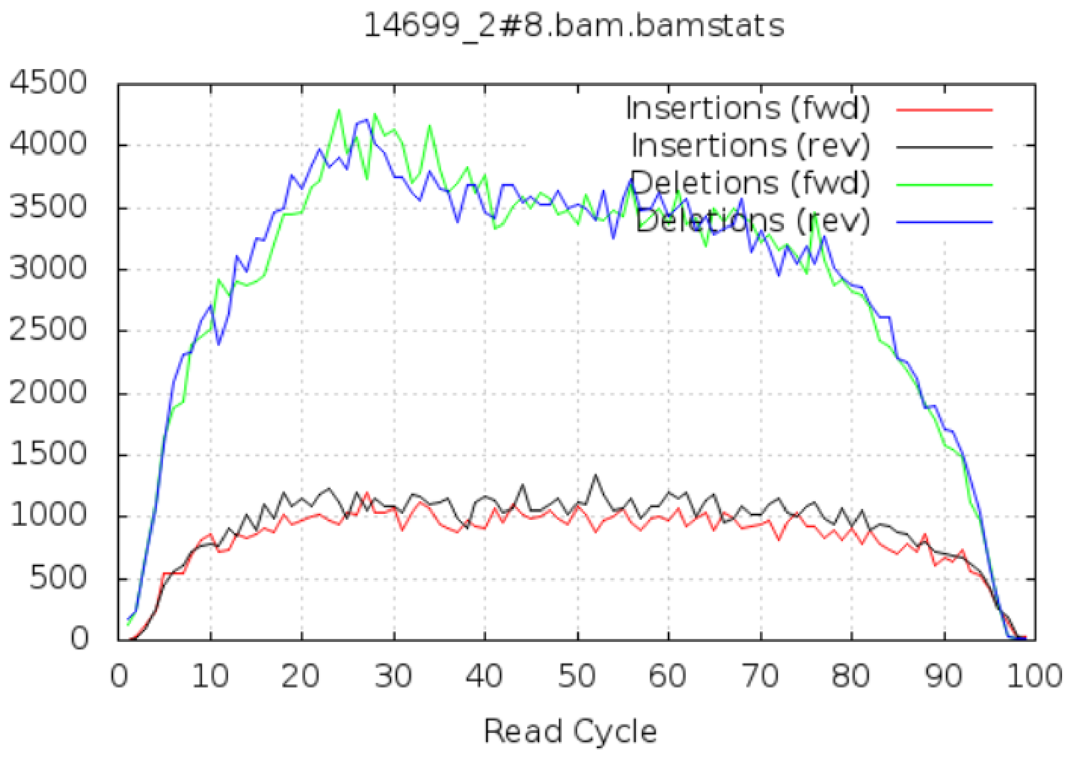
3.6.1 Exercises

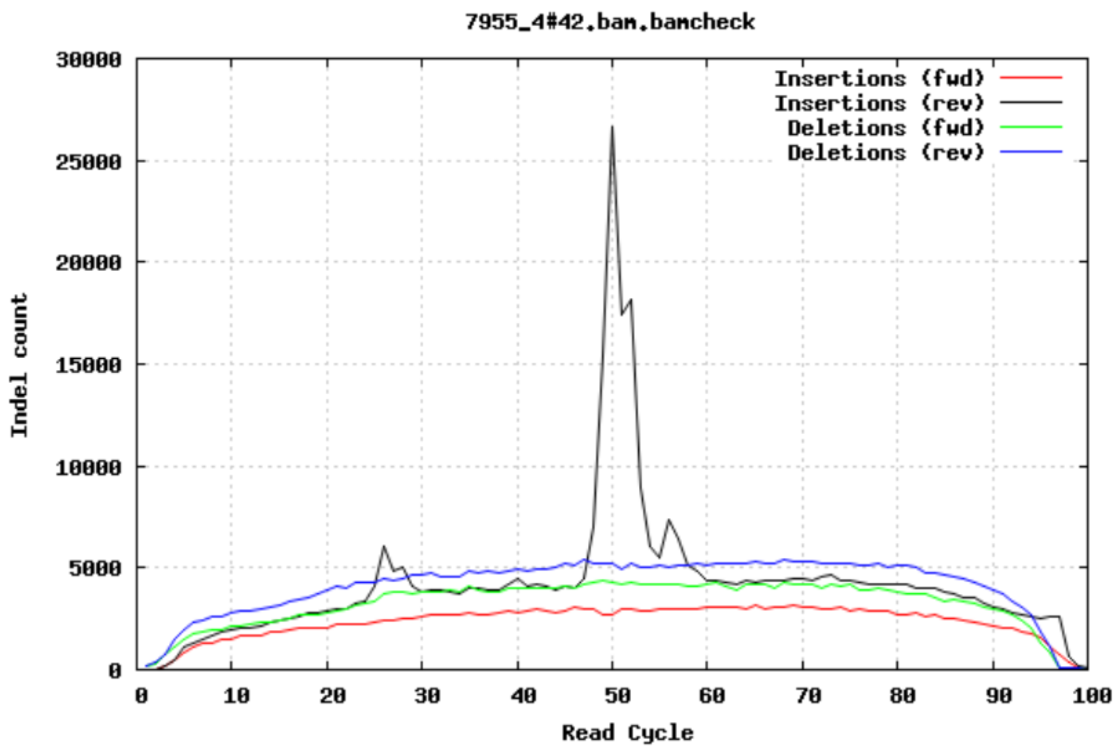
Q1: The figure below is from a 100bp paired-end sequencing. Can you spot any problems?



3.7 Insertions/Deletions per cycle

Sometimes, air bubbles occur in the flow cell, which can manifest as false indels. The spike in the second 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. We will use the bwa mapper to align the data to the Saccromyces cerevisiae genome, followed by samtools stats to generate the stats.

Read pairs are usually stored in two separate FASTQ files so that n-th read in the first file and the n-th read in the second file constitute a read pair. Can you devise a quick sanity check that reads in these two files indeed form pairs? The files must have the same number of lines and the naming of the reads usually suggests if they form a pair. The location of the files is:

```
lane1/s_7_1.fastq
lane1/s_7_2.fastq
```

[ ]:
```

```

Let's have a look at the script we are going to run to create the mappings:

[ ]:
```
cat create_mapping.sh
```

TDon't worry about he details of this for now as we will cover mapping data to a reference genome later in the course).

Now run the script to create the mappings and stats:

[ ]:
```
./create_mapping.sh
```

The script will produce the BAM file lane1.sorted.bam and a matching index file:

[ ]:
```
ls -alrt
```

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 lane1.sorted.bam \
    > lane1.sorted.bam.bchk
```

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

[ ]:
```
head -n 47 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 ^'#' 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.

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?

[ ]:

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 lane1-plots/ 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?

[ ]:

3.9 FastQC

An alternative method to asseing the QC of your sequnce data is to the the FastQC software application. This is a usefull tool to get an initial assement of the quality of your sequence data without having to align/map your data to a reference genome. It provide similar statistics and graphs as above and additional information like estimates of duplicated sequnces, adapter contamination, number of N's in the sequence runs. Let's take a look.

To run FastQC on XXX

[ ]:
```
fastqc *.fastq.gz -o fastqc_results
```

Let's look at the output files produced:

[ ]:
```
ls fastqc_results
```

As you can see FastQC runs on each fastq file seperately and generates QC stats and an html web page for each fastq file.

Open the html file for xxx.fastq.gz

[ ]:
```
firefox XXX &
```

Ask questions about the results:

Often you will hace sequence data for multiple samples or multiple runs and it will be useful to look at the quality reports for all data in one combined report. This combined report can be generated with a software application called multiqc. Let;s gather together all the fastqc report files from above into a single report.

[ ]:
```
multiqc fastqc_results
```

[ ]:
```
ls
```

This generates a singel report called multiqc_report.html. Let's look at this report:

[ ]:
```
firefox multiqc_report.html &
```

What is your assessmment of this sequencing run? Is it good quality data?

Now continue to the next section of the tutorial: Identifying contamination

4 Identifying contamination

It is always a good idea to check that your data is from the species you expect it to be.

4.1 Bactinspector

One useful software application for doing this is bactinspector. Bactinspector....

[ ]:
```
bactinspector check_species -fq 13681_1#18_1.fastq.gz
```

What is the predicted species?

Another useful tool for identifying contamination is Kraken.

4.2 Installing kraken

Up until now all the software you required has been available on the virtual machine as it has been installed via conda (a software package manager). Kraken is not available on the virtual machine so for this part of the practical we will attempt to use conda to install kraken:

[ ]:
```
conda create -n kraken kraken=x.y
```

Once installed sucessfully activate the software environment that contains kraken.

[ ]:
```
conda activate kraken
```

The reason we have to activate the environment is that often bioinformatics software has multiple software dependencies so having all software installed centrally in one loaction can cause conflicts. For example if trying to use a software application that relies on python 2 and another software application that relies on python 3 it is impossible to have them exist togther. Therefore we create software environments (or boxes) that conatin only the software and depencies needed and switch between them as needed. Now that we have activated the kraken environment let's use it to look for contamination.

4.3 Setting up a database

To run Kraken you need to either build a database or download an existing one. The standard database is very large (33 GB), but thankfully there are some smaller, pre-built databased available. To download the smallest of them, the 4 GB MiniKraken, run:

[ ]:
```
wget https://ccb.jhu.edu/software/kraken/dl/minikraken_20171019_4GB.tgz
```

This may take some time to run. So skip ahead to the next section on Heterozygous SNPs and come back to this section when the download is cmplete. (If you have trouble installing Kraken or downloading the database then we have pre-generated the kraken report file.)

Onve the database is downloaded all you need to do is un-tar it:

[ ]:
```
tar -zxvf minikraken_20171019_4GB.tgz
```

This version of the database is constructed from complete bacterial, archaeal, and viral genomes in RefSeq,

4.4 Running Kraken

To run Kraken, you need to provide the path to the database you just created. By default, the input files are assumed to be in FASTA format, so in this case we also need to tell Kraken that our input files are in FASTQ format, gzipped, and that they are paired end reads:

[ ]:
```
kraken --db ./minikraken_20171013_4GB --output kraken_results \
    --fastq-input --gzip-compressed --paired \
    data/13681_1#18_1.fastq.gz data/13681_1#18_2.fastq.gz
```

The five columns in the file that's generated are:

1. "C"/"U": one letter code indicating that the sequence was either classified or unclassified.
2. The sequence ID, obtained from the FASTA/FASTQ header.
3. The taxonomy ID Kraken used to label the sequence; this is 0 if the sequence is unclassified.
4. The length of the sequence in bp.
5. A space-delimited list indicating the LCA mapping of each k-mer in the sequence.

To get a better overview you can create a kraken report:

[ ]:
```
kraken-report --db ./minikraken_20171013_4GB kraken_results > kraken-report
```

4.5 Looking at the results

Let's have a closer look at the kraken_report for the sample. If for some reason your kraken-run failed there is a prebaked kraken-report at data/kraken-report

[ ]:
```
head -n 20 kraken-report
```

The six columns in this file are:

1. Percentage of reads covered by the clade rooted at this taxon
2. Number of reads covered by the clade rooted at this taxon
3. Number of reads assigned directly to this taxon
4. A rank code, indicating (U)nclassified, (D)omain, (K)ingdom, (P)hylum, (C)lass, (O)rder, (F)amily, (G)enus, or (S)pecies. All other ranks are simply '-'.
5. NCBI taxonomy ID
6. Scientific name

4.6 Exercises

Q1: What is the most prevalent species in this sample?

[ ]:
```

```

Q2: Does this match the bactinspector results?

```
[ ]:
```

Q3: What percentage of reads could not be classified?

```
[ ]:
```

4.7 Heterozygous SNPs

For bacteria, another thing that you can look at to detect contamination is if there are heterozygous SNPs in your samples. Simply put, if you align your reads to a reference, you would expect any SNPs to be homozygous, i.e. if one read differs from the reference genome, then the rest of the reads that map to that same location will also do so:

Homozygous SNP

```
Ref:   CTTGAGACGAAATCACTAAAAAACGTGACGACTTG
Read1: CTTGAGtCG
Read2: CTTGAGtCGAAA
Read3:    GAGtCGAAATCACTAAAA
Read4:       GtCGAAATCA
```

But if there is contamination, this may not be the case. In the example below, half of the mapped reads have the T allele and half have the A.

Heterozygous SNP

```
Ref:   CTTGAGACGAAATCACTAAAAAACGTGACGACTTG
Read1: CTTGAGtCG
Read2: CTTGAGaCGAAA
Read3:    GAGaCGAAATCACTAAAA
Read4:       GtCGAAATCA
```

One way to asses this is to map your reads to a reference genoe and acll variants and use bcftools to count the number of heterzygous SNPs.

4.8 ConFindr

An alternative to counting the number of heterozygous variants is to use a tool called ConFindr. ConFindr is a pipeline that can detect contamination in bacterial NGS data, both between and within species. Instead of looking at SNPs/variants across the whole genome, ConFindr works by looking at conserved core genes - either using rMLST genes (53 genes are known to be single copy and conserved across all bacteria with some known exceptions, which ConFindr handles), or custom sets of genes derived from core-genome schemes. As the genes ConFindr looks at are single copy, any sample that has multiple alleles of one or more gene is likely to be contaminated.

To read more information about ConFindr visit:

https://olc-bioinformatics.github.io/ConFindr/

Unfortunately we do not have time to run ConFindr here. But some of the automatated high throughput pipelines for mapping and snp calling and genome assembly include QC assment in the preprocessing steps like FastQC, bactinspector and confindr. We will be covering these pipelines later in this course.

```
[ ]: Congratulations! You have reached the end of this tutorial.
```