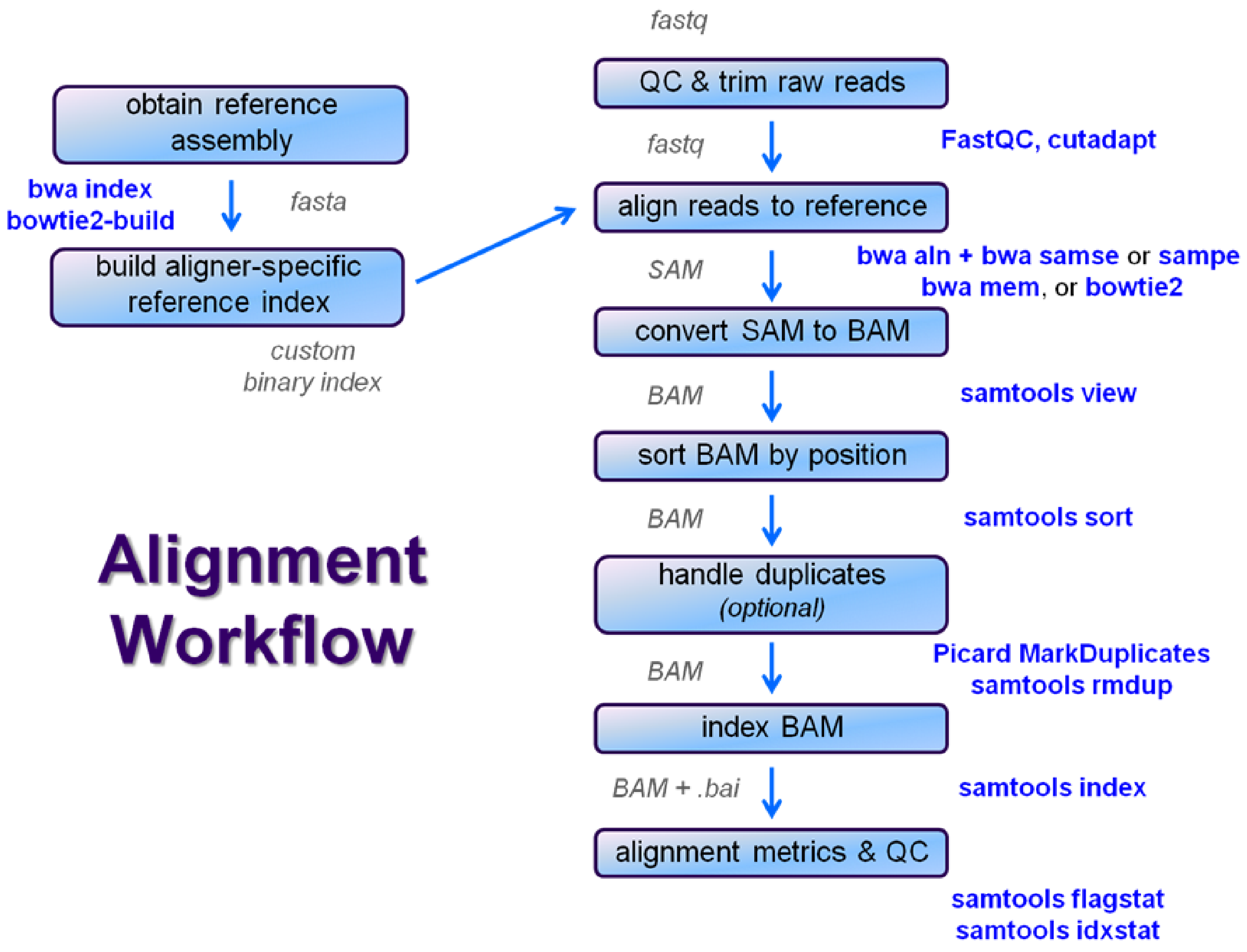
The Basic Alignment Workflow

Reservations

Use our summer school *reservation* (**CoreNGSday4**) when submitting batch jobs to get higher priority on the **ls6** normal queue ***today***:

```
sbatch --reservation=CoreNGSday4 <batch_file>.slurm
idev -m 180 -N 1 -A OTH21164 -r CoreNGSday4
```

Overview



After raw sequence files are generated (in **FASTQ** format), quality-checked, and pre-processed in some way, the next step in many NGS pipelines is mapping to a reference genome.

For individual sequences it is common to use a tool like **BLAST** to identify genes or species of origin. However a normal NGS dataset will have tens to hundreds of millions of sequences, which **BLAST** and similar tools are not designed to handle. Thus a large set of computational tools have been developed to quickly align each read to its best location (if any) in a reference.

Even though many mapping tools exist, a few individual programs have a dominant "market share" of the NGS world. In this section, we will primarily focus on two of the most versatile general-purpose ones: **BWA** and **Bowtie2** (the latter being part of the **Tuxedo** suite which includes the transcriptome-aware RNA-seq aligner **Tophat2** as well as other downstream quantifiaction tools).

Stage the alignment data

First connect to **ls6.tacc.utexas.edu** and start an **idev** session. This should be second nature by now 🙂

Start an idev session

```
idev -m 180 -N 1 -A OTH21164 -r CoreNGSday4
```

Then stage the sample datasets and references we will use.

Get the alignment exercises files

```
# Copy the FASTA files for building references
mkdir -p $SCRATCH/core_ngs/references/fasta
cp $CORENGS/references/fasta/*.fa   $SCRATCH/core_ngs/references/fasta/

# Copy the FASTQ files that will be used for alignment
mkdir -p $SCRATCH/core_ngs/alignment/fastq
cp $CORENGS/alignment/*fastq.gz $SCRATCH/core_ngs/alignment/fastq/
cd $SCRATCH/core_ngs/alignment/fastq
```

These are descriptions of the **FASTQ** files we copied:

File Name	Description	Sample
Sample_Yeast_L005_R1.cat.fastq.gz	Paired-end Illumina, First of pair, FASTQ	Yeast ChIP-seq
Sample_Yeast_L005_R2.cat.fastq.gz	Paired-end Illumina, Second of pair, FASTQ	Yeast ChIP-seq
human_rnaseq.fastq.gz	Paired-end Illumina, First of pair only, FASTQ	Human RNA-seq
human_mirnaseq.fastq.gz	Single-end Illumina, FASTQ	Human microRNA-seq
cholera_rnaseq.fastq.gz	Single-end Illumina, FASTQ	*V. cholerae* RNA-seq

Reference Genomes

Before we get to alignment, we need a reference to align to. This is usually an organism's genome, but can also be ***any set of names sequences***, such as a transcriptome or other set of genes.

Here are the four reference genomes we will be using today, with some information about them. These are not necessarily the most recent versions of these references (e.g. the newest human reference genome is **hg38** and the most recent **miRBase** annotation is **v21**. (See here for information about many more genomes.)

Reference	Species	Base Length	Contig Number	Source	Download
hg19	Human	3.1 Gbp	25 (really 93)	**UCSC**	UCSC GoldenPath
sacCer3	Yeast	12.2 Mbp	17	**UCSC**	UCSC GoldenPath
mirbase v20	Human subset	160 Kbp	1908	**miRBase**	miRBase Downloads
vibCho (O395)	*Vibrio cholerae*	~4 Mbp	2	**GenBank**	GenBank Downloads

Searching genomes is computationally hard work and takes a long time if done on linear genomic sequence. So aligners require that references first be ***ind exed*** to accelerate lookup. The aligners we are using each require a different index, but use the same method (the Burrows-Wheeler Transform) to get the job done.

Building a reference index involves taking a **FASTA** file as input, with each ***contig*** (contiguous string of bases, e.g. a chromosome) as a separate **FASTA** entry, and producing an aligner-specific set of files as output. Those output index files are then used to perform the sequence alignment, and alignments are reported using coordinates referencing names and offset positions based on the original **FASTA** file contig entries.

We can quickly index the references for the yeast genome, the human miRNAs, and the *V. cholerae* genome, because they are all small, so we'll build each index from the appropriate **FASTA** files right before we use them.

hg19 is way too big for us to index here so we will use an existing set of **BWA** **hg19** index files located at:

BWA hg19 index location

```
/work/projects/BioITeam/ref_genome/bwa/bwtsw/hg19
```

The BioITeam maintains a set of reference indexes for many common organisms and aligners. They can be found in aligner-specific sub-directories of the **/work/projects/BioITeam/ref_genome** area. E.g.:

```
/work/projects/BioITeam/ref_genome/
   bowtie2/
   bwa/
   hisat2/
   kallisto/
   star/
   tophat/
```

Exploring FASTA with grep

It is often useful to know what chromosomes/contigs are in a **FASTA** file before you start an alignment so that you're familiar with the contig naming convention – and to verify that it's the one you expect. For example, chromosome 1 is specified differently in different references and organisms: **chr1** (**US CS** human), **chrI** (**UCSC** yeast), or just **1** (**Ensembl** human GRCh37).

A **FASTA** file consists of a number of contig name entries, each one starting with a right carat (**>**) character, followed by many lines of base characters. E. g.:

```
>chrI
CCACACCACACCCACACACCCACACACCACACCACACACCACACCACACC
CACACACACACATCCTAACACTACCCTAACACAGCCCTAATCTAACCCTG
GCCAACCTGTCTCTCAACTTACCCTCCATTACCCTGCCTCCACTCGTTAC
CCTGTCCCATTCAACCATACCACTCCGAACCACCATCCATCCCTCTACTT
```

How do we dig out just the lines that have the contig names and ignore all the sequences? Well, the contig name lines all follow the pattern above, and since the **>** character is not a valid base, it will never appear on a sequence line.

We've discovered a ***pattern*** (also known as a ***regular expression***) to use in searching, and the command line tool that does regular expression matching is **grep** (**g**eneral **r**egular **e**xpression **p**arser). Read more about grep here: Advanced commands: grep.

Regular expressions are so powerful that nearly every modern computer language includes a "***regex***" module of some sort. There are many online tutorials for regular expressions, and several slightly different "flavors" of them. But the most common is the **Perl** style (http://perldoc.perl.org/perlretut.html), which was one of the fist and still the most powerful (there's a reason **Perl** was used extensively when assembling the human genome). We're only going to use simple regular expressions here, but learning more about them will pay handsome dividends for you in the future.

Here's how to execute **grep** to list contig names in a **FASTA** file.

grep to match contig names in a FASTA file

```
# If you haven't staged the fasta files
cds
mkdir -p core_ngs/references/fasta
cd core_ngs/references/fasta
cp $CORENGS/references/fasta/*.fa .

cd $SCRATCH/core_ngs/references/fasta
grep -P '^>' sacCer3.fa | more
```

Notes:

- The **-P** option tells **grep** to **Perl**-style regular expression patterns.
    - This makes including special characters like Tab (**\t**), Carriage Return (**\r**) or Linefeed (**\n**) much easier that the default POSIX paterns.
    - While it is not required here, it generally doesn't hurt to include this option.
- **'^>'** is the regular expression describing the pattern we're looking for (described below)
- **sacCer3.fa** is the file to search.
    - lines with text that ***match*** our pattern will be written to ***standard output***
    - non matching lines will be omitted
- We pipe to **more** just in case there are a lot of contig names.

Now down to the nuts and bolts of the pattern: **'^>'**

First, the ***single quotes*** around the pattern – this tells the **bash** shell to pass the ***exact string contents*** to **grep**.

As part of its friendly command line parsing and evaluation, the shell will often look for special characters on the command line that mean something to it (for example, the **$** in front of an environment variable name, like in **$SCRATCH**). Well, regular expressions treat the **$** specially too – but in a completely different way! Those ***single quotes*** tell the shell "don't look inside here for special characters – treat this as a literal string and pass it to the program". The shell will obey, will strip the single quotes off the string, and will pass the actual pattern, **^>**, to the **grep** program. (Note that the shell ***does*** look inside double quotes (**"**) for certain special signals, such as looking for environment variable names to evaluate. Read more about Quoting in the shell.)

So what does **^>** mean to **grep**? We know that contig name lines always start with a **>** character, so **>** is a ***literal*** for **grep** to use in its pattern match.

We might be able to get away with just using this literal alone as our regex, specifying **'>'** as the command line argument. But for **grep**, the more specific the pattern, the better. So we ***constrain*** where the **>** can appear on the line. The special carat (**^**) ***metacharacter*** represents "***beginning of line***". So **^>** means "beginning of a line followed by a **>** character".

```
# Copy the FASTA files for building references
mkdir -p $SCRATCH/core_ngs/references/fasta
cp $CORENGS/references/fasta/*.fa $SCRATCH/core_ngs/references/fasta/
```

Exercise: How many contigs are there in the sacCer3 reference?

```
cd $SCRATCH/core_ngs/references/fasta
grep -P '^>' sacCer3.fa | wc -l
```

Or use **grep**'s **-c** option that says "just **c**ount the line matches"

```
grep -P -c '^>' sacCer3.fa
```

There are 17 contigs.

Aligner overview

There are many aligners available, but we will concentrate on two of the most popular general-purpose ones: **bwa** and **bowtie2**. The table below outlines the available protocols for them.

alignment type	aligner options	pro's	con's

global with **bwa**	**single end reads**: • **bwa aln <R1>** • **bwa samse** **paired end reads**: • **bwa aln <R1>** • **bwa aln <R2>** • **bwa sampe**	• simple to use (take default options) • good for basic ***global*** alignment	• multiple steps needed
global with **bowtie2**	**bowtie2**	• extremely configurable • can be used for RNAseq alignment (after adapter trimming) because of its many options	• complex (many options)
local with **bwa**	**bwa mem**	• simple to use (take default options) • very fast • no adapter trimming needed • good for simple RNAseq analysis • the secondary alignments it reports provide splice junction information	• ***always*** produces alignments with secondary reads • must be filtered if not desired
local with **bowtie2**	**bowtie2 --local**	• extremely configurable • no adapter trimming needed • good for small RNA alignment because of its many options	• complex – many options

Exercise #1: BWA global alignment – Yeast ChIP-seq

Overview ChIP-seq alignment workflow with BWA

We will perform a ***global alignment*** of the paired-end Yeast ChIP-seq sequences using **bwa**. This workflow has the following steps:

1. Trim the **FASTQ** sequences down to 50 with **fastx_clipper**
   - this removes most of any 5' adapter contamination without the fuss of specific adapter trimming w/**cutadapt**
2. Prepare the **sacCer3** reference index for **bwa** using **bwa index**
   - this is done once, and re-used for later alignments
3. Perform a global **bwa** alignment on the R1 reads (**bwa aln**) producing a BWA-specific binary **.sai** intermediate file
4. Perform a global **bwa** alignment on the R2 reads (**bwa aln**) producing a BWA-specific binary **.sai** intermediate file
5. Perform pairing of the separately aligned reads and report the alignments in SAM format using **bwa sampe**
6. Convert the SAM file to a BAM file (**samtools view**)
7. Sort the BAM file by genomic location (**samtools sort**)
8. Index the BAM file (**samtools index**)
9. Gather simple alignment statistics (**samtools flagstat** and **samtools idxstat**)

We're going to skip the trimming step for now and see how it goes. We'll perform steps 2 - 5 now and leave **samtools** for a later exercise since steps 6 - 10 are common to nearly all post-alignment workflows.

Introducing BWA

Like other tools you've worked with so far, you first need to load **bwa**. Do that now, and then enter **bwa** with no arguments to view the top-level help page (many NGS tools will provide some help when called with no arguments). **bwa** is available as a **BioContainers**. module.

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGSday4
```

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

BWA suite usage

```
Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.17-r1188
Contact: Heng Li <lh3@sanger.ac.uk>

Usage:   bwa <command> [options]

Command: index         index sequences in the FASTA format
         mem           BWA-MEM algorithm
         fastmap       identify super-maximal exact matches
         pemerge       merge overlapping paired ends (EXPERIMENTAL)
         aln           gapped/ungapped alignment
         samse         generate alignment (single ended)
         sampe         generate alignment (paired ended)
         bwasw         BWA-SW for long queries

         shm           manage indices in shared memory
         fa2pac        convert FASTA to PAC format
         pac2bwt       generate BWT from PAC
         pac2bwtgen    alternative algorithm for generating BWT
         bwtupdate     update .bwt to the new format
         bwt2sa        generate SA from BWT and Occ

Note: To use BWA, you need to first index the genome with `bwa index'.
      There are three alignment algorithms in BWA: `mem', `bwasw', and
      `aln/samse/sampe'. If you are not sure which to use, try `bwa mem'
      first. Please `man ./bwa.1' for the manual.
```

As you can see, **bwa** include many ***sub-commands*** that perform the tasks we are interested in.

Building the BWA sacCer3 index

We will index the genome with the **bwa index** command. Type **bwa index** with no arguments to see usage for this sub-command.

bwa index usage

```
Usage:   bwa index [options] <in.fasta>

Options: -a STR    BWT construction algorithm: bwtsw, is or rb2 [auto]
         -p STR    prefix of the index [same as fasta name]
         -b INT    block size for the bwtsw algorithm (effective with -a bwtsw) [10000000]
         -6        index files named as <in.fasta>.64.* instead of <in.fasta>.*

Warning: `-a bwtsw' does not work for short genomes, while `-a is' and
         `-a div' do not work not for long genomes.
```

Based on the usage description, we only need to specify two things:

- The name of the **FASTA** file
- Whether to use the **bwtsw** or **is** algorithm for indexing

Since **sacCer3** is relative large (~12 Mbp) we will specify **bwtsw** as the indexing option (as indicated by the "Warning" message), and the name of the **FASTA** file is **sacCer3.fa**.

The output of this command is a ***group*** of files that are all required together as the index. So, within our **references** directory, we will create another directory called **references/bwa/sacCer3** and build the index there. To remind ourselves which **FASTA** was used to build the index, we create a symbolic link to our **references/fasta/sacCer3.fa** file (note the use of the **../.. *relative path*** syntax).

Get the alignment exercises files

```
mkdir -p $SCRATCH/core_ngs/alignment/fastq
mkdir -p $SCRATCH/core_ngs/references/fasta
cp $CORENGS/alignment/*fastq.gz   $SCRATCH/core_ngs/alignment/fastq/
cp $CORENGS/references/fasta/*.fa $SCRATCH/core_ngs/references/fasta/
```

Prepare BWA reference directory for sacCer3

```
mkdir -p $SCRATCH/core_ngs/references/bwa/sacCer3
cd $SCRATCH/core_ngs/references/bwa/sacCer3
ln -s ../../fasta/sacCer3.fa
ls -l
```

Now execute the **bwa index** command.

Build BWA index for sacCer3

```
bwa index -a bwtsw sacCer3.fa
```

Since the yeast genome is not large when compared to human, this should not take long to execute (otherwise we would do it as a batch job). When it is complete you should see a set of index files like this:

BWA index files for sacCer3

```
sacCer3.fa
sacCer3.fa.amb
sacCer3.fa.ann
sacCer3.fa.bwt
sacCer3.fa.pac
sacCer3.fa.sa
```

Performing the bwa alignment

Now, we're ready to execute the actual alignment, with the goal of initially producing a **SAM** file from the input **FASTQ** files and reference. First prepare a directory for this exercise and link the **sacCer3** reference directories there (this will make our commands more readable).

```
# Copy the FASTA files for building references
mkdir -p $SCRATCH/core_ngs/references
cp $CORENGS/references/fasta/*.fa $SCRATCH/core_ngs/references/fasta/

# Copy a pre-built bwa index for sacCer3
mkdir -p $SCRATCH/core_ngs/references/bwa/sacCer3
cp $CORENGS/references/bwa/sacCer3/*.* $SCRATCH/core_ngs/references/bwa/sacCer3/

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

Prepare to align yeast data

```
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
ln -s -f ../fastq
ln -s -f ../../references/bwa/sacCer3
```

As our workflow indicated, we first use **bwa aln** on the R1 and R2 **FASTQ**s, producing a BWA-specific **.sai** intermediate binary files.

What does **bwa aln** needs in the way of arguments?

```
bwa aln
```

There are lots of options, but here is a summary of the most important ones.

Option	Effect
-l	Specifies the length of the seed (default = 32)
-k	Specifies the number of mismatches allowable in the seed of each alignment (default = 2)
-n	Specifies the number of mismatches (or fraction of bases in a given alignment that can be mismatches) in the ***entire*** alignment (including the seed) (default = 0.04)
-t	Specifies the number of threads

Other options control the details of how much a mismatch or gap is penalized, limits on the number of acceptable hits per read, and so on. Much more information can be found on the BWA manual page.

For a basic alignment like this, we can just go with the default alignment parameters.

Note that **bwa** writes its (binary) output to ***standard output*** by default, so we need to ***redirect*** that to a **.sai** file.

For simplicity, we will just execute these commands directly, one at a time. Each command should only take few minutes and you will see **bwa**'s progress messages in your terminal.

bwa aln commands for yeast R1 and R2

```
# If not already loaded:
module load biocontainers
module load bwa

cd $SCRATCH/core_ngs/alignment/yeast_bwa
bwa aln sacCer3/sacCer3.fa fastq/Sample_Yeast_L005_R1.cat.fastq.gz > yeast_pe_R1.sai
bwa aln sacCer3/sacCer3.fa fastq/Sample_Yeast_L005_R2.cat.fastq.gz > yeast_pe_R2.sai
```

When all is done you should have two **.sai** files: **yeast_pe_R1.sai** and **yeast_pe_R2.sai**.

Make sure your output files are not empty

Double check that output was written by doing **ls -lh** and making sure the file sizes listed are not 0.

Exercise: How long did it take to align the R2 file?

The last few lines of bwa's execution output should look something like this:

```
[bwa_aln] 17bp reads: max_diff = 2
[bwa_aln] 38bp reads: max_diff = 3
[bwa_aln] 64bp reads: max_diff = 4
[bwa_aln] 93bp reads: max_diff = 5
[bwa_aln] 124bp reads: max_diff = 6
[bwa_aln] 157bp reads: max_diff = 7
[bwa_aln] 190bp reads: max_diff = 8
[bwa_aln] 225bp reads: max_diff = 9
[bwa_aln_core] calculate SA coordinate... 50.76 sec
[bwa_aln_core] write to the disk... 0.07 sec
[bwa_aln_core] 262144 sequences have been processed.
[bwa_aln_core] calculate SA coordinate... 50.35 sec
[bwa_aln_core] write to the disk... 0.07 sec
[bwa_aln_core] 524288 sequences have been processed.
[bwa_aln_core] calculate SA coordinate... 13.64 sec
[bwa_aln_core] write to the disk... 0.01 sec
[bwa_aln_core] 592180 sequences have been processed.
[main] Version: 0.7.17-r1188
[main] CMD: /usr/local/bin/bwa aln sacCer3/sacCer3.fa fastq/Sample_Yeast_L005_R1.cat.fastq.gz
[main] Real time: 78.185 sec; CPU: 77.598 sec
```

So the R2 alignment took ~78 seconds (~1.3 minutes).

Since you have your own private compute node, you can use all its resources. It has 128 cores, so re-run the R2 alignment asking for 60 execution threads.

```
bwa aln -t 60 sacCer3/sacCer3.fa fastq/Sample_Yeast_L005_R2.cat.fastq.gz > yeast_pe_R2.sai
```

Exercise: How much of a speedup did you seen when aligning the R2 file with 60 threads?

The last few lines of **bwa**'s execution output should look something like this:

```
[bwa_aln] 17bp reads: max_diff = 2
[bwa_aln] 38bp reads: max_diff = 3
[bwa_aln] 64bp reads: max_diff = 4
[bwa_aln] 93bp reads: max_diff = 5
[bwa_aln] 124bp reads: max_diff = 6
[bwa_aln] 157bp reads: max_diff = 7
[bwa_aln] 190bp reads: max_diff = 8
[bwa_aln] 225bp reads: max_diff = 9
[bwa_aln_core] calculate SA coordinate... 266.70 sec
[bwa_aln_core] write to the disk... 0.04 sec
[bwa_aln_core] 262144 sequences have been processed.
[bwa_aln_core] calculate SA coordinate... 268.94 sec
[bwa_aln_core] write to the disk... 0.03 sec
[bwa_aln_core] 524288 sequences have been processed.
[bwa_aln_core] calculate SA coordinate... 72.26 sec
[bwa_aln_core] write to the disk... 0.01 sec
[bwa_aln_core] 592180 sequences have been processed.
[main] Version: 0.7.17-r1188
[main] CMD: /usr/local/bin/bwa aln -t 60 sacCer3/sacCer3.fa fastq/Sample_Yeast_L005_R2.cat.fastq.gz
[main] Real time: 5.013 sec; CPU: 142.813 sec
```

So the R2 alignment took only ~5 seconds (real time), or 15+ times as fast as with only one processing thread.

Note, though, that the CPU time with 60 threads was greater (142.8 sec) than with only 1 thread (77.6 sec). That's because of the thread management overhead when using multiple threads.

Next we use the **bwa sampe** command to pair the reads and output **SAM** format data. Just type that command in with no arguments to see its usage.

For this command you provide the same reference index prefix as for **bwa aln**, along with the two **.sai** files and the two original **FASTQ** files. Also, **bwa** writ es its output to ***standard output***, so redirect that to a **.sam** file.

Here is the command line statement you need. Just execute it on the command line.

```
# Copy the FASTA files for building references
mkdir -p $SCRATCH/core_ngs/references
cp $CORENGS/references/fasta/*.fa $SCRATCH/core_ngs/references/fasta/

# Copy a pre-built bwa index for sacCer3
mkdir -p $SCRATCH/core_ngs/references/bwa/sacCer3
cp $CORENGS/references/bwa/sacCer3/*.* $SCRATCH/core_ngs/references/bwa/sacCer3/

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

# Stage the BWA .sai files
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
ln -sf ../fastq
ln -sf ../../references/bwa/sacCer3
cp $CORENGS/catchup/yeast_bwa/*.sai .
```

Pairing of BWA R1 and R2 aligned reads

```
cd $SCRATCH/core_ngs/alignment/yeast_bwa
bwa sampe sacCer3/sacCer3.fa yeast_pe_R1.sai yeast_pe_R2.sai \
  fastq/Sample_Yeast_L005_R1.cat.fastq.gz \
  fastq/Sample_Yeast_L005_R2.cat.fastq.gz > yeast_pe.sam
```

You should now have a **SAM** file (**yeast_pe.sam**) that contains the alignments. It's just a text file, so take a look with **head**, **more**, **less**, **tail**, or whatever you feel like. Later you'll learn additional ways to analyze the data with **samtools** once you create a **BAM** file.

Exercise: What kind of information is in the first lines of the SAM file?

The **SAM** file has a number of header lines, which all start with an at sign (@).

The **@SQ** lines describe each contig (chromosome) and its length.

There is also a **@PG** line that describes the way the **bwa sampe** was performed.

Exercise: How many alignment records (not header records) are in the SAM file?

This looks for the pattern **'^HWI'** which is the start of every read name (which starts every alignment record).
Remember **-c** says just count the records, don't display them.

```
grep -P -c '^HWI' yeast_pe.sam
```

Or use the **-v** (in**v**ert) option to tell **grep** to print all lines that ***don't*** match a particular pattern; here, all header lines, which start with @.

```
grep -P -v -c '^@' yeast_pe.sam
```

There are 1,184,360 alignment records.

Exercise: How many sequences were in the R1 and R2 FASTQ files combined?

```
zcat fastq/Sample_Yeast_L005_R[12].cat.fastq.gz | wc -l | awk '{print $1/4}'
```

There were a total of 1,184,360 original sequences (R1s + R2s)

Exercises:

- **Do both R1 and R2 reads have separate alignment records?**
- **Does the SAM file contain both mapped and un-mapped reads?**
- **What is the order of the alignment records in this SAM file?**

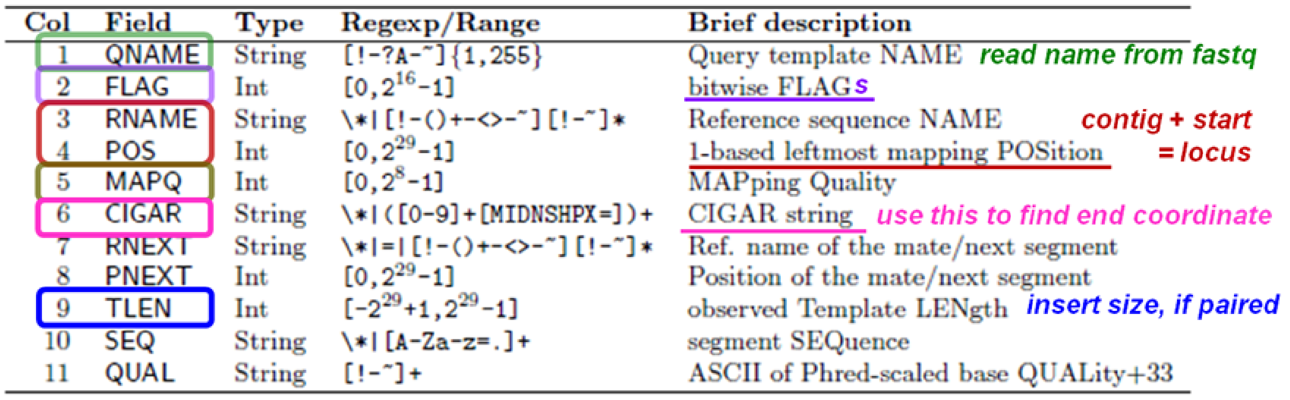
Both R1 and R2 reads must have separate alignment records, because there were 1,184,360 R1+R2 reads and the same number of alignment records.

The SAM file must contain both mapped and unmapped reads, because there were 1,184,360 R1+R2 reads and the same number of alignment records.

Alignment records occur in the same read-name order as they did in the **FASTQ**, except that they come in pairs. The R1 read comes 1st, then the corresponding R2. This is called ***read name ordering***.

Using cut to isolate fields

Recall the format of a **SAM** alignment record:



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 FLAGs	
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	

Suppose you wanted to look only at field 3 (contig name) values in the SAM file. You can do this with the handy **cut** command. Below is a simple example where you're asking **cut** to display the 3rd column value for the last 10 alignment records.

```
# Stage the aligned SAM file
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
cp $CORENGS/catchup/yeast_bwa/yeast_pe.sam .
```

Cut syntax for a single field

```
tail yeast_pe.sam | cut -f 3
```

By default **cut** assumes the field delimiter is **Tab**, which is the delimiter used in the majority of NGS file formats. You can specify a different delimiter with the **-d** option.

You can also specify a range of fields, and mix adjacent and non-adjacent fields. This displays fields 2 through 6, field 9:

Cut syntax for multiple fields

```
tail -20 yeast_pe.sam | cut -f 2-6,9
```

You may have noticed that some alignment records contain contig names (e.g. **chrV**) in field 3 while others contain an asterisk (*****). The ***** means the record didn't map. We're going to use this heuristic along with cut to see about how many records represent aligned sequences. (Note this is not the strictly correct method of finding unmapped reads because not all unmapped reads have an asterisk in field 3. Later you'll see how to properly distinguish between mapped and unmapped reads using **samtools**.)

First we need to make sure that we don't look at fields in the **SAM** header lines. We're going to end up with a series of pipe operations, and the best way to make sure you're on track is to enter them one at a time piping to **head**:

Grep pattern that doesn't match header

```
# the ^@ pattern matches lines starting with @ (only header lines),
# and -v says output lines that don't match
grep -v -P '^@' yeast_pe.sam | head
```

Ok, it looks like we're seeing only alignment records. Now let's pull out only field 3 using **cut**:

Get contig name info with cut

```
grep -v -P '^@' yeast_pairedend.sam | cut -f 3 | head
```

Cool, we're only seeing the contig name info now. Next we use **grep** again, piping it our contig info and using the **-v** (in**v**ert) switch to say print lines that ***don't*** match the pattern:

Filter contig name of * (unaligned)

```
grep -v -P '^@' yeast_pe.sam | cut -f 3 | grep -v '*' | head
```

Perfect! We're only seeing real contig names that (usually) represent aligned reads. Let's count them by piping to **wc -l** (and omitting omit **head** of course – we want to count everything).

Count aligned SAM records

```
grep -v -P '^@' yeast_pe.sam | cut -f 3 | grep -v '*' | wc -l
```

Exercise: About how many records represent aligned sequences? What alignment rate does this represent?

The expression above returns 612,968. There were 1,184,360 records total, so the percentage is:

Calculate alignment rate

```
awk 'BEGIN{print 612968/1184360}'
```

or about 51%. Not great.

Note we perform this calculation in **awk**'s **BEGIN** block, which is always executed, instead of the body block, which is only executed for lines of input. And here we call **awk** without piping it any input. See Linux fundamentals: cut,sort,uniq,grep,awk

Exercise: What might we try in order to improve the alignment rate?

Recall that these are 100 bp reads and we did not remove adapter contamination. There will be a distribution of fragment sizes – some will be short – and those short fragments may not align without adapter removal (e.g. with **fastx_trimmer).**

Exercise #2: Basic SAMtools Utilities

The **SAMtools** program is a commonly used set of tools that allow a user to manipulate **SAM**/**BAM** files in many different ways, ranging from simple tasks (like **SAM/BAM** format conversion) to more complex functions (like sorting, indexing and statistics gathering). It is available in the TACC module system (as well as in **BioContainers**). Load that module and see what **samtools** has to offer:

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGSday4
```

```
# If not already loaded
module load biocontainers  # takes a while

module load samtools
samtools
```

SAMtools suite usage

```
Program: samtools (Tools for alignments in the SAM format)
Version: 1.9 (using htslib 1.9)

Usage:   samtools <command> [options]

Commands:
  -- Indexing
     dict           create a sequence dictionary file
     faidx          index/extract FASTA
     fqidx          index/extract FASTQ
     index          index alignment

  -- Editing
     calmd          recalculate MD/NM tags and '=' bases
     fixmate        fix mate information
     reheader       replace BAM header
     targetcut      cut fosmid regions (for fosmid pool only)
     addreplacerg   adds or replaces RG tags
     markdup        mark duplicates

  -- File operations
     collate        shuffle and group alignments by name
     cat            concatenate BAMs
     merge          merge sorted alignments
     mpileup        multi-way pileup
     sort           sort alignment file
     split          splits a file by read group
     quickcheck     quickly check if SAM/BAM/CRAM file appears intact
     fastq          converts a BAM to a FASTQ
     fasta          converts a BAM to a FASTA

  -- Statistics
     bedcov         read depth per BED region
     coverage       alignment depth and percent coverage
     depth          compute the depth
     flagstat       simple stats
     idxstats       BAM index stats
     phase          phase heterozygotes
     stats          generate stats (former bamcheck)

  -- Viewing
     flags          explain BAM flags
     tview          text alignment viewer
     view           SAM<->BAM<->CRAM conversion
     depad          convert padded BAM to unpadded BAM
```

In this exercise, we will explore five utilities provided by **samtools**: **view**, **sort, index**, **flagstat**, and **idxstats**. Each of these is executed in one line for a given **SAM/BAM** file. In the **SAMtools**/**BEDtools** sections tomorrow we will explore **samtools** in more depth.

Know your samtools version!

There are two main "eras" of **SAMtools** development:

- "original" samtools
    - v **0.1.19** is the last stable version
- "modern" samtools
    - v 1.0, 1.1, 1.2 – avoid these (very buggy!)
    - v **1.3**+ – finally stable!

Unfortunately, ***some functions with the same name*** in both version eras have ***different options and arguments***! So be sure you know which version you're using. (The **samtools** version is usually reported at the top of its usage listing).

TACC **BioContainers** also offers the original **samtools** version: **samtools/ctr-0.1.19--3**.

samtools view

The **samtools view** utility provides a way of converting between **SAM** (text) and **BAM** (binary, compressed) format. It also provides many, many other functions which we will discuss lster. To get a preview, execute **samtools view** without any other arguments. You should see:

samtools view usage

```
Usage: samtools view [options] <in.bam>|<in.sam>|<in.cram> [region ...]

Options:
  -b       output BAM
  -C       output CRAM (requires -T)
  -1       use fast BAM compression (implies -b)
  -u       uncompressed BAM output (implies -b)
  -h       include header in SAM output
  -H       print SAM header only (no alignments)
  -c       print only the count of matching records
  -o FILE  output file name [stdout]
  -U FILE  output reads not selected by filters to FILE [null]
  -t FILE  FILE listing reference names and lengths (see long help) [null]
  -X       include customized index file
  -L FILE  only include reads overlapping this BED FILE [null]
  -r STR   only include reads in read group STR [null]
  -R FILE  only include reads with read group listed in FILE [null]
  -d STR:STR
           only include reads with tag STR and associated value STR [null]
  -D STR:FILE
           only include reads with tag STR and associated values listed in
           FILE [null]
  -q INT   only include reads with mapping quality >= INT [0]
  -l STR   only include reads in library STR [null]
  -m INT   only include reads with number of CIGAR operations consuming
           query sequence >= INT [0]
  -f INT   only include reads with all  of the FLAGs in INT present [0]
  -F INT   only include reads with none of the FLAGS in INT present [0]
  -G INT   only EXCLUDE reads with all  of the FLAGs in INT present [0]
  -s FLOAT subsample reads (given INT.FRAC option value, 0.FRAC is the
           fraction of templates/read pairs to keep; INT part sets seed)
  -M       use the multi-region iterator (increases the speed, removes
           duplicates and outputs the reads as they are ordered in the file)
  -x STR   read tag to strip (repeatable) [null]
  -B       collapse the backward CIGAR operation
  -?       print long help, including note about region specification
  -S       ignored (input format is auto-detected)
  --no-PG  do not add a PG line
      --input-fmt-option OPT[=VAL]
               Specify a single input file format option in the form
               of OPTION or OPTION=VALUE
  -O, --output-fmt FORMAT[,OPT[=VAL]]...
               Specify output format (SAM, BAM, CRAM)
      --output-fmt-option OPT[=VAL]
               Specify a single output file format option in the form
               of OPTION or OPTION=VALUE
  -T, --reference FILE
               Reference sequence FASTA FILE [null]
  -@, --threads INT
               Number of additional threads to use [0]
      --write-index
               Automatically index the output files [off]
      --verbosity INT
               Set level of verbosity
```

That is a lot to process! For now, we just want to read in a **SAM** file and output a **BAM** file. The input format is auto-detected, so we don't need to specify it (although you do in v0.1.19). We just need to tell the tool to output the file in **BAM** format, and to include the header records.

Get the alignment exercises files

```
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
cp $CORENGS/catchup/yeast_bwa/yeast_pe.sam .
```

Convert SAM to binary BAM

```
cd $SCRATCH/core_ngs/alignment/yeast_bwa
samtools view -b yeast_pe.sam > yeast_pe.bam
```

- the **-b** option tells the tool to output **BAM** format

How do you look at the **BAM** file contents now? That's simple. Just use **samtools view** without the **-b** option. Remember to pipe output to a pager!

View BAM records

```
samtools view yeast_pe.bam | more
```

Notice that this does not show us the header record we saw at the start of the **SAM** file.

Exercise: What samtools view option will include the header records in its output? Which option would show only the header records?

samtools view -h shows header records ***along with*** alignment records.

samtools view -H shows ***header records only***.

samtools sort

Looking at some of the alignment record information (e.g. **samtools view yeast_pairedend.bam | cut -f 1-4 | more**), you will notice that read names appear in adjacent pairs (for the R1 and R2), in the same order they appeared in the original **FASTQ** file. Since that means the corresponding mappings are in no particular order, searching through the file very inefficient. **samtools sort** re-orders entries in the **SAM** file either by ***locus*** (***contig name*** + ***coordinate position***) or by ***read name***.

If you execute **samtools sort** without any options, you see its help page:

samtools sort usage

```
Usage: samtools sort [options...] [in.bam]
Options:
  -l INT     Set compression level, from 0 (uncompressed) to 9 (best)
  -m INT     Set maximum memory per thread; suffix K/M/G recognized [768M]
  -n         Sort by read name
  -t TAG     Sort by value of TAG. Uses position as secondary index (or read name if -n is set)
  -o FILE    Write final output to FILE rather than standard output
  -T PREFIX  Write temporary files to PREFIX.nnnn.bam
      --input-fmt-option OPT[=VAL]
               Specify a single input file format option in the form
               of OPTION or OPTION=VALUE
  -O, --output-fmt FORMAT[,OPT[=VAL]]...
               Specify output format (SAM, BAM, CRAM)
      --output-fmt-option OPT[=VAL]
               Specify a single output file format option in the form
               of OPTION or OPTION=VALUE
      --reference FILE
               Reference sequence FASTA FILE [null]
  -@, --threads INT
               Number of additional threads to use [0]
```

In most cases you will be sorting a **BAM** file from ***name order*** to ***locus order***. You can use either **-o** or redirection with **>** to control the output.

```
# Stage the aligned yeast SAM and BAM files
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
cp $CORENGS/catchup/yeast_bwa/yeast_pe.[bs]am .
```

To sort the paired-end yeast **BAM** file by position, and get a **BAM** file named **yeast_pe.sort.bam** as output, execute the following command:

Sort a BAM file

```
cd $SCRATCH/core_ngs/alignment/yeast_bwa
samtools sort -O bam -T yeast_pe.tmp yeast_pe.bam > yeast_pe.sort.bam
```

- The **-O** options says the **O**utput format should be **BAM**
- The **-T** options gives a prefix for **T**emporary files produced during sorting
  - sorting large **BAM**s will produce ***many*** temporary files during processing
  - ***make sure the temporary file prefix is different from the input BAM file prefix!***
- By default **sort** writes its output to ***standard output***, so we use **>** to redirect to a file named **yeast_pairedend.sort.bam**

Exercise: Compare the file sizes of the yeast_pe .sam, .bam, and .sort.bam files and explain why they are different.

```
ls -lh yeast_pe*
```

The **yeast_pe.sam** text file is the largest at ~348 MB.

The name-ordered binary **yeast_pe.bam** text file only about 1/3 that size, ~111 MB. They contain ***exactly*** the same records, in the same order, but conversion from text to binary results in a much smaller file.

The coordinate-ordered binary **yeast_pe.sort.bam** file is even slightly smaller, ~92 MB. This is because **BAM** files are actually ***customized*** **gzip**-format files. The customization allows blocks of data (e.g. all alignment records for a contig) to be represented in an even more compact form. You can read more about this in section 4 of the SAM format specification.

samtools index

Many tools (like **IGV**, the **I**ntegrative **G**enomics **V**iewer) only need to use portions of a **BAM** file at a given point in time. For example, if you are viewing alignments that are within a particular gene, alignment records on other chromosomes do not need to be loaded. In order to speed up access, **BAM** files are ***indexed***, producing **BAI** files which allow fast random access. This is especially important when you have many alignment records.

The utility **samtools index** creates an index that has the same name as the input **BAM** file, with suffix **.bai** appended. Here's the **samtools index** usage:

samtools index usage

```
Usage: samtools index [-bc] [-m INT] <in.bam> [out.index]
Options:
  -b       Generate BAI-format index for BAM files [default]
  -c       Generate CSI-format index for BAM files
  -m INT   Set minimum interval size for CSI indices to 2^INT [14]
  -@ INT   Sets the number of threads [none]
```

The syntax here is way, way easier. We want a **BAI**-format index which is the default. (CSI-format is used with extremely long contigs, which don't apply here - the most common use case is for polyploid plant genomes).

So all we have to provide is the ***sorted*** BAM:

Index a sorted bam

```
samtools index yeast_pe.sort.bam
```

This will produce a file named **yeast_pe.bam.bai**.

Most of the time when an index is required, it will be automatically located as long as it is in the ***same directory*** as its **BAM** file and shares the same name up until the **.bai** extension.

Exercise: Compare the sizes of the sorted BAM file and its BAI index.

```
ls -lh yeast_pe.sort.bam*
```

While the **yeast_pe.sort.bam** file is ~92 **M**B, its index (**yeast_pe.sort.bai**) is only 20 **K**B.

samtools flagstat

Since the **BAM** file contains records for both mapped and unmapped reads, just counting records doesn't provide information about the ***mapping rate*** of our alignment. The **samtools flagstat** tool provides a simple analysis of mapping rate based on the the **SAM** flag fields.

Here's how to run **samtools flagstat** and both see the output in the terminal and save it in a file – the **samtools flagstat** ***standard output*** is piped to **tee**, which both writes it to the specified file and sends it to its ***standard output***:

```
# Stage the aligned yeast SAM and BAM files
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
cp $CORENGS/catchup/yeast_bwa/yeast_pe.sort.bam* .
```

Run samtools flagstat using tee

```
samtools flagstat yeast_pe.sort.bam | tee yeast_pe.flagstat.txt
```

You should see something like this:

samtools flagstat output

```
1184360 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
547664 + 0 mapped (46.24% : N/A)
1184360 + 0 paired in sequencing
592180 + 0 read1
592180 + 0 read2
473114 + 0 properly paired (39.95% : N/A)
482360 + 0 with itself and mate mapped
65304 + 0 singletons (5.51% : N/A)
534 + 0 with mate mapped to a different chr
227 + 0 with mate mapped to a different chr (mapQ>=5)
```

Ignore the "**+ 0**" addition to each line - that is a carry-over convention for counting "QA-failed reads" that is no longer relevant.

The most important statistic is the mapping rate (here 46%) but this readout also allows you to verify that some common expectations (e.g. that about the same number of R1 and R2 reads aligned, and that most mapped reads are proper pairs) are met.

Exercise: What proportion of mapped reads were properly paired?

Divide the number of properly paired reads by the number of mapped reads:

```
awk 'BEGIN{ print 473114 / 547664 }'
# or
echo $(( 473114 * 100 / 547664 ))
# or
echo "473114 547664" | awk '{printf("%0.1f%%\n", 100*$1/$2)}'
```

About 86% of mapped read were properly paired. This is actually a bit on the low side for ChIP-seq alignments which typically over 90%.

samtools idxstats

More information about the alignment is provided by the **samtools idxstats** report, which shows how many reads aligned to each contig in your reference. Note that **samtools idxstats** must be run on a ***sorted, indexed BAM*** file.

```
# Stage the aligned yeast SAM and BAM files
mkdir -p $SCRATCH/core_ngs/alignment/yeast_bwa
cd $SCRATCH/core_ngs/alignment/yeast_bwa
cp $CORENGS/catchup/yeast_bwa/yeast_pe.sort.bam* .
```

Use samtools idxstats to summarize mapped reads by contig

```
samtools idxstats yeast_pe.sort.bam | tee yeast_pe.idxstats.txt
```

Here we use the **tee** command which reports its ***standard input*** to ***standard output*** before also writing it to the specified file.

samtools idxstats output

```
chrI	230218	8820	1640
chrII	813184	36616	4026
chrIII	316620	13973	1530
chrIV	1531933	72675	8039
chrV	576874	27466	2806
chrVI	270161	10866	1222
chrVII	1090940	50893	5786
chrVIII	562643	24672	3273
chrIX	439888	16246	1739
chrX	745751	31748	3611
chrXI	666816	28017	2776
chrXII	1078177	54783	10124
chrXIII	924431	40921	4556
chrXIV	784333	33070	3703
chrXV	1091291	48714	5150
chrXVI	948066	44916	5032
chrM	85779	3268	291
*	0	0	571392
```

The output has four tab-delimited columns:

1. contig name
2. contig length
3. number of mapped reads
4. number of unmapped reads

The reason that the "unmapped reads" field for named chromosomes is not zero is that the aligner may initially assign a potential mapping (contig name and start coordinate) to a read, but then mark it later as unampped if it does meet various quality thresholds.

If you're mapping to a non-genomic reference such as miRBase miRNAs or another set of genes (a transcriptome), **samtools idxstats** gives you a quick look at quantitative alignment results.