More Alignment exercises

Reservations

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

```
sbatch --reservation=CoreNGS-Thu <batch_file>.slurm
```


Exercise #3: PE alignment with BioITeam scripts

Now that you've done everything the hard way, let's see how to do run an alignment pipeline using a **BWA** alignment script maintained by the BioITeam, **/work/projects/BioITeam/common/script/align_bwa_illumina.sh**. Type in the script name to see its usage.

```
align_bwa_illumina.sh 2022_05_05
Align Illumina SE or PE data with bwa. Produces a sorted, indexed,
duplicate-marked BAM file and various statistics files. Usage:

align_bwa_illumina.sh <aln_mode> <in_file> <out_pfx> <assembly> [ paired trim_sz trim_sz2 seq_fmt qual_fmt ]

Required arguments:
  aln_mode  Alignment mode, either global (bwa aln) or local (bwa mem).
  in_file   For single-end alignments, path to input sequence file.
            For paired-end alignments using fastq, path to the the R1
            fastq file which must contain the string 'R1' in its name.
            The corresponding 'R2' must have the same path except for 'R1'.
  out_pfx   Desired prefix of output files in the current directory.
  assembly  One of hg38, hg19, hg38, mm10, mm9, sacCer3, sacCer1, ce11, ce10,
            danRer7, hs_mirbase, mm_mirbase, or reference index prefix.
Optional arguments:
  paired    0 = single end alignment (default); 1 = paired end.
  trim_sz   Size to trim reads to. Default 0 (no trimming)
  trim_sz2  Size to trim R2 reads to for paired end alignments.
            Defaults to trim_sz
  seq_fmt   Format of sequence file (fastq, bam or scarf). Default is
            fastq if the input file has a '.fastq' extension; scarf
            if it has a '.sequence.txt' extension.
  qual_type Type of read quality scores (sanger, illumina or solexa).
            Default is sanger for fastq, illumina for scarf.
Environment variables:
  show_only  1 = only show what would be done (default not set)
  aln_args   other bowtie2 options (e.g. '-T 20' for mem, '-l 20' for aln)
  no_markdup 1 = don't mark duplicates (default 0, mark duplicates)
  run_fastqc 1 = run fastqc (default 0, don't run). Note that output
             will be in the directory containing the fastq files.
  keep       1 = keep unsorted BAM (default 0, don't keep)
  bwa_bin    BWA binary to use. Default bwa 0.7.x. Note that bwa 0.6.2
             or earlier should be used for scarf and other short reads.
  also: NUM_THREADS, BAM_SORT_MEM, SORT_THREADS, JAVA_MEM_ARG

Examples:
  align_bwa_illumina.sh local  ABC_L001_R1.fastq.gz my_abc hg38 1
  align_bwa_illumina.sh global ABC_L001_R1.fastq.gz my_abc hg38 1 50
  align_bwa_illumina.sh global sequence.txt old sacCer3 0 '' '' scarf solexa
```

There are lots of bells and whistles in the arguments, but the most important are the first few:

1. **aln_mode** – whether to perform a **global** or **local** alignment (the 1st argument must be one of those words)
    - **global** mode uses the **bwa aln** workflow as we did above
    - **local** mode uses the **bwa mem** command
2. **in_file** – full or relative path to the **FASTQ** file (just the R1 fastq if paired end). Can be compressed (**.gz**)
3. **out_pfx** – ***prefix*** for all the output files produced by the script. Should relate back to what the data is.
4. **assembly** – genome assembly to use.
    - there are pre-built indexes for some common eukaryotes (**hg38**, **hg19**, **mm10**, **mm9**, **danRer7**, **sacCer3**) that you can use
    - or provide a full path for a **bwa** reference index you have built somewhere
5. **paired** flag – **0** means ***single end*** (the default); **1** means ***paired end***
6. **trim_sz** – if you want the **FASTQ** hard trimmed down to a specific length before alignment, supply that number here

We're going to run this script and a similar **Bowtie2** alignment script, on the yeast data using the TACC batch system. In a new directory, copy over the commands and submit the batch job. We ask for 2 hours (**-t 02:00:00**) with 4 tasks/node (**-w 4**); since we have 4 commands, this will run on 1 compute node.

Run multiple alignments using the TACC batch system

```
# Make sure you're not in an idev session by looking at the hostname
hostname
# If the hostname looks like "c455-004.ls6.tacc.utexas.edu", exit the idev session

# Copy over the Yeast data if needed
mkdir -p $SCRATCH/core_ngs/alignment/fastq
cp $CORENGS/alignment/Sample_Yeast*.gz $SCRATCH/core_ngs/alignment/fastq/

# Make a new alignment directory for running these scripts
mkdir -p $SCRATCH/core_ngs/alignment/bwa_script
cd $SCRATCH/core_ngs/alignment/bwa_script
ln -s -f ../fastq

# Copy the alignment commands file and submit the batch job
cd $SCRATCH/core_ngs/alignment/bwa_script
cp $CORENGS/tacc/aln_script.cmds .

launcher_creator.py -j aln_script.cmds -n aln_script -t 01:00:00 -w 4 -a OTH21164 -q normal
sbatch --reservation=CoreNGS-Thu aln_script.slurm

# or
launcher_creator.py -j aln_script.cmds -n aln_script -t 01:00:00 -w 4 -a OTH21164 -q development
sbatch aln_script.slurm

showq -u
```

While we're waiting for the job to complete, lets look at the **aln_script.cmds** file.

Commands to run multiple alignment scripts

```
/work/projects/BioITeam/common/script/align_bwa_illumina.sh     global ./fastq/Sample_Yeast_L005_R1.cat.fastq.
gz bwa_global sacCer3 1 50
/work/projects/BioITeam/common/script/align_bwa_illumina.sh     local  ./fastq/Sample_Yeast_L005_R1.cat.fastq.
gz bwa_local  sacCer3 1
/work/projects/BioITeam/common/script/align_bowtie2_illumina.sh global ./fastq/Sample_Yeast_L005_R1.cat.fastq.
gz bt2_global sacCer3 1 50
/work/projects/BioITeam/common/script/align_bowtie2_illumina.sh local  ./fastq/Sample_Yeast_L005_R1.cat.fastq.
gz bt2_local  sacCer3 1
```

Notes:

- The **1st** command performs a paired-end **BWA *global*** alignment (similar to above), but asks that the 100 bp reads be trimmed to **50** first.
  - we refer to the pre-built index for yeast by name: **sacCer3**
    - this index is located in the **/work/projects/BioITeam/ref_genome/bwa/bwtsw/sacCer3/** directory
  - we provide the name of the **R1 FASTQ** file
    - because we request a PE alignment (the **1** argument) the script will look for a similarly-named **R2** file.
  - all output files associated with this command will be named with the prefix **bwa_global**.
- The **2nd** command performs a paired-end **BWA *local*** alignment.
  - all output files associated with this command will be named with the prefix **bwa_local**.
  - no trimming is requested because the ***local*** alignment should ignore 5' and 3' bases that don't match the reference genome
- The **3rd** command performs a paired-end **Bowtie2 *global*** alignment.
  - the **Bowtie2** alignment script has the same first arguments as the **BWA** alignment script.
  - all output files associated with this command will be named with the prefix **bt2_global**.
  - again, we specify that reads should first be trimmed to 50 bp.
- The **4th** command performs a paired-end **Bowtie2 *local*** alignment.
  - all output files associated with this command will be named with the prefix **bt2_local**.
  - again, no trimming is requested for the ***local*** alignment.

Output files

This alignment pipeline script performs the following steps:

- Hard trims **FASTQ**, if optionally specified (**fastx_trimmer**)
- Performs the **global** or **local** alignment (here, a PE alignment)
  - **BWA global**: **bwa aln** the R1 and R2 separately, then **bwa sampe** to produce a **SAM** file
  - **BWA local**: call **bwa mem** with both R1 and R2 to produce a **SAM** file

    - **Bowtie2 global**: call **bowtie2** in its default global (end-to-end) mode on both R1 and R2 to produce a **SAM** file
    - **Bowtie2 local**: call **bowtie2 --local** with both R1 and R2 to produce a **SAM** file
- Converts **SAM** to **BAM** (**samtools view**)
- Sorts the **BAM** (**samtools sort**)
- Marks duplicates (**Picard MarkDuplicates**)
- Indexes the sorted, duplicate-marked **BAM** (**samtools index**)
- Gathers statistics (**samtools idxstats**, **samtools flagstat**, plus a custom statistics script of Anna's)
- Removes intermediate files

There are a number of output files, with the most important being those desribed below.

1. **<prefix>.align.log** – Log file of the entire alignment process.
    - check the **tail** of this file to make sure the alignment was successful
2. **<prefix>.sort.dup.bam** – Sorted, duplicate-marked alignment file.
3. **<prefix>.sort.dup.bam.bai** – Index for the sorted, duplicate-marked alignment file
4. **<prefix>.flagstat.txt** – **samtools flagstat** output
5. **<prefix>.idxstats.txt** – **samtools idxstats** output
6. **<prefix>.samstats.txt** – Summary alignment statistics from Anna's stats script
7. **<prefix>.iszinfo.txt** – Insert size statistics (for paired-end alignments) from Anna's stats script

Verifying alignment success

The alignment log will have a "I ran successfully" message at the end if all went well, and if there was an error, the important information should also be at the end of the log file. So you can use **tail** to check the status of an alignment. For example:

Checking the alignment log file

```
tail bwa_global.align.log
```

This will show something like:

```
------------------------------------------------------------------
..Done alignmentUtils.pl bamstats - 2022-06-10 12:59:05
.. samstats file 'bwa_global.samstats.txt' exists and is not empty - 2022-06-10 12:59:05
================================================================================
## Cleaning up files (keep 0) - 2022-06-10 12:59:05
================================================================================
ckRes 0 cleanup
================================================================================
## All bwa alignment tasks completed successfully! - 2022-06-10 12:59:06
================================================================================
```

Notice that *success message*: "***All bwa alignment tasks completed successfully!***". It should only appear ***once*** in any successful alignment log.

When multiple alignment commands are run in parallel it is important to check them all, and you can use **grep** looking for part of the unique success message to do this. For example:

Count the number of successful alignments

```
grep 'completed successfully!' *align.log | wc -l
```

If this command returns 4 (the number of alignment tasks we performed), all went well, and we're done.

But what if something went wrong? How can we tell which alignment task was not successful? You could **tail** the log files one by one to see which one(s) don't have the message, but you can also use a special **grep** option to do this work.

Check for failed alignment tasks

```
grep -L 'completed successfully' *.align.log
```

The **-L** option tells **grep** to only print the ***filenames*** that ***don't contain*** the pattern. Perfect! To see happens in the case of failure, try it on a file that doesn't contain that message:

```
grep -L 'completed successfully' aln_script.cmds
```

Checking alignment statistics

The **<prefix>.samstats.txt** statistics files produced by the alignment pipeline has a lot of good information in one place. If you look at **bwa_global.samstats.txt** you'll see something like this:

<prefix>.samstats.txt output

```
-----------------------------------------------
             Aligner:       bwa
     Total sequences:   1184360
        Total mapped:    539079 (45.5 %)
      Total unmapped:    645281 (54.5 %)
             Primary:    539079 (100.0 %)
           Secondary:
          Duplicates:    249655 (46.3 %)
          Fwd strand:    267978 (49.7 %)
          Rev strand:    271101 (50.3 %)
          Unique hit:    503629 (93.4 %)
           Multi hit:     35450 (6.6 %)
           Soft clip:
           All match:    531746 (98.6 %)
              Indels:      7333 (1.4 %)
            Spliced:
-----------------------------------------------
       Total PE seqs:   1184360
      PE seqs mapped:    539079 (45.5 %)
        Num PE pairs:    592180
   F5 1st end mapped:    372121 (62.8 %)
   F3 2nd end mapped:    166958 (28.2 %)
     PE pairs mapped:     80975 (13.7 %)
     PE proper pairs:     16817 (2.8 %)
-----------------------------------------------
```

Since this was a paired end alignment there is paired-end specific information reported.

You can also view statistics on insert sizes for properly paired reads in the **bwa_global.iszinfo.txt** file. This tells you the average (mean) insert size, standard deviation, mode (most common value), and fivenum values (minimum, 1st quartile, median, 3rd quartile, maximum).

<prefix>.iszinfo.txt output

```
  Insert size stats for: bwa_global
        Number of pairs: 16807 (proper)
 Number of insert sizes: 406
        Mean [-/+ 1 SD]: 296 [176 416]  (sd 120)
         Mode [Fivenum]: 228  [51 224 232 241 500]
```

A quick way to check alignment stats if you have run multiple alignments is again to use **grep**. For example:

Review multiple alignment rates

```
grep 'Total mapped' *samstats.txt
```

will produce output like this:

```
bt2_global.samstats.txt:        Total mapped:    602893 (50.9 %)
bt2_local.samstats.txt:        Total mapped:    788069 (66.5 %)
bwa_global.samstats.txt:        Total mapped:    539079 (45.5 %)
bwa_local.samstats.txt:        Total mapped:   1008000 (76.5 %
```

Exercise: How would you list the median insert size for all the alignments?

That information is in the ***.iszinfo.txt** files, on the line labeled **Mode**.

The median value is th 3rd value in the 5 fivnum values; it is the 7th whitespace-separated field on the **Mode** line.

Use **grep** to isolate the **Mode** line, and **awk** to isolate the median value field:

```
grep 'Mode' *.iszinfo.txt | awk '{print $1,"Median insert size:",$7}'
```

TACC batch system considerations

The great thing about pipeline scripts like this is that you can perform alignments on many datasets in parallel at TACC, and they are written to take advantage of having multiple cores on TACC nodes where possible.

On the **ls6** the pipeline scripts are designed to run best with no more than 4 tasks per node. Although each **ls6** node has 128 physical cores per node, the alignment workflow is heavily I/O bound overall, and we don't want to overload the file system.

> **Always specify wayness 4 for alignment pipeline scripts**
>
> These alignment scripts should always be run with a ***wayness*** of **4** (**-w 4**) in the **ls6** batch system, meaning at most 4 commands per node.

Exercise #4: Bowtie2 alignment - Vibrio cholerae RNA-seq

While we have focused on aligning eukaryotic data, the same tools can be used with prokaryotic data. The major differences are less about the underlying data and much more about the external/public databases that store and distribute reference data. If we want to study a prokaryote, the reference data is usually downloaded from a resource like GenBank.

In this exercise, we will use some RNA-seq data from *Vibrio cholerae*, published on GEO here, and align it to a reference genome.

Overview of Vibrio cholerae alignment workflow with Bowtie2

Alignment of this prokaryotic data follows the workflow below. Here we will concentrate on steps 1 and 2.

1. Prepare the **vibCho** reference index for **bowtie2** from **GenBank** records
2. Align reads using **bowtie2**, producing a **SAM** file
3. Convert the **SAM** file to a **BAM** file (**samtools view**)
4. Sort the **BAM** file by genomic location (**samtools sort**)
5. Index the **BAM** file (**samtools index**)
6. Gather simple alignment statistics (**samtools flagstat** and **samtools idxstats**)

Obtaining the GenBank records

First prepare a directory for the **vibCho fasta**, and change to it:

```
mkdir -p $SCRATCH/core_ngs/references/fasta
cd $SCRATCH/core_ngs/references/fasta
```

V. cholerae has two chromosomes. We download each separately.

1. Navigate to http://www.ncbi.nlm.nih.gov/nuccore/NC_012582
   - click on the **Send to** down arrow (top right of page)
     - select **Complete Record**
     - select **File** as **Destination**, and **Format FASTA**
     - click **Create File**
   - in the **Opening File** dialog, select **Save File** then **OK**
     - Save the file on your local computer as **NC_012582.fa**
2. Back on the main http://www.ncbi.nlm.nih.gov/nuccore/NC_012582 page
   - click on the **Send to** down arrow (top right of page)
     - select **Complete Record**
     - select **File** as **Destination**, and **Format GFF3**
     - click **Create File**
   - in the **Opening File** dialog, select **Save File** then **OK**
     - Save the file on your local computer as **NC_012582.gff3**
3. Repeat steps 1 and 2 for the 2nd chromosome
   - NCBI URL is http://www.ncbi.nlm.nih.gov/nuccore/NC_012583
   - use **NC_012583** as the filename prefix for the files you save
   - you should now have 4 files:
     - **NC_012582.fa**, **NC_012582.gff3**
     - **NC_012583.fa**, **NC_012583.gff3**
4. Transfer the files from your local computer to TACC
   - to the **~/scratch/core_ngs/references/vibCho** directory created above
     - On a Mac or Windows 10 or later, use **scp** from your laptop
     - Otherwise on Windows, use the **pscp.exe** **PuTTy** tool
     - See Copying files between TACC and your laptop

```
mkdir -p $SCRATCH/core_ngs/references/fasta
cd $SCRATCH/core_ngs/references/fasta
cp $CORENGS/references/fasta/NC_* .
```

Once you have the 4 files locally in your **$SCRATCH/core_ngs/references/vibCho** directory, combine them using **cat**:

```
cd $SCRATCH/core_ngs/references/fasta
cat NC_01258[23].fa   > vibCho.O395.fa
cat NC_01258[23].gff3 > vibCho.O395.gff3

# verify there are 2 contigs in vibCho.O395.fa
grep -P '^>' vibCho.O395.fa
```

Now we have a reference sequence file that we can use with the **bowtie2** reference builder, and ultimately align sequence data against.

Introducing bowtie2

First make sure you're in an **idev** session:

Start an idev session

```
idev -m 120 -A OTH21164 -N 1 -r CoreNGS-Thu
# or
idev -m 90 -A OTH21164 -N 1 -p development
```

Go ahead and load the **bowtie2** module so we can examine some help pages and options.

```
module load biocontainers
module load bowtie2
```

Now that it's loaded, check out the options. There are ***a lot*** of them! In fact for the full range of options and their meaning, Google "Bowtie2 manual" and bring up that page (http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml). The ***Table of Contents*** is several pages long! Ouch!

This is the key to using **bowtie2** - it allows you to control almost everything about its behavior, which make it the go-to aligner for specialized alignment tasks (e.g. aligning miRNA or other small reads). But it also makes it is much more challenging to use than **bwa** – and it's easier to screw things up too!

Building the bowtie2 vibCho index

Before the alignment, of course, we've got to build a **bowtie2**- specific index using **bowtie2-build**. Go ahead and check out its options. Unlike for the aligner itself, we only need to worry about a few things here:

- **reference_in** file is just the **vibCho.O395.fa** FASTA we built from GenBank records
- **bt2_index_base** is the ***prefix*** of all the **bowtie2-build** output file

Here, to build the reference index for alignment, we only need the FASTA file. (This is not always true - extensively spliced transcriptomes requires splice junction annotations to align RNA-seq data properly.)

First create a directory specifically for the **bowtie2** index, then build the index using **bowtie-build**.

```
mkdir -p $SCRATCH/core_ngs/references/fasta
cd $SCRATCH/core_ngs/references/fasta
cp $CORENGS/references/fasta/vibCho* .
```

Prepare Bowtie2 index files for vibCho

```
mkdir -p $SCRATCH/core_ngs/references/bt2/vibCho
cd $SCRATCH/core_ngs/references/bt2/vibCho

# Symlink to the fasta file you created using relative path syntax
ln -sf ../../fasta/vibCho.O395.fa

bowtie2-build vibCho.O395.fa vibCho.O395
```

This should also go pretty fast. You can see the resulting files using **ls** like before.

Performing the bowtie2 alignment

Make sure you're in an **idev** session with the **bowtie2** **BioContainers** module loaded:

```
idev -m 120 -A OTH21164  -N 1 -r CoreNGS-Thu
# or
idev -m 90 -A OTH21164 -N 1 -p development

module load biocontainers
module load bowtie2
```

We'll set up a new directory to perform the *V. cholerae* data alignment. But first make sure you have the **FASTQ** file to align and the vibCho **bowtie2** index.

```
# Get a pre-built vibCho index if you didn't already build one
mkdir -p $SCRATCH/core_ngs/references/bt2/vibCho
cp $CORENGS/references/bt2/vibCho/*.*  $SCRATCH/core_ngs/references/bt2/vibCho/

# Get the FASTQ to align
mkdir -p $SCRATCH/core_ngs/alignment/fastq
cp $CORENGS/alignment/*fastq.gz  $SCRATCH/core_ngs/alignment/fastq/
```

Now set up a directory to do this alignment, with symbolic links to the **bowtie2** index directory and the directory containing the **FASTQ** to align:

```
mkdir -p $SCRATCH/core_ngs/alignment/vibCho
cd $SCRATCH/core_ngs/alignment/vibCho
ln -sf ../../references/bt2/vibCho
ln -sf ../../alignment/fastq fq
```

We'll be aligning the *V. cholerae* reads now in **./fq/cholera_rnaseq.fastq.gz** (how many sequences does it contain?)

Note that here the data is from standard mRNA sequencing, meaning that the DNA ***fragments*** are typically longer than the ***reads***. There is likely to be very little contamination that would require using a local rather than global alignment, or many other pre-processing steps (e.g. adapter trimming). Thus, we will run **bowtie2** with default parameters, omitting options other than the input, output, and reference index. This performs a ***global*** alignment.

As you can tell from looking at the **bowtie2** help message, the general syntax looks like this:

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

So execute this **bowtie2** global, single-end alignment command:

```
mkdir -p $SCRATCH/core_ngs/references/fasta
cp $CORENGS/references/fasta/vibCho*  $SCRATCH/core_ngs/references/fasta/

mkdir -p $SCRATCH/core_ngs/references/bt2/vibCho
cp $CORENGS/references/bt2/vibCho/*.*  $SCRATCH/core_ngs/references/bt2/vibCho/

mkdir -p $SCRATCH/core_ngs/alignment/vibCho
cd $SCRATCH/core_ngs/alignment/vibCho
ln -sf ../../references/bt2/vibCho
ln -sf ../../alignment/fastq fqmkdir -p $SCRATCH/core_ngs/alignment/vibCho
```

```
cd $SCRATCH/core_ngs/alignment/vibCho
bowtie2 -x vibCho/vibCho.O395 -U fq/cholera_rnaseq.fastq.gz \
  -S cholera_rnaseq.sam 2>&1 | tee aln_global.log
```

Notes:

- **-x vibCho/vibCho.O395.fa** – prefix path of inde**x** files
- **-U fq/cholera_rnaseq.fastq.gz** – FASTQ file for single-end (**U**npaired) alignment
- **-S cholera_rnaseq.sam** – tells **bowtie2** to report alignments in **SAM** format to the specified file
- **2>&1** redirects ***standard error*** to ***standard output***
    - while the alignment data is being written to the **cholera_rnaseq.sam** file, **bowtie2** will report its progress to ***standard error***.
- **| tee aln.log** takes the **bowtie2** progress output and pipes it to the **tee** program
    - **tee** takes its ***standard input*** and writes it to the specified file and also to ***standard output***
    - that way, you can see the progress output now, but also save it to review later (or supply to **MultiQC**)

Since the **FASTQ** file is not large, this should not take too long, and you will see progress output like this:

```
89006 reads; of these:
  89006 (100.00%) were unpaired; of these:
    5902 (6.63%) aligned 0 times
    51483 (57.84%) aligned exactly 1 time
    31621 (35.53%) aligned >1 times
93.37% overall alignment rate
```

When the job is complete you should have a **cholera_rnaseq.sam** file that you can examine using whatever commands you like. Remember, to further process it downstream, you should create a sorted, indexed **BAM** file from this **SAM** output.

Exercise: Repeat the alignment performing a local alignment, and write the output in BAM format. How do the alignment statistics compare?

--local

```
module load samtools
cd $SCRATCH/core_ngs/alignment/vibCho
bowtie2 --local -x vibCho/vibCho.O395 -U fq/cholera_rnaseq.fastq.gz 2>aln_local.log | \
  samtools view -b > cholera_rnaseq.local.bam
```

Reports these alignment statistics:

```
89006 reads; of these:
  89006 (100.00%) were unpaired; of these:
    13359 (15.01%) aligned 0 times
    46173 (51.88%) aligned exactly 1 time
    29474 (33.11%) aligned >1 times
84.99% overall alignment rate
```

Interestingly, the local alignment rate here is lower than we saw with the global alignment. Usually local alignments have higher alignment rates than corresponding global ones.

Exercise #5: BWA-MEM - Human mRNA-seq

After **bowtie2** came out with a local alignment option, it wasn't long before **bwa** developed its own local alignment algorithm called BWA-MEM (for **M**aximal xact **M**atches), implemented by the **bwa mem** command.

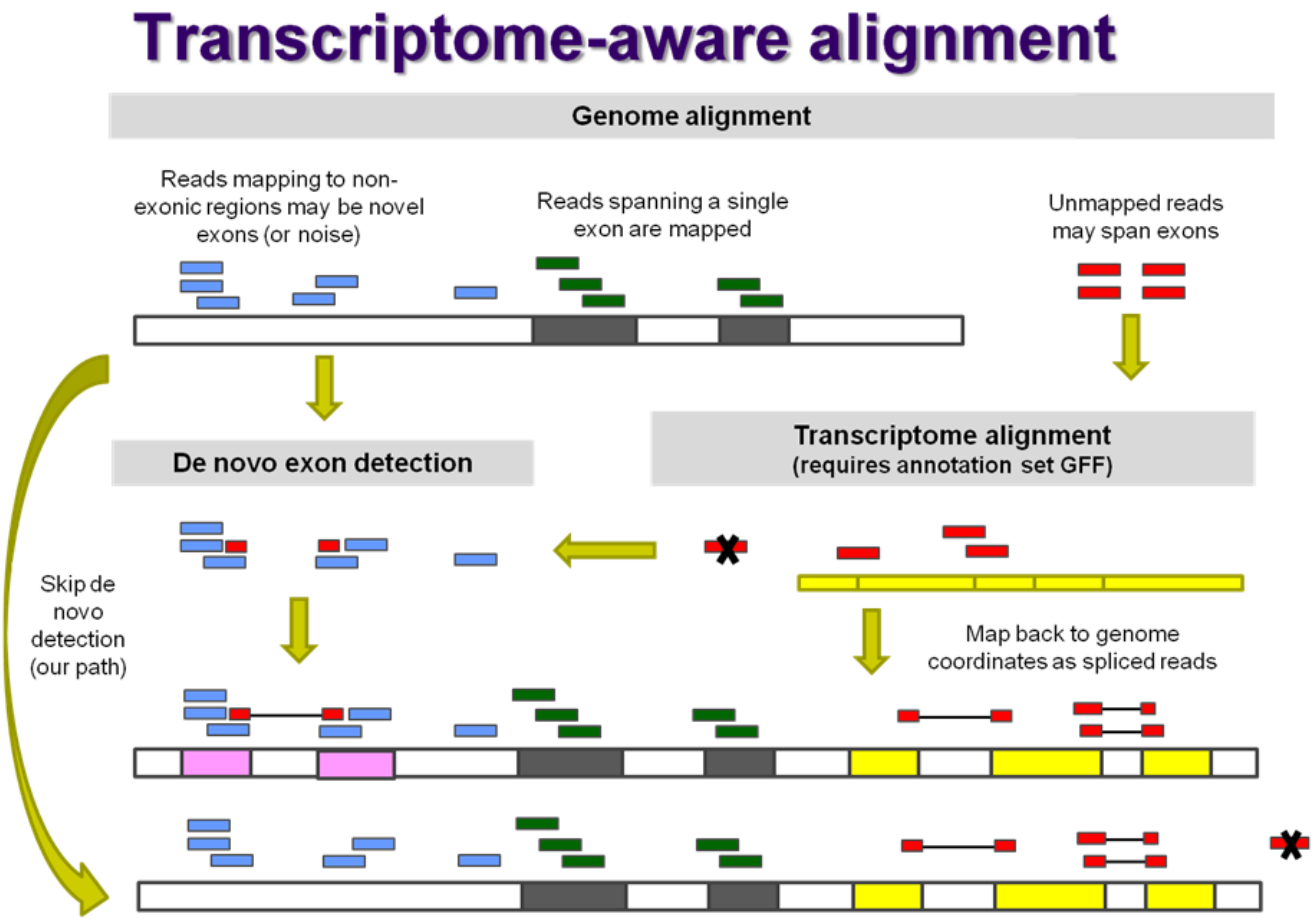
bwa mem has the following advantages:

- It provides the simplicity of using **bwa** without the complexities of local alignment
- It can align ***different portions of a read to different locations on the genome***
    - In a total RNA-seq experiment, reads will (at some frequency) span a splice junction themselves
        - or a pair of reads in a paired-end library will fall on either side of a splice junction.
    - We want to be able to align these splice-adjacent reads for many reasons, from accurate transcript quantification to novel fusion transcript discovery.

This exercise will align a human total RNA-seq dataset that includes numerous reads that cross splice junctions.

A word about real splice-aware aligners

Using **bwa mem** for RNA-seq alignment is sort of a "poor man's" RNA-seq alignment method. Real splice-aware aligners like **tophat2, hisat2** or **STAR** hav e more complex algorithms (as shown below) – and take a lot more time!

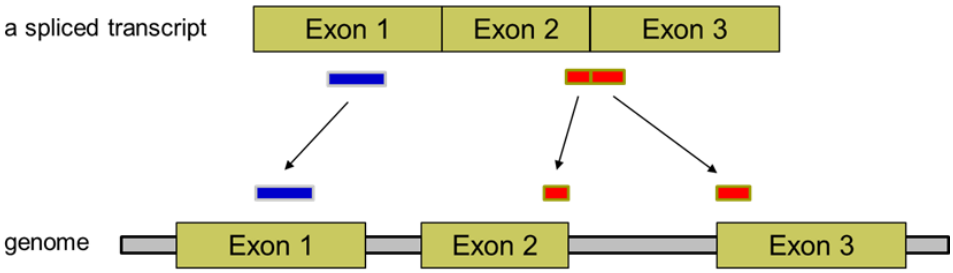


based on Kim *et al. Genome Biology* 2013, 14:R36

In the transcriptome-aware alignment above, reads that span splice junctions are reported in the **SAM** file with ***genomic coordinates*** that start in the first exon and end in the second exon (the **CIGAR** string uses the **N** operator, e.g. 30**M**1000**N**60**M**).

BWA MEM does not know about the exon structure of the genome. But it can align different sub-sections of a read to two different locations, producing ***two*** alignment records from ***one*** input read (one of the two will be marked as ***secondary*** (**0x100** flag).

BWA MEM splits junction-spanning reads into two alignment records



Setup for BWA mem

First set up our working directory for this alignment. Since it takes a long time to build a **bwa** index for a large genome (here human **hg38**/**GRCh38**), we'll use one that the BioITeam maintains in its **/work/projects/BioITeam/ref_genome** area.

Run multiple alignments using the TACC batch system

```
# Make sure you're in an idev session
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Thu
# or
idev -m 90 -N 1 -A OTH21164 -p development

# Load the modules we'll need
module load biocontainers
module load bwa
module load samtools

# Copy over the FASTQ data if needed
mkdir -p $SCRATCH/core_ngs/alignment/fastq
cp $CORENGS/alignment/*.gz $SCRATCH/core_ngs/alignment/fastq/

# Make a new alignment directory for running these scripts
cds
mkdir -p core_ngs/alignment/bwamem
cd core_ngs/alignment/bwamem
ln -sf ../fastq
ln -sf /work/projects/BioITeam/ref_genome/bwa/bwtsw/hg38
```

Now take a look at **bwa mem** usage (type **bwa mem** with no arguments). The most important parameters are the following:

Option	Effect
-k	Controls the minimum seed length (default = 19)
-w	Controls the "gap bandwidth", or the length of a maximum gap. This is particularly relevant for MEM, since it can determine whether a read is split into two separate alignments or is reported as one long alignment with a long gap in the middle (default = 100)
-M	For split reads, mark the shorter read as secondary
-r	Controls how long an alignment must be relative to its seed before it is re-seeded to try to find a best-fit local match (default = 1.5, e.g. the value of -k multiplied by 1.5)
-c	Controls how many matches a MEM must have in the genome before it is discarded (default = 10000)
-t	Controls the number of threads to use

RNA-seq alignment with bwa mem

Based on its help info, this is the structure of the **bwa mem** command we will use:

```
bwa mem -M <ref.fa> <reads.fq> > outfile.sam
```

After performing the setup above, execute the following command in your **idev** session:

```
cd $SCRATCH/core_ngs/alignment/bwamem
bwa mem -M hg38/hg38.fa fastq/human_rnaseq.fastq.gz 2>hs_rna.bwamem.log |
  samtools view -b | \
  samtools sort -O BAM -T human_rnaseq.tmp > human_rnaseq.sort.bam
```

This multi-pipe command performs three steps:

- The **bwa mem** alignment
  - the program's progress output (on ***standard error***) is redirected to a log file (**2>hs_rna.bwamem.log**)
  - its alignment records (on ***standard output***) is piped to the next step (conversion to **BAM**)
- Conversion of **bwa mem**'s **SAM** output to **BAM** format
  - recall that the **-b** option to **samtools view** says to output in **BAM** format
- Sorting the **BAM** file
  - **samtools sort** takes the binary output from **samtools view** and writes a sorted **BAM** file.

Because the progress output is being redirected to a log file, we won't see anything until the command completes. Then you should have a **human_rnaseq.sort.bam** file and an **hs_rna.bwamem.log** logfile.

Exercise: Compare the number of original FASTQ reads to the number of alignment records.

Use the **zcat** | **wc -l** | **awk** idiom to count FASTQ reads.

Use **samtools flagstat** to report alignment statistics.
Count the **FASTQ** file reads:

```
cd $SCRATCH/core_ngs/alignment/bwamem
zcat ./fastq/human_rnaseq.fastq.gz | wc -l | awk '{print $1/4}'
```

The file has 100,000 reads.

Generate alignment statistics from the sorted BAM file:

```
cd $SCRATCH/core_ngs/alignment/bwamem
samtools flagstat human_rnaseq.sort.bam | tee hs_rnaseq.flagstat.txt
```

Output will look like this:

```
133570 + 0 in total (QC-passed reads + QC-failed reads)
33570 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
133450 + 0 mapped (99.91% : N/A)
0 + 0 paired in sequencing
0 + 0 read1
0 + 0 read2
0 + 0 properly paired (N/A : N/A)
0 + 0 with itself and mate mapped
0 + 0 singletons (N/A : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

There were 133,570 alignment records reported for the 100,000 input reads.

Because **bwa mem** can split reads and report two alignment records ***for the same read***, there are 33,570 ***secondary*** reads reported here.

Be aware that some downstream tools (for example the **Picard** suite, often used before SNP calling) do not like it when a read name appears more than once in the **SAM** file. Such reads can be filtered, but only if they can be identified as ***secondary*** by specifying the **bwa mem -M** option as we did above. This option reports the longest alignment normally but marks additional alignments for the read as secondary (the **0x100** BAM flag). This designation also allows you to easily filter out the secondary reads with **samtools view -F 0x104** if desired.