Filtering with SAMTools

Reservations

Use our summer school *reservation* (**CoreNGS-Fri**) when submitting batch jobs to get higher priority on the **ls6** normal queue ***today***:

```
sbatch --reservation=CoreNGS-Fri <batch_file>.slurm
idev -m 180 -N 1 -A OTH21164 -r CoreNGS-Fri
```


Overview

As we have seen, the **SAMTools** suite allows you to manipulate the **SAM/BAM** files produced by most aligners. There are many sub-commands in this suite, but the most common and useful are:

1. Convert text-format **SAM** files into binary **BAM** files (**samtools view**) and vice versa
2. Sort **BAM** files by reference coordinates (**samtools sort**)
3. Index **BAM** files that have been sorted (**samtools index**)
4. ***Filter*** alignment records based on BAM ***flags***, mapping quality or location (**samtools view**)

Since **BAM** files are binary, they can't be viewed directly using standard Unix file viewers such as **more**, **less** and **head.** We have seen how **samtools view** can be used to binary-format **BAM** files into text format for viewing. But **samtools view** also has options that let you do powerful filtering of the output. We focus on this filtering capability in this set of exercises.

The most common **samtools view** filtering options are:

- **-q N** – only report alignment records with ***mapping quality*** of at least **N** (**>= N**).
- **-f 0xXX** – only report alignment records where the specified ***flags*** are all ***set*** (are all **1**)
  - you can provide the flags in decimal, or as here as hexadecimal
- **-F 0xXX** – only report alignment records where the specified ***flags*** are all ***cleared*** (are all **0**)

Setup

Login to **ls6.tacc.utexas.edu** and start an **idev** session.

Start an idev session

```
idev -m 180 -N 1 -A OTH21164 -r CoreNGS-Fri
# or
idev -m 90 -N 1 -A OTH21164 -p development
```

Next set up a directory for these exercises, and copy an indexed **BAM** file there. This is the **yeast_pe.sort.bam** file from our The Basic Alignment Workflow exercises.

Setup for samtools exercises

```
mkdir -p $SCRATCH/core_ngs/samtools
cd $SCRATCH/core_ngs/samtools
cp $CORENGS/catchup/for_samtools/* .
```

References

Handy links

- The SAM format specification
  - especially section 1.4 - alignment section fields
- Manual for SAMTools
  - especially the 1st section on **samtools view**.

SAM header fields

The 11 **SAM** alignment record required fields (**Tab**-delimited).

Col	Field	Type	Regexp/Range	Brief description	
1	QNAME	String	`[!-?A-~]{1,255}`	Query template NAME	*read name from fastq*
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG*s*	
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME	*contig + start*
4	POS	Int	$[0,2^{29}-1]$	1-based leftmost mapping POSition	*= locus*
5	MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality	
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string	*use this to find end coordinate*
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next segment	
8	PNEXT	Int	$[0,2^{29}-1]$	Position of the mate/next segment	
9	TLEN	Int	$[-2^{29}+1,2^{29}-1]$	observed Template LENgth	*insert size, if paired*
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence	
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33	

SAM flags field

Meaning of each bit (***flag***) in the **SAM** alignment records ***flags*** field (column 2). The most commonly flags are denoted in ***red***.

Bit Decimal	Bit Hex	Description		
1	0x1	template having multiple segments in sequencing		*1 = part of a read pair*
2	0x2	each segment properly aligned according to the aligner		*1 = "properly" paired*
4	0x4	segment unmapped	*read did map = 0*	*1 = read did not map*
8	0x8	next segment in the template unmapped		*1 = mate did not map*
16	0x10	SEQ being reverse complemented	*plus strand read = 0*	*1 = minus strand read*
32	0x20	SEQ of the next segment in the template being reverse complemented		*1 = mate on minus strand*
64	0x40	the first segment in the template		*1 = R1 read*
128	0x80	the last segment in the template		*1 = R2 read*
256	0x100	secondary alignment		*1 = secondary alignment*
512	0x200	not passing filters, such as platform/vendor quality controls		
1024	0x400	PCR or optical duplicate		*1 = marked as duplicate*
2048	0x800	supplementary alignment		*1 = maps to ALT contig*

SAM CIGAR string

Format of the CIGAR string in column 6 of **SAM** alignment records.

```
Ref   CTGGCCATTATCTC--GGTGGTAGGACATGGCATGCCC
Read          aaATGTCGCGGTG.TAGGAggatcc
```

2S|5M|2I|4M|1D|5M|6S

	Op	BAM	Description
	M	0	alignment match (can be a sequence match or mismatch)
	I	1	insertion to the reference
	D	2	deletion from the reference
	N	3	skipped region from the reference
	S	4	soft clipping (clipped sequences present in SEQ)
*	H	5	hard clipping (clipped sequences NOT present in SEQ)
*	P	6	padding (silent deletion from padded reference)
*	=	7	sequence match
*	X	8	sequence mismatch

"N" indicates splicing event in RNAseq BAMs

*Rarer / newer

CIGAR = "Concise Idiosyncratic Gapped Alignment Report"

Exercises

Analyzing the CIGAR string for indels

Suppose we want to know how many alignments included insertions or deletions (***indels***) versus the reference. Looking at the CIGAR field definition table above, we see that insertions are denoted with the character **I** and deletions with the character **D**. We'll use this information, along with **samtools view**, **cut** and **grep**, to count the number of indels.

```
mkdir -p $SCRATCH/core_ngs/samtools
cd $SCRATCH/core_ngs/samtools
cp $CORENGS/catchup/for_samtools/* .
```

We'll do this first exercise one step at a time to get comfortable with piping commands. Start by just looking at the first few alignment records of the **BAM** file:

```
module load biocontainers  # takes a while
module load samtools

cd $SCRATCH/core_ngs/samtools
samtools view yeast_pe.sort.bam | head
```

With all the line wrapping, it looks pretty ugly. Still, you can pick out the CIGAR string in column 6. Let's select just that column with **cut**:

```
samtools view yeast_pe.sort.bam | cut -f 6 | head -20
```

Next, make sure we're only looking at alignment records that represent mapped reads. The **-F 0x4** option says to filter records where the **0x4** flag (read ***un mapped***) is **0**, resulting it only ***mapped*** reads being output.

```
samtools view -F 0x4 yeast_pe.sort.bam | cut -f 6 | head -20
```

Now we use **grep** with the pattern **'[ID]'** to select lines (which are now just CIGAR strings) that have **I**nsert or **D**eletion operators. The brackets ([ ]) denote a character class pattern, which matches any of the characters inside the brackets. Be sure to ask for **Perl** regular expressions (**-P**) so that this ***character class*** syntax is recognized.

```
samtools view -F 0x4 yeast_pe.sort.bam | cut -f 6 | grep -P '[ID]' | head
```

Ok, this looks good. We're ready to run this against all the alignment records, and count the results:

Count reads that mappedi with indels

```
samtools view -F 0x4 yeast_pe.sort.bam | cut -f 6 | grep -P '[ID]' | wc -l
```

There are 6697 such records.

What is that in terms of the rate of indels? For that we need to count the total number of mapped reads. Here we can just use the **-c** (**c**ount only) option to **samtools view**.

Count all mapped reads

```
samtools view -c -F 0x4 yeast_pe.sort.bam
```

There should be 547664 mapped alignments.

Knowing these two numbers we can just divide them, using **awk** (remember, **bash** only does integer arithmetic). Because we're not piping anything in to **aw** , any body we specify won't be executed. So we do the math in the **BEGIN** section:

```
awk 'BEGIN{ print 100*6697/547664,"%" }'
```

The result should be 1.22283 %.

In addition to the rate, converted to a percentage, we also output the literal percent sign (**%**). The double quotes (**"**) denote a literal string in **awk**, and the comma between the number and the string says to separate the two fields with whatever the default **O**utput **F**ield **S**eparator (**OFS**) is. By default, both **awk**'s Input **F**ield **S**eparator (**FS**) is **whitespace** (any number of **space** and **Tab** characters) and its **O**utput **F**ield **S**eparator is a **single space**.

So what if we want to get fancy and do all this in a one-liner command? We can use **echo** and ***backtick evaluation*** to put both numbers in a string, then pipe that string to **awk**. This time we use **awk** body code, and refer to the two **whitespace**-separated fields being passed in: total count (**$1**) and indel count (**$2**).

One-liner for calculating BAM indel rate

```
echo "`samtools view -F 0x4 -c yeast_pe.sort.bam` \
    $( samtools view -F 0x4 yeast_pe.sort.bam | cut -f 6 | grep -P '[ID]' | wc -l )" \
    | awk '{ print 100 * $2/$1,"%" }'
```

awk also has a **printf** function, which can take the standard formatting commands (see https://en.wikipedia.org/wiki/Printf_format_string#Type_field).

```
awk 'BEGIN{ printf("%.2f %%\n", 100*6697/547664) }'
```

Notes:

- The **printf** arguments are enclosed in parentheses since it is a true function.
- The 1st argument is the ***format string***, enclosed in double quotes.
  - The **%.2f** ***format specifier*** says to output a floating point number with 2 digits after the decimal place.
  - The **%%** format specifier is used to output a single, literal percent sign.
  - Unlike the standard **print** statement, the **printf** function does not automatically append a **newline** to the output, so **\n** is added to the format string here.
- Remaining arguments to **printf** are the values to be substituted for the format specifiers (here our percentage computation).

Filtering by location range

Sometimes you just want to examine a subset of reads in detail. Once you have a ***sorted and indexed*** BAM, you can use the coordinate filtering options of **samtools view** to do this. Here are some examples:

When you're interested in ***mapped reads*** (which is true most of the time) be sure to specify the **-F 0x4** option, which says to filter records where the **0x4** flag (read ***unmapped***) is **0**, resulting it only ***mapped*** reads being output.

```
mkdir -p $SCRATCH/core_ngs/samtools
cd $SCRATCH/core_ngs/samtools
cp $CORENGS/catchup/for_samtools/* .
module load biocontainers
module load samtools
```

```
cd $SCRATCH/core_ngs/samtools

# count the number of reads mapped to chromosome 2 (chrII)
samtools view -c -F 0x4 yeast_pe.sort.bam chrII

# count the number of reads mapped to chromosomes 1 or M (chrI, chrM)
samtools view -c -F 0x4 yeast_pe.sort.bam chrI chrM

# count the number of reads mapped to chromosomes 1 that overlap coordinates 1000-2000
samtools view -c -F 0x4 yeast_pe.sort.bam chrI:1000-2000

# since there are only 20 reads in the chrI:1000-2000 region, examine them individually
samtools view -F 0x4 yeast_pe.sort.bam chrI:1000-2000

# look at a subset of field for chrI:1000-2000 reads
#   2=flags, 3=contig, 4=start, 5=mapping quality, 6=CIGAR, 9=insert size
samtools view -F 0x4 yeast_pe.sort.bam chrI:1000-2000 | cut -f 2-6,9
```

samtools view -L <bed_file>

You can also provide a **BED**-format file with one record for each desired overlap region: **samtools view -L <bed_file>**. This is a quick way to perform one of the functions of **bedtools intersect**.

Since you will find yourself wanting to interpret the flags field, and it's easier to do that when it is represented as hexadecimal, let's use **awk** to help do that.

```
# look at a subset of field for chrI:1000-2000 reads
#   2=flags, 3=contig, 4=start, 5=mapping quality, 6=CIGAR, 9=insert size
samtools view -F 0x4 yeast_pe.sort.bam chrI:1000-2000 | cut -f 2-6,9 | \
  awk 'BEGIN{FS=OFS="\t"}
       {$1 = sprintf("0x%x", $1); print}'
```

Notes:

- If a command line is continued on more than one line, the line continuation character (\ ) is used.
- There is a **sprintf** (**s**tring **print f**ormatted) function in **awk**, just as there is in many modern programming languages.
- We use **sprintf** to re-format the 1st input field (here the **cut** ***flags***) in hex
  - using the ***format directive*** (**%x**)
  - also adding a literal "**0x**" prefix to denote the numeric ***base*** is he**x**adecimal
- The results of **sprintf** are stored back into the 1st field (**$1**), replacing the original value.
- **awk**'s print statement is then used with no other arguments to print ***all*** its input fields, including the changed **$1**.

Exercise: How many of the chrI:1000-2000 alignments are from *properly paired* mapped reads?

Properly paired reads have the **0x2** flag set (**1**). ***All*** our reads also have the **0x1** flag set because they are paired-end reads. Mapped reads will have the **0x4** flag cleared (**0**), and properly paired mapped reads will have the **0x4** flag cleared (**0**) as well (***mate not unmapped*** = mate mapped). So ***all mapped proper pairs*** will end in **0x3** = **0b0011** (binary).

Also, using **'$'** in a **grep** pattern anchors the pattern to the ***end of the line*** (only patterns matching before a linefeed are printed).
Here's one way to do it, building on our last command line:

```
samtools view -F 0x4 yeast_pe.sort.bam chrI:1000-2000 | awk '
  BEGIN{FS=OFS="\t"}
  {$2 = sprintf("0x%x", $2); print $2}' | grep -c '3$'
```

Note that here we don't need the line continuation character because the newline after the first single quote is part of the script command string.

Use the **0x2** flag (**1** = ***properly paired***)
Here's another way of doing it:

```
samtools view -c -F 0x4 -f 0x2 yeast_pe.sort.bam chrI:1000-2000
```

About mapping quality

Mapping qualities are a measure of how likely a given sequence alignment to its reported location is correct. If a read's mapping quality is low (especially if it is zero, or ***mapQ 0*** for short) the ***read maps to multiple locations on the genome*** (they are ***multi-hit*** or ***multi-mapping*** reads), and we can't be sure whether the reported location is the correct one.

Aligners also differ in whether they report alternate alignments for ***multi-hit*** reads. Some things to keep in mind:

- Alternate locations for a mapped read will be flagged as ***secondary*** (flag **0x100**)
- While they often provide valuable information, ***secondary*** reads must be filtered for some downstream applications
    - e.g., ChIP-seq peak calling and variant analysis with **GATK**.
- When ***secondary*** reads are reported, the total number of alignment records in the **BAM** file is ***greater*** than the number of reads in the input **FAST(** files!
    - this affects how the true mapping rate must be calculated
    - true mapping rate = (***pirmary*** mapped reads) / (total **BAM** file sequences - ***secondary*** mapped reads)

Here are some examples of how different aligners handle reporting of multi-hit reads and their mapping qualities:

- **bwa aln** (global alignment) and **bowtie2** with ***default parameters*** (both **--local** default end-to-end mode) report ***at most one location*** for a read that maps
    - this will be the location with the ***best*** mapping quality and alignment
    - if a given read maps ***equally well*** to multiple locations, these aligners ***pick one location at random***
        - **bwa aln** will always report a **0** mapping quality for these multi-hit reads
        - **bowtie2** will report a low mapping quality (**< 10**), based on the complexity (information content) of the sequence
- **bwa mem** (local alignment) can ***always report more than one location*** for a mapped read
    - its definition of a ***secondary*** alignment is different (and a bit non-standard)
        - if ***one part of a read*** maps to one location and ***another part*** maps somewhere else (e.g. because of RNA splicing), the longer alignment is marked as ***primary*** and the shorter as ***secondary***.
    - there is no way to disable reporting of ***secondary*** alignments with **bwa mem**.
        - but they can be filtered from the sorted BAM with **-F 0x100** (***secondary*** alignment flag = **0**).
- **bowtie2** can be configured to ***report more than one location*** for a mapped read
    - the **-k N** option says to report up to **N** alignments for a read
- most ***transcriptome-aware*** RNA-seq aligners by default ***report more than one location*** for a mapped read
    - e.g. **hisat2, star, tophat2**.
    - when reads are quantified (counted with respect to genes), multiply-mapped reads can be counted fractionally
        - e.g. if a read maps to 5 genes, it can be counted as 1/5 for each of the genes

Filtering for high-quality reads

Using our **yeast_pe.sort.bam** file, let's do some some quality filtering.

```
mkdir -p $SCRATCH/core_ngs/samtools
cd $SCRATCH/core_ngs/samtools
cp $CORENGS/catchup/for_samtools/* .
module load biocontainers
module load samtools
```

Exercise: Use samtools view with -F, -f and -q options to create a BAM containing only mapped, properly paired, high-quality (mapQ 20+) reads. Call the output file yeast_pe.sort.filt.bam.

Note that we use the **-b** flag to tell **samtools view** to output **BAM** format (not the **SAM** format text we've been looking at).

solution

```
samtools view -b -F 0x4 -f 0x2 -q 20 yeast_pe.sort.bam > yeast_pe.sort.filt.bam
# or
samtools view -b -F 0x4 -f 0x2 -q 20 -o yeast_pe.sort.filt.bam  yeast_pe.sort.bam
```

Exercise: How many records are in the filtered BAM compared to the original? How many read *pairs* does this represent?

samtools view -c

solution

```
samtools view -c yeast_pe.sort.bam
samtools view -c yeast_pe.sort.filt.bam

# or to be really fancy...
echo "`samtools view -c yeast_pe.sort.bam` \
      `samtools view -c yeast_pe.sort.filt.bam`" | \
       awk '{printf "%d original reads\n", $1;
             printf "%d Q20 reads (%d read pairs)\n", $2, $2/2;
             printf "%0.2f%% high-quality reads\n", 100*$2/$1}'
```

There were 1,184,360 alignment records in the original BAM, and only 456,890 in the quality-filtered BAM, around 38% of our starting reads.

Since we have only properly paired reads, the filtered BAM will contain equal numbers of both **R1s** and **R2s**. So the number of read pairs is 456890/2 or 228451.

Exercise: How many *primary* aligned reads (0x100 = 0) are in the bwa_local.sort.dup.bam file?

Combining SAM flags

samtools view only pays attention to ***one*** **-F** or **-f** option, so to specify more than one flag value you need to combine them into one number. For example, combining **0x100** and **0x4** yields **0x104**.

You want both the ***secondary*** (**0x100**) and ***unmapped*** **(0x4)** flags to be **0**.

solution

```
samtools view -F 0x104 -c bwa_local.sort.dup.bam
```

There are 874791 ***primary*** alignments.

solution

```
samtools view -F 0x4 -f 0x100 -c bwa_local.sort.dup.bam
```

And 133209 ***secondary*** alignments.