Mapping reads to a reference genome

Dr. Robert Kofler

October 17, 2014

RESOURCES

- the lecture: http://drrobertkofler.wikispaces.com/NGSandEELecture
- virtual machine image

SEQUENCING TECHNOLOGIES

Illumina HiSeq

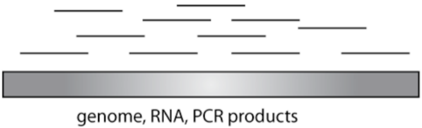
ABI Solid

Roche 454

PacBio RS

UNIFYING FEATURE: SHORT READS

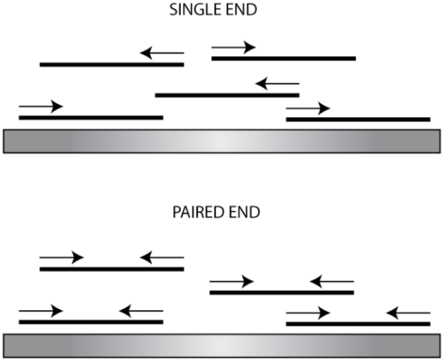
No matter the sequencing technology they all produce ”short reads”. These are stretches of DNA sequences randomly distributed in the feature of interest which is usually the genome or the transcriptome.



However the technologies vary with respect to

- read length (30 – 10000)
- number of reads (up to 100 million)
- error rate $< 1\% - 15\%$
- type of error (base substitutions, indels)

SINGLE END VS. PAIRED END READS



During "sample prep" (sample preparation) the genome (or the transcriptome) is randomly chopped into small pieces (e.g.: with a nebulizer).

If the resulting fragments (size 300-500bp) are sequenced only from one side we obtain single end reads. If they are sequenced from both ends we obtain paired end reads. In any case the reads are randomly distributed over the feature of interest.

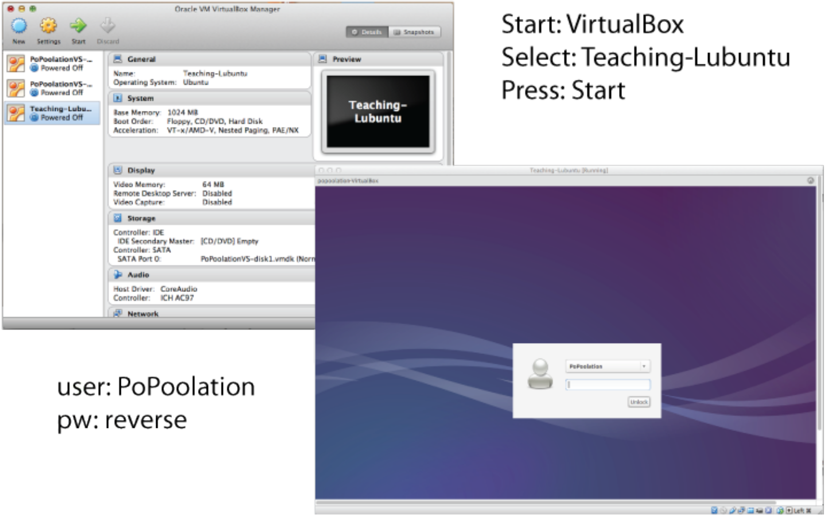
Section 2

fastq

Output from NGS technologies

So what do I get from such a NGS machine??
Not surprising the technologies have different output file formats. One common output is a ″fastq″ file. This name has historical reasons, it is derived from the first alignment software ″fasta″. The ″q″ stands for quality.

IN MEDIAS RES



General
Name: Teaching-Lubuntu
Operating System: Ubuntu

System
Base Memory: 1024 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Preview
Teaching-Lubuntu

Display
Video Memory: 64 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: PoPoolationVS-disk1.vmdk (Norm

Audio
Host Driver: CoreAudio
Controller: ICH AC97

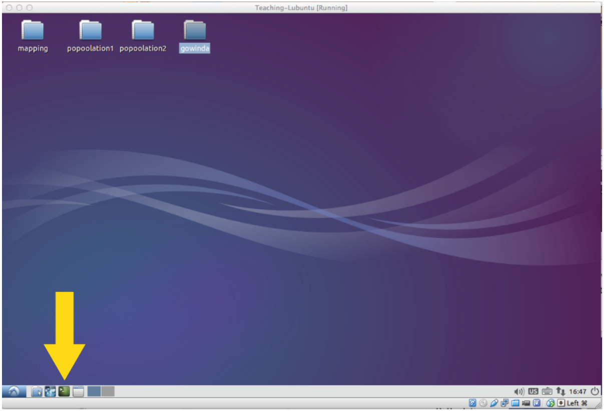
Network

Start: VirtualBox
Select: Teaching-Lubuntu
Press: Start

user: PoPoolation
pw: reverse

LUBUNTU - LIGHTWEIGHT

1. set the correct keyboard (german) ⇒ start ⇒ preferences ⇒ Keyboard input methods ⇒ Input method..
2. enter the command line



INSPECTING A FASTQ FILE

```
cd Desktop/mapping
less read_1.fastq
# voila that's a fastq file
```

```
@HWUSI-EAS300R:7:1:7:674#0/1
CTTTTGTAGTTTACAAATCATGAATAATTTATAGAGTTAGTAACTTATAATTAATATACCTAGGAAATAAGTTA
+HWUSI-EAS300R:7:1:7:674#0/1
abbb_[V`a_abaababa`abbaaba_bbbabbaab`aaaabaaba`baabbbabaaa`\bbaaa`aaa_b``a
@HWUSI-EAS300R:7:1:9:1897#0/1
ACTTAATTATTTATGCTTTTCTCTACTCTGCACGGCATGCAAATGCAATATAGATGCAAGGCGAGCCGAAACAA
+HWUSI-EAS300R:7:1:9:1897#0/1
aaab\`bb^bababbaababbbbbababbbbabaaaaabbbb[_bba_`^aabaabaaaa__aaaaaa[Saa_`
@HWUSI-EAS300R:7:1:13:849#0/1
GAAATATTGCGTAGCCGGAAACAAAAGAGTGCAAATACATTTCGACGATGATGAGAGAGCTATTCAGGGCTGTA
+HWUSI-EAS300R:7:1:13:849#0/1
a^``aaaabbb^`bab`a_Xaa_aa^_a__`_aaaa`ba`aaa_`_aaa`aZ_a^a]_^aR_a^_^`_`]^_`\Y
......
```

STRUCTURE OF A FASTQ-FILE

```
@HWUSI-EAS300R:7:1:7:674#0/1
CTTTTGTAGTTTACAAATCATGAATAATTTATAGAGTTAGTAACTTATAATTAATATACCTAGGAAATAAGTTA
+HWUSI-EAS300R:7:1:7:674#0/1
abbb_[V`a_abaababa`abbaaba_bbbabbaab`aaaabaaba`baabbbabaaa`\bbaaa`aaa_b``a
```

Every read occupies four lines in a fastq file

- @: name of the read (always starts with '@')
- CTT..: the sequence of the read
- +: the name again (only useful with multiline fastq entries)
- the base quality of the DNA sequence; one character for every nucleotide

Navigating a fastq-file

- ↑ ↓ scroll through the file
- ␣ next page
- ′b′ previous page
- ′g′ move to beginning of file
- ′G′ move to end of file
- /blabla search for blabla
- ′n′ move to the next search result
- ′N′ move to the previous search result

EXERCISE: NAVIGATING A FASTQ-FILE

- search for the DNA sequence ′AAAAAAAAAA′ (10x). Is it usually repeated more than 10x?
- also search for the next three hits of ′AAAAAAAAAA′
- search for the read ′HWUSI-EAS300R:7:1:14:1748′. How can it be found?
- what is the sequence of the last read in the file?

FASTQ QUALITY ENCODING

```
@HWUSI-EAS300R:7:1:7:674#0/1
CTTTTGTAGTTTACAAATCATGAATAATTTATAGAGTTAGTAACTTATAATTAATATACCTAGGAAATAAGTTA
+HWUSI-EAS300R:7:1:7:674#0/1
abbb_[V`a_abaababa`abbaaba_bbbabbaab`aaaabaaba`baabbbabaaa`\bbaaa`aaa_b``a
```

In a fastq file the first quality character refers to the first DNA character and so on..

```
CTTTTGTAGTTTACAAATCATGAATAATTTATAGAGTTAGTAACTTATAATTAATATACCTAGGAAATAAGTTA
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
abbb_[V`a_abaababa`abbaaba_bbbabbaab`aaaabaaba`baabbbabaaa`\bbaaa`aaa_b``a
```

So what does a quality of ’a’ or ’b’ mean? Usually there would be a simple answer but thanks to Illumina the answer is: ”it depends on the quality encoding”. Accordingly we now have different types of fastq files.

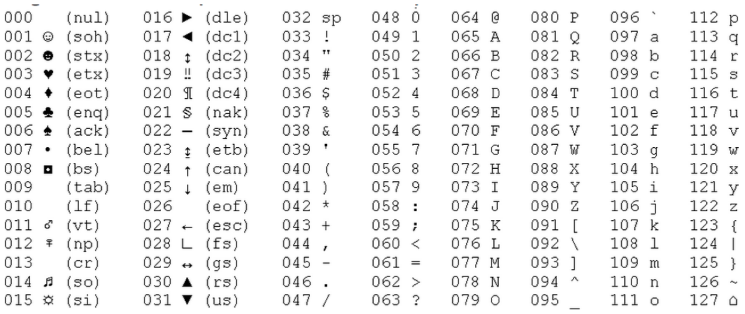
Quality demystified

Basically every base quality entry gives you the probability that the corresponding DNA nucleotide is wrong (thus a sequencing error). So a quality of 0.001 means that on average 1 in 1000 bases having this quality is wrong.
The encoding of the quality involves a few steps

- get the decimal value of the quality character from an ASCII table (see next page)
- subtract the offset (sanger=33 illumina=64)
- voila: the quality of the base; usually a number between 0-40 [=dec(ascii)]
- if you want, you can go on and calculate the probability of being a sequencing error:

$$p = 10^{-\frac{dec(ascii)}{10}}$$

ASCII TABLE



000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

In informatics every character has a number. For the computer they are actually interchangeable. The binary ’01000001’ refers at the same time to the number ’65’ and to the character ’A’. Only the context decides which definition will be used. It is therefore fairly straight forward to translate numbers between 0-127 to characters.

Example fastq decoding

```
@read1
TTACGTTTTT
+read1
87ba777777
```

Base 'A' in Illumina encoding (*offset* $= 64$)

- A has the quality b
- b $\Rightarrow$ 98 (from ascii table)
- $98 - 64 = 34$; my base quality for A is therefore 34
- error probability $p = 10^{-34/10} = 0.000398$

This error probability can be interpreted as 1 in 2511 ($= 1/0.000398$) base pairs being wrongly sequenced (= sequencing error)

EXERCISES FASTQ DECODING

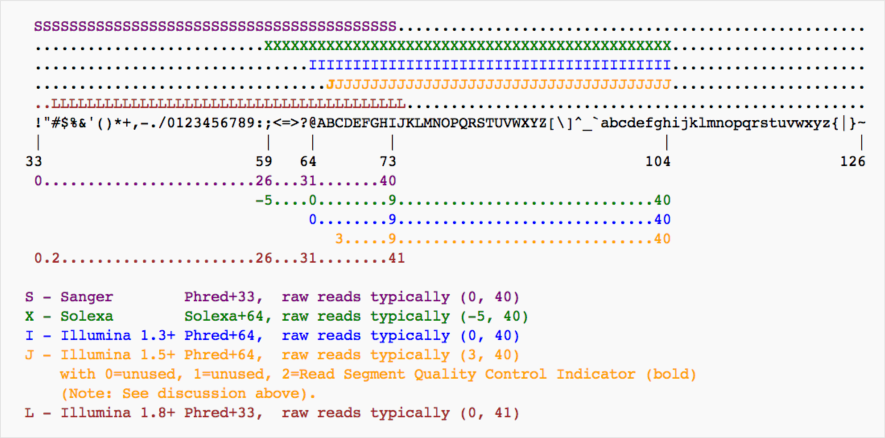
```
@read1
TTTACGTTTT
+read1
aaaa8aaaa
```

- Q1: Base quality of ’A’ in Illumina
- Q2: Error probability of ’A’ in Illumina
- Q3: Base quality of ’A’ in Sanger
- Q4: Error probability of ’A’ in Sanger
- Q5: Base quality of C in Sanger and Illumina. Is there anything weird about this results?
- Q6: Can you decide if the fastq file shown above is in Sanger or Illumina.

IS MY FASTQ-FILE SANGER OR ILLUMINA ENCODED?

If any of the bases has a negative quality with Illumina (64) than the encoding is Sanger (33). Therefore if you find any ascii character < 64 (e.g.: 1,2,3,4,5,6,7,8,9) in your fastq file than your encoding is Sanger; if not than it is Illumina.
Exercise: which quality encoding is your fastq file?

Overview fastq



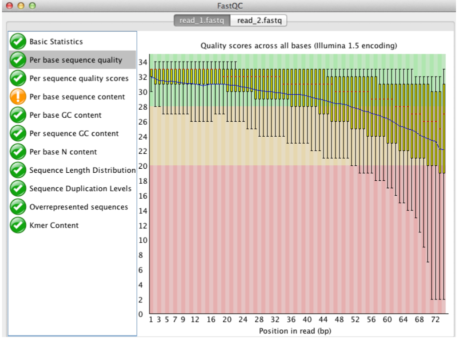
```
 SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....................................................
 ..........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......................
 ...............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII......................
 .................................JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ......................
 ..LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL....................................................
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
 |                         |    |        |                              |                     |
33                        59   64       73                            104                   126
 0........................26...31.......40
                          -5....0........9.............................40
                                0........9.............................40
                                   3.....9.............................40
 0.2......................26...31........41

S - Sanger        Phred+33,  raw reads typically (0, 40)
X - Solexa        Solexa+64, raw reads typically (-5, 40)
I - Illumina 1.3+ Phred+64,  raw reads typically (0, 40)
J - Illumina 1.5+ Phred+64,  raw reads typically (3, 40)
    with 0=unused, 1=unused, 2=Read Segment Quality Control Indicator (bold)
    (Note: See discussion above).
L - Illumina 1.8+ Phred+33,  raw reads typically (0, 41)
```

Note: in our institute we call the offset of 33 sanger and the offset of 64 illumina, which is not entirely in agreement with the definition shown here (source Wikipedia). So, in theory, there are actually five fastq files, in practice however only the offset is relevant (33, 64).

FastQC: Quality control of short reads

After obtaining the short read files, the first questions you are usually asking is: How successful has the sequencing of my reads been? What is the quality of my reads? The user-friendly tool FastQC can help to answer these questions.


FastQC
read_1.fastq
read_2.fastq
✓ Basic Statistics
✓ Per base sequence quality
✓ Per sequence quality scores
! Per base sequence content
✓ Per base GC content
✓ Per sequence GC content
✓ Per base N content
✓ Sequence Length Distribution
✓ Sequence Duplication Levels
✓ Overrepresented sequences
✓ Kmer Content
Quality scores across all bases (Illumina 1.5 encoding)
34 32 30 28 26 24 22 20 18 16 14 12 10 8 6 4 2 0
1 3 5 7 9 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
Position in read (bp)

EXERCISE FASTQC

Open FastQC (File manager ⇒ programs ⇒ FastQC ⇒ fastqc ⇒ execute), load the file 'read_1.fastq" and answer the following questions:

- number of reads in your file
- the fastq encoding of your file
- the read length
- the average quality of your reads
- fraction of duplicates, triplicates, etc

Section 3

Single-end semi-global alignment

GREAT SHORT READS..

OK, great now we have the short reads, what can we do with them? Any ideas?

Some examples of applications for short reads

- Where is the position of the read in the genome: mapping
- Can we piece them together into larger junks (like playing a puzzle): assembly
- Measure abundance of reads (count duplicates, triplicates etc)

Most biological interesting questions involve mapping of the reads. A few biological questions require assembly, and even less the counting of the reads (this is more for quality testing of the sequencing step). We will therefore only focus on mapping (alignment).

Before mapping

ALWAYS make sure your data are complete. Quick and dirty

```
wc read_*
>  209288  209288 11279782 read_1.fastq
>  209288  209288 11279782 read_2.fastq
>  418576  418576 22559564 total
```

more professional

```
md5sum read_*
>MD5 (read_1.fastq) = fd8fdfce336391e106fdc84ee60dd622
>MD5 (read_2.fastq) = 18d01db158ea29334d90b68880d9f6bb
```

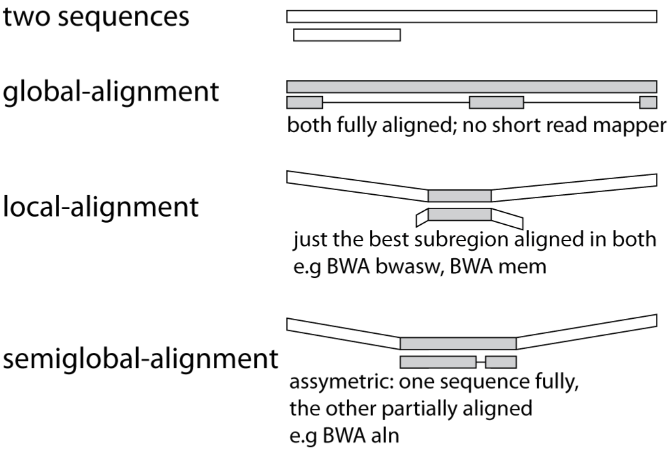
The wordcount or, much better, the md5-sum should be compared to the values provided by the sequencing facility (e.g.: BGI).

MAPPING SOFTWARE

- BWA ⇐
- Bowtie
- GEM
- BFAST
- STAMPY
- MAQ
- SOAP
- BLAT

In this introduction, we are using BWA for aligning short reads to the reference genome. BWA is currently one of the most widely used alignment algorithm. It is fast and flexible (many options)

TYPES OF ALIGNMENTS


two sequences
global-alignment
both fully aligned; no short read mapper
local-alignment
just the best subregion aligned in both
e.g BWA bwasw, BWA mem
semiglobal-alignment
assymetric: one sequence fully,
the other partially aligned
e.g BWA aln

⇒ we start with 'bwa aln' = semi-global

FIRST: THE REFERENCE GENOME (FASTA-FILE)

Now in addition to the fastq file we are encountering the fasta-file. To view the reference genome type:

```
less dmel-2R-chromosome-r5.22.fasta
```

and you obtain:

```
>2R type=chromosome_arm; loc=2R:1..21146708; ID=2R; dbxref=GB:AE013599; MD5=1589a9447d4dc94c048aa48ea5b8099d; length=21146708; release=r5.22; species=Dmel;
GACCCGCTAGGAGATGTTGAGATTGTGAGTACTTCTTGGAATTTGGTTAT
CTATTATAAAATGGATCCATATTTTAAAATGTTAACAAAGGGTAATGCGC
TTATACAAAGTATGAGGAAAGTTTGCGAAAGACTTCATAGCTTTGAAGAG
....
```

- symbol indicating start of a new sequence ’>’ directly followed, without space, by the name of the sequence; after space some description of the sequence
- the actual sequence, mostly in chunks of 60 nucleotides per line (sometimes 50, the length is irrelevant)
- optional: additional sequences with the same formatting

Preparing the reference sequence for mapping

```
mkdir wg
awk '{print $1}' dmel-2R-chromosome-r5.22.fasta
     > wg/dmel-2R-short.fa
bwa index wg/dmel-2R-short.fa
```

Note; the awk command just prints the first column, where a column is defined as everything being separated by a space. With this command we are therefore removing the description of the fasta entry. This step is strongly recommended as unnecessarily long fasta identifiers may lead to problems in downstream analysis.

```
>2R type=chromosome_arm; loc=2R:1..21146708; ID=2R; dbxref=GB:AE013599; MD5=1589
     a9447d4dc94c048aa48ea5b8099d; length=21146708; release=r5.22; species=Dmel;
GACCCGCTAGGAGATGTTGAGATTGTGAGTACTTCTTGGAATTTGGTTAT
...

=> transformed into:

>2R
GACCCGCTAGGAGATGTTGAGATTGTGAGTACTTCTTGGAATTTGGTTAT
...
```

SINGLE END MAPPING

```
mkdir se
bwa aln -I -m 100000 -o 1 -n 0.01 -l 200 -e 12 -d 12 -t 2
    wg/dmel-2R-short.fa read_1.fastq > se/read_1.sai
bwa samse wg/dmel-2R-short.fa se/read_1.sai read_1.fastq>
    se/read_1.sam
```

- -I input is in Illumina encoding (offset 64); do not provide this when input is in sanger! Very important parameter!
- -m not important; just telling bwa to process smaller amounts of reads at once
- -l 200 seed size (needs to be longer than the read length to disable seeding)
- -e 12 -d 12 gap length (for insertions and deletions)
- -o 1 maximum number of gaps
- -n 0.01 the number of allowed mismatches in terms of probability of missing the read. In general the lower the value the more mismatches are allowed. The exact translation is shown at the beginning of the mapping
- -t 2 number of threads, the more the faster

CONVERTING SAM TO BAM

- sam.. Sequence Alignment Map format ⇒ optimized for humans
- bam.. binary sam ⇒ optimized for computers

It is easily possible to convert a sam to bam and vice versa a bam to sam. In the following we convert a sam into a bam and finally sort the bam file

```
samtools view -Sb se/read_1.sam | samtools sort - se/read_1
```

- -S input is sam
- -b output is bam (-S may be merged with -b to -Sb)
- 'sort - outpufile' input for sorting is the pipe (rather than a file)

⇒ ignore the whining of samtools..

VISUALIZE THE RESULTS

First we need to index the bam file (necessary for visualization but nothing else)

```
samtools index se/read_1.bam
mkdir igv
# open IGV, a genome browser written in java
java -Xmx1g -jar ~/programs/IGV_2.3.26/igv.jar
# -Xmx1g = assign 1 GB of RAM to Java
# -jar *.jar = start igv.jar (jar = java archive)
```

- Genomes ⇒ Create .genome File; Unique identifier=”Dmel2R”; Desriptive name=”Drosophila chromosome 2R”; FASTA file=’wg/dmel-2R-short.fa’;
- Save genome as ’igv/Dmel2R.genome’
- Load the mapped reads (singleend/read_1.bam)
- Zoom in at position 8,250,000
- Inspect the alignments

PAY SPECIAL ATTENTION TO

- the coverage graph on top
- SNPs (colored bases in the alignment); zoom in and out
- reference sequence (zoom in and out)
- alignment strands of the reads (find some forward and some reverse aligned reads);
- right click; color alignments by read strand
- find some insertions and deletions
- hover over read, what is happening now?
- find reads of low mapping quality (faint and white-ish)

INSPECTING THE SAM-FILE

```
less se/read_1.sam
```

and you will get something like

```
@SQ     SN:2R   LN:21146708
HWUSI-EAS300R:7:1:7:674#0 16 2R 7923527 37 74M * 0 0 TAACT... BAAC@B... XT:A:U NM:i:0
HWUSI-EAS300R:7:1:9:1897#0 0 2R 8172056 37 74M * 0 0 ACTTA... BBBC=A... XT:A:U NM:i:0
HWUSI-EAS300R:7:1:13:849#0 0 2R 8294036 37 74M * 0 0 GAAAT... B?ABBB... XT:A:U NM:i:1
```

- **samtools**: `http://samtools.sourceforge.net/`
- **documentation**: `http://samtools.sourceforge.net/SAMv1.pdf`

SAM FILE

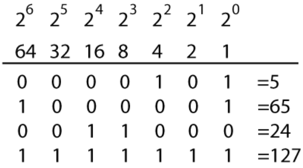
A sam file has two parts, first a general info (lines starting with an '@') and second the aligned reads (all other lines) which have the following columns

- col 1: name of the read
- col 2: binary flag (contains lots of useful information, compressed into one number...)
- col 3: ID of the reference chromosome (e.g.: a read may align to 2L or 3R ...)
- col 4: position of the read in the reference chromosome (most 5' position is given)
- col 5: mapping quality (do not mix up with base quality)
- col 6: CIGAR string; specifies the alignment of the read with the reference chromosome
- col 10: sequence of the read (like in fastq)
- col 11: base quality of read (like in fastq); Per definition, the quality encoding has to be Sanger! (There should be only one version of sam file as opposed to the 5 fastq files)
- col 12+: optional information; dependent on the mapping software

SAM FILE; COLUMN 2 - BINARY FLAG

They developed a human readable format (sam) and a computer readable format (binary sam), yet they included a binary flag into the human readable format...seriously...wtf..

However, as the sam file is a currently quite standard we still have to learn the meaning of this flag. What you actually find in column 2 is an integer. In informatics every integer can be displayed as binary number:



2^6	2^5	2^4	2^3	2^2	2^1	2^0	
64	32	16	8	4	2	1	
0	0	0	0	1	0	1	=5
1	0	0	0	0	0	1	=65
0	0	1	1	0	0	0	=24
1	1	1	1	1	1	1	=127

In the binary flag of the sam-file, every of these 0 or 1 has a meaning. For example a 1 at the third position (**from the right**) means that the read could not be mapped to the reference genome (e.g.: to many mismatches) whereas a 0 at the third position means the read could be mapped to the reference genome

THE MOST IMPORTANT FLAGS

To make matters worse, the actual meaning of every flag is provided in hexadecimal numbers. Overview of the most important flags. For more details see `http://samtools.sourceforge.net/SAMv1.pdf`.

- 0x1 = first position from the right; if set to 1, than the read is a part of a paired end read (0 means single end)
- 0x2 = second position: if set to 1, than the reads map as a proper pair (definition depends on mapping software)
- 0x4 = third position: the read has not been mapped (remember 0 means mapped)
- 0x8 = fourth position: the mate (the other read of the pair) could not be mapped
- 0x10 = fifth position: the read maps as reverse complement (the sequence in col10 has been reverse complemented to)
- 0x20 = sixth position: the mate maps as reverse complement
- 0x100 = ninth position: this is a secondary alignment (thus there is an alternative more suitable alignment for the given read)
- 0x200 = tenth position: the read does not pass quality control (Illuminas requirements)
- 0x400 = eleventh position: the read is a duplicate (either PCR or optical); a suitable software has to be used to identify duplicates (e.g.: Picard)

QUICK WAY TO VISUALIZE BINARY FLAGS?

```
echo '83'  |wcalc -b
echo '0x20'|wcalc -b
```

A binary flag (e.g.: 0x20) is set if 'col 2' (e.g.: 83) contains a bit (1) at the given position. In the following example the flag is not set

```
0b1010011 (83)
 0b100000 (0x20)
```

In the following example the flag is set

```
0b1010011 (83)
     0b10 (0x2)
```

Exercise binary flag

- interpret the binary flag ’4’
- interpret the binary flag ’0’
- you had a Unix introduction so what is the command at the bottom doing?
- type the command and interpret the results

```
cat se/read_1.sam | awk '{print $2}' | sort |
   uniq -c
```

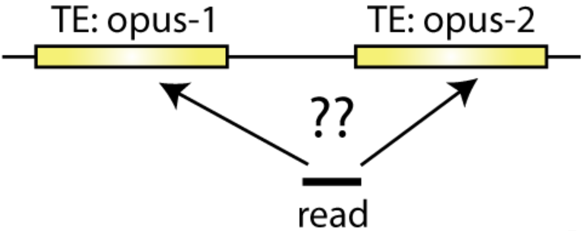
Quick way to check the meaning of a binary flag: `http://broadinstitute.github.io/picard/explain-flags.html`

AMBIGUOUS MAPPING POSITION AND MAPPING QUALITY

Remember column 5 of the sam file contains the mapping quality. Similarly to the base quality, the mapping quality is the log scaled probability that the position of the read is wrong.

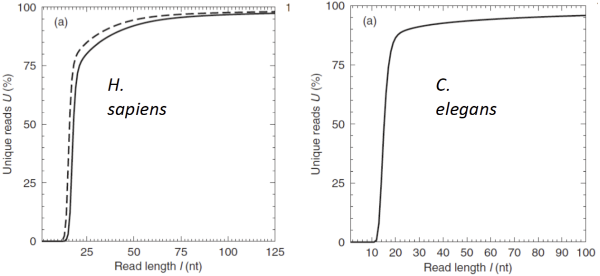
- 20.. one out of 100 reads is wrongly mapped
- 30.. one out of 1000 reads is wrongly mapped
- 0.. every read is wrongly mapped

A major cause for incorrect or ambiguous mapping positions are repetitive regions in the genome



FRACTION UNIQUE IN THE GENOME

Of course, the fraction of unique positions in the genome depends on the read length. Increasing the read length will lead to more unique alignment positions.



- mismatches are not considered in this study (thus when allowing for sequencing errors, longer reads will be required)
- in principle the same limitations also apply to paired end reads
- examples of repetitive regions: transposable elements, microsatellites, satellite DNA, copy number variation, large gene families

Source: Whiteford N. (2005) An analysis of feasibility of short read sequencing

Remove reads with a low mapping quality

To get rid of these unreliably (ambiguously) aligned reads we can remove reads with a low mapping quality from the bam file. In the following we are removing all reads with a mapping quality lower than 20.

```
samtools view -q 20 -b se/read_1.bam > se/read_1.q20
    .bam
samtools index se/read_1.q20.bam
```

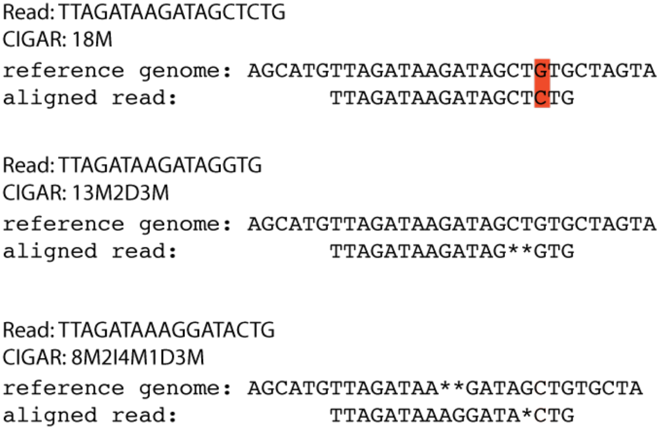
Exercise:

- load ’se/read_1.q20.bam into IGV
- compare to ’se/read_1.bam
- search for regions in the genome that differ between these two files

e.g.: 2R:8,251,975-8,254,865

CIGAR STRING (COL 6)

The CIGAR string ('col 6') only specifies the alignment between the read and the reference genome. Remember the starting position is given in 'col 4' and the sequence of the read in 'col 10'. Note mismatches are not affecting the CIGAR string



Read: TTAGATAAGATAGCTCTG
CIGAR: 18M

```
reference genome: AGCATGTTAGATAAGATAGCTGTGCTAGTA
aligned read:           TTAGATAAGATAGCTCTG
```

Read: TTAGATAAGATAGGTG
CIGAR: 13M2D3M

```
reference genome: AGCATGTTAGATAAGATAGCTGTGCTAGTA
aligned read:           TTAGATAAGATAG**GTG
```

Read: TTAGATAAAGGATACTG
CIGAR: 8M2I4M1D3M

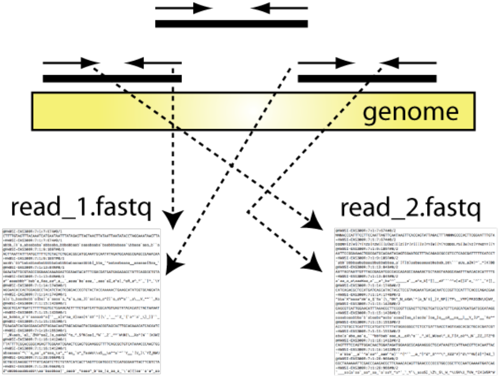
```
reference genome: AGCATGTTAGATAA**GATAGCTGTGCTA
aligned read:           TTAGATAAAGGATA*CTG
```

Section 4

Paired-end mapping

Paired end reads

With the Illumina technology you will get two fastq-files for paired end reads. Reads are always provided in the same order in both fastq-files. The two reads of one pair can therefore be recognized by having the same index in the both fastq-files.



Note: the first read (read_1.fastq) is not necessarily the 5′ read. Assignment as the first read is a stochastic process (therefore usually 50% 5′ reads and 50% 3′ reads).

PAIRED-END MAPPING; SEMI-GLOBAL

```
mkdir pe
bwa aln -I -m 100000 -o 1 -n 0.01 -l 200 -e 12 -d 12 -t 2 wg/dmel
    -2R-short.fa read_1.fastq > pe/read_1.sai
bwa aln -I -m 100000 -o 1 -n 0.01 -l 200 -e 12 -d 12 -t 2 wg/dmel
    -2R-short.fa read_2.fastq > pe/read_2.sai
bwa sampe wg/dmel-2R-short.fa pe/read_1.sai pe/read_2.sai read_1.
    fastq read_2.fastq > pe/pe.semiglob.sam
# inspect the file
less pe/pe.semiglob.sam
```

Note: the reads are actually mapped as single ends. Only the 'bwa sampe' step creates the paired end information.

PE FLAGS

Following some flags of the sam file that are mostly relevant for PE reads

- 0x1 = first position from the right; if set to 1, than the read is a part of a paired end read (0 means single end)
- 0x2 = second position: if set to 1, than the reads map as a proper pair (definition depends on mapping software)
- 0x4 = third position: the read has not been mapped (remember 0 means mapped)
- 0x8 = fourth position: the mate (the other read of the pair) could not be mapped
- 0x10 = fifth position: the read maps as reverse complement (the sequence in col10 has been reverse complemented to)
- 0x20 = sixth position: the mate maps as reverse complement

Exercise

- open 'pe/pe.semiglob.sam'
- What is the meaning of the flag of the first read (83)?
- What is the meaning of the flag of the second read (163)?

PAIRED-END MAPPING; LOCAL

BWA also implements a local alignment algorithm, 'mem'

```
mkdir mem
bwa mem wg/dmel-2R-short.fa read_1.fastq read_2.fastq > mem/pe.local.sam
# Note, 'mem' does not have the -I flag and we thus obtain wrong results!
# For 'mem' the fastq file has to be sanger encoded.
# Generally i recommend to only work with fastq files in Sanger!
# => Convert illumina encoded fastq files immediately
```

Prepare for visualization

```
# first the semi-global alignment
samtools view -Sb pe/pe.semiglob.sam | samtools sort - pe/pe.semiglob.sort
samtools index pe/pe.semiglob.sort.bam

# than the local alignment
samtools view -Sb mem/pe.local.sam | samtools sort - mem/pe.local.sort
samtools index mem/pe.local.sort.bam

# start IGV
java -Xmx1g -jar ~/programs/IGV_2.3.26/igv.jar
```

VISUALIZING END RESULTS

- Load the single end reads (se/read_1.sort.bam)
- Load the PE semi-globaly aligned reads (pe/pe.semiglob.sort.bam)
- Load the PE locally aligned reads (mem/pe.local.sort.bam)
- Zoom in at position 8,250,000
- Right click on PE alignments ⇒ View as pairs
- Inspect the alignments
- Find some orphan reads (reads without mate). What could be the reason for orphan reads?
- Find some locally aligned PE fragments (search for SNPs specific to one sample); What do you find? Could this result in a bias?

2R:8,250,685

DEEPENING THE UNDERSTANDING OF SAM-FILES

- Open the sam file (less mem/pe.local.sam)
- Pick a read and locate the read in IGV (start position)
- Find a read with an insertion (CIGAR: I) and locate the read in IGV
- Find a read with a deletion (CIGAR: D) and locate the read in IGV
- Find a soft-clipped read (CIGAR: S) and locate the read in IGV
- Find a read mapped to the reverse strand and locate the read in IGV
- Find a read mapped to the forward strand and locate the read in IGV