Bi188 2013

Introduction to bioinformatics: formats, packages and pipelines.

Contents

1 Introductory notes

Real-life bioinformatics practice consists of the following components: 1) transferring data from one format to another so that code can run on it and useful information extracted out of it; 2) writing code; 3) filtering out artifacts and finding bugs in yours or other people's code; 4) the glorious moments of real biological discovery. The latter are rare, and because of that, all the more rewarding when they do occur, but the bulk of one's time is occupied with 1) and 3). Since you have several other classes to worry about in addition to Bi188, and the last thing we would like to accomplish by making you do these exercises is to discourage you from learning more about these topics and possibly working in the field some day, we

will try to save you as much effort as possible that you would otherwise have to expend figuring these things out on your own, which is not always possible anyway - the best way to learn is to figure things out on your own, however, information is not always readily available and in such cases you need to learn directly from someone knowledgeable. This is not the case here - none of the information presented here belongs to that category of hidden knowledge - but in practice, a lot of often critical information is hidden in the code and difficult to unearth.

Note that the following is by no means meant to be an exhaustive presentation of all commonly used file formats, software packages and analysis pipelines in genomics, let alone bioinformatics in general. The goal is to get you familiar with those you will be working with during the course of the class so that you do not make silly mistakes due to misunderstanding of file formats (which happens a lot in real life because the general trend is for format specifications, even for very commonly used formats, to not be exactly highly visible, readily accessible, or even sufficiently complete; on top of that, the problem that people rarely read them carefully is added). We will try to avoid all these problems by providing you not only the information we think you will need plus some snippets of code, which will be given to you together with the data for each assignment.

2 A warning about genomic positions

It is a sad fact of life that two different conventions for storing genomic positions exist in the various file formats. A number of file formats (for example, GTF and VCF) use 1-based coordinates, i.e. the first base in each chromosome is position 1, while others use 0-based coordinates. Everything in Python is 0-based, in particular the list and string data structures you will be extensively using and this is something you should be aware of at all times, otherwise you will commit the infamous "off-by-1" error that has been plaguing bioinformatics for as long as it has existed.

3 Sequence storage formats

Most of genomics revolves around working with strings of nucleotides. Two main formats have emerged over the years for storing DNA sequence, FASTA and FASTQ.

3.1 FASTA

The FASTA format is what is usually used for storing the sequences of genomes, transcripts and other pieces of genomic sequence as well as proteins. It consists of a header line containing the description of the sequence that follows, and always starting with the `'>'` symbol, followed by one or more lines containing the actual sequence (for whole chromosomes, it is always multiple lines, as many programs have a limit to the length of lines they can handle and chromosomes can be huge - the human chr1 is more than 200Mb). Here is an example of what the beginning of the human mitochondrial chromosome looks like in FASTA format:

```
more chrM.fa
>chrM
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCAT
TTGGTATTTTCGTCTGGGGGGTGTGCACGCGATAGCATTGCGAGACGCTG
GAGCCGGAGCACCCTATGTCGCAGTATCTGTCTTTGATTCCTGCCTCATT
CTATTATTTATCGCACCTACGTTCAATATTACAGGCGAACATACCTACTA
AAGTGTGTTAATTAATTAATGCTTGTAGGACATAATAATAACAATTGAAT
GTCTGCACAGCCGCTTTCCACACAGACATCATAACAAAAAATTTCCACCA
```

3.2 FASTQ

The FASTQ format [2] has become the standard for storing sequencing reads. Note that there is a fundamental difference between sequencing reads and genomic sequence and it is that sequencing reads are much more likely to contain errors. You should not understand this as meaning that genome sequences do not contain errors, they do, but they are generally rare and we assume that they are correct; in contrast sequencing errors are common and it is important to take them into account. You should also be aware that sequencing platforms do not output the actual sequence of the read, but rather a probabilistic estimate of what the sequence is (if you want to know why that is, you should do some exploration on how the Illumina technology, or even the classic Sanger sequencing, works). Most of the time the probability is very high and with the improvement of sequencing technologies error rates have been steadily going down. Still, it is necessary to capture that probability and this is done by the addition of quality scores to the sequence in the form of FASTQ files. There are several different quality scores used in practice (which has caused widespread confusion before their recent standardization), but the most common one is the following:

$$Q_{sanger} = -10 \log_{10} p$$

where p is the probability that the base call is correct. The quality scores are then encoded in the form of ASCI symbols corresponding to their integer values.

Here is an example of an Illumina sequencing read in FASTQ format:

```
@ILLUMINA-EAS295:72:70H93AAXX:7:1:19686:1557 1:Y:0:
AACTAAAAAGCTCATTGGAAGCTATCAGGGTGCAGT
+
@.3)3@:9=:A##########################
```

Each FASTQ entry consists of 4 lines. The first one is a header containing information about the read, similar to the FASTA format, but starting with `'@'` instead of `'>'`. This is followed by the read sequence, by a line separating the sequence from the quality score, starting with `'+'` (which can optionally also contain the header information), and finally by the quality scores (note that the quality scores can also contain the `'@'` symbol so you should not rely on lines starting with it to figure out where a new read entry is starting).

The Wikipedia article on FASTQ is quite informative if you want to know more about it: `http://en.wikipedia.org/wiki/FASTQ_format`

4 The SAM/BAM format for storing sequence alignments

In the early years of high-throughput sequencing, a wide range of different formats for storing read alignments against a reference genome were used (typically each of the numerous mapping programs had its own format). Later, a common format was established, and this was the SAM format [11] (where SAM stands for Sequence Alignment/Map) and its more commonly used compressed binary version BAM. A widely used set of tools for manipulating SAM and BAM files called `samtools` exists, which you will be using extensively and which we will discuss in a later section. In addition, a very useful python module for working with BAM files has been developed called `pysam`; you will not be required to use it to go deep into the intricacies of BAM files, but it would nevertheless be useful to familiarize yourself with what they look like. The specifications for the SAM format are available from the class website (`http://woldlab.caltech.edu/bi188/private/PSets/SAM1.pdf`).

4.1 The samtools package for working with BAM files

The main tool for manipulating BAM files, and one you will be using extensively is `samtools` [11]. It consists of the following programs:

```
samtools

Usage:   samtools <command> [options]

Command: view        SAM<->BAM conversion
         sort        sort alignment file
         pileup      generate pileup output
         mpileup     multi-way pileup
         depth       compute the depth
         faidx       index/extract FASTA
         tview       text alignment viewer
         index       index alignment
         idxstats    BAM index stats (r595 or later)
         fixmate     fix mate information
         glfview     print GLFv3 file
         flagstat    simple stats
         calmd       recalculate MD/NM tags and '=' bases
         merge       merge sorted alignments
         rmdup       remove PCR duplicates
         reheader    replace BAM header
         cat         concatenate BAMs
         targetcut   cut fosmid regions (for fosmid pool only)
         phase       phase heterozygotes
```

You will be using the following programs during the course of the class (more examples can be found in the description of the variant calling pipeline you will be running):

1. `samtools view` converts a SAM file into a BAM file and the other way around.

   ```
   Usage:   samtools view [options] <in.bam>|<in.sam> [region1 [...]]

   Options: -b       output BAM
            -h       print header for the SAM output
            -H       print header only (no alignments)
            -S       input is SAM
            -u       uncompressed BAM output (force -b)
            -1       fast compression (force -b)
            -x       output FLAG in HEX (samtools-C specific)
            -X       output FLAG in string (samtools-C specific)
            -c       print only the count of matching records
            -L FILE  output alignments overlapping the input BED FILE [null]
            -t FILE  list of reference names and lengths (force -S) [null]
            -T FILE  reference sequence file (force -S) [null]
            -o FILE  output file name [stdout]
   ```

```
-R FILE  list of read groups to be outputted [null]
-f INT   required flag, 0 for unset [0]
-F INT   filtering flag, 0 for unset [0]
-q INT   minimum mapping quality [0]
-l STR   only output reads in library STR [null]
-r STR   only output reads in read group STR [null]
-?       longer help
```

2. `samtools sort` sorts a BAM file by chromosome and chromosome position:

   ```
   samtools sort [-on] [-m <maxMem>] <in.bam> <out.prefix>
   ```

3. `samtools index` indexes a BAM file for fast random access:

   ```
   Usage: samtools index <in.bam> [out.index]
   ```

4. `samtools idxstats` will tell you how many alignments there are for each chromosome in the BAM file (it only works on indexed files)

   ```
   Usage: samtools idxstats <in.bam>
   ```

5. `samtools rmdup` will remove apparent PCR duplicates (see discussion on this topic later for more information)

   ```
   Usage:  samtools rmdup [-sS] <input.srt.bam> <output.bam>

   Option: -s    rmdup for SE reads
           -S    treat PE reads as SE in rmdup (force -s)
   ```

6. `samtools mpileup` is a SNV and indel variant caller. It can work on multiple BAM files in the same time and does all sorts of useful things. Refer to the variant discovery pipeline for the exact parameters we will be using.

   ```
   Usage: samtools mpileup [options] in1.bam [in2.bam [...]]

   Input options:

          -6           assume the quality is in the Illumina-1.3+ encoding
          -A           count anomalous read pairs
          -B           disable BAQ computation
          -b FILE      list of input BAM files [null]
          -d INT       max per-BAM depth to avoid excessive memory usage [250]
          -E           extended BAQ for higher sensitivity but lower specificity
          -f FILE      faidx indexed reference sequence file [null]
          -G FILE      exclude read groups listed in FILE [null]
          -l FILE      list of positions (chr pos) or regions (BED) [null]
   ```

```
        -M INT       cap mapping quality at INT [60]
        -r STR       region in which pileup is generated [null]
        -q INT       skip alignments with mapQ smaller than INT [0]
        -Q INT       skip bases with baseQ/BAQ smaller than INT [13]

    Output options:

        -D           output per-sample DP in BCF (require -g/-u)
        -g           generate BCF output (genotype likelihoods)
        -S           output per-sample strand bias P-value in BCF (require -g/-u)
        -u           generate uncompress BCF output

    SNP/INDEL genotype likelihoods options (effective with '-g' or '-u'):

        -e INT       Phred-scaled gap extension seq error probability [20]
        -F FLOAT     minimum fraction of gapped reads for candidates [0.002]
        -h INT       coefficient for homopolymer errors [100]
        -I           do not perform indel calling
        -L INT       max per-sample depth for INDEL calling [250]
        -m INT       minimum gapped reads for indel candidates [1]
        -o INT       Phred-scaled gap open sequencing error probability [40]
        -P STR       comma separated list of platforms for indels [all]
```

You can find more information about `samtools` here: `http://samtools.sourceforge.net/samtools.shtml`

5 VCF format for storing sequence variant calls

The VCF (Variant Call Format) [3] is the standard format for storing sequence variants relative to the reference. Here is an example of what a portion of a VCF file looks like (from [5]):

```
#various header lines
#CHROM  POS      ID  REF         ALT        QUAL  FILTER  INFO  FORMAT    SAMPLE1           SAMPLE2
1       899752   .   G           A          189   .       .     GT:PL:GQ  0/1:219,0,248:99  1/0:219,0,248:99
1       1334052  .   CTAGAGTAGA  CTAGA      217   .       .     GT:PL:GQ  0/1:255,0,255:99  1/1:219,0,248:99
1       5964854  .   C           T,A        222   .       .     GT:PL:GQ  1/1:255,0,253:99  2/2:219,0,248:99
1       38027866 .   GGGTGG      GGGTGGTGG  19.5  .       .     GT:PL:GQ  1/1:255,0,253:99  1/0:219,0,248:99
```

Note that in this example, the VCF file contains the genotypes for two samples, but there is no limitations to how many samples can be annotated in a single VCF. The 1000 Genome projects has produced gigantic VCF files with thousands of columns while dbSNP has no individuals in its VCF files (as it does not contain any information about specific samples, just the commonly found SNPs in human populations).

Here are some explanations about what each column means. Note that you will not have to worry about what everything means, for your purposes it will be sufficient to look at the `#CHROM`, `POS`, `REF`, `ALT`, `FORMAT`, and sample fields, you can ignore the others.

The first column contains the chromosome; usually it is encoded as simply a number or letter, without the `'chr'` letters in front, while pretty much all other formats use the `chr` + number/letter format, so beware of that.

The second column contains the position of the variant, where the position is defined as the position in the genome of the first base in the `REF` field in the fourth column (1-based).

The third column contains the variant ID (each SNP in dbSNP is assigned an unique ID).

The fourth column contains the reference allele.

The fifth column contains the alternative alleles. Note that this can contain more than one variant, and if this is the case, they are comma-separated (the third variant shown above is an example of this)

The ninth column specifies the format in which genotypes (and other information you need not worry about) is specified for each sample. The various fields are separated by the `':'` symbol, and usually, the genotype is the first subfield, indicated as `'GT'`.

As mentioned above, the sample fields contain information about the genotype of each sample, in the same order as in the `FORMAT` field, and also separated by the `':'` symbol. The genotype field is specified in the form of `'/'`-separated numbers of variants for the two alleles (always in the same parental order), where the number 0 means the allele is the reference one, the number 1 means the allele is the first one in the `ALT` field, 2 that the allele is the second one in the `ALT` field, etc.

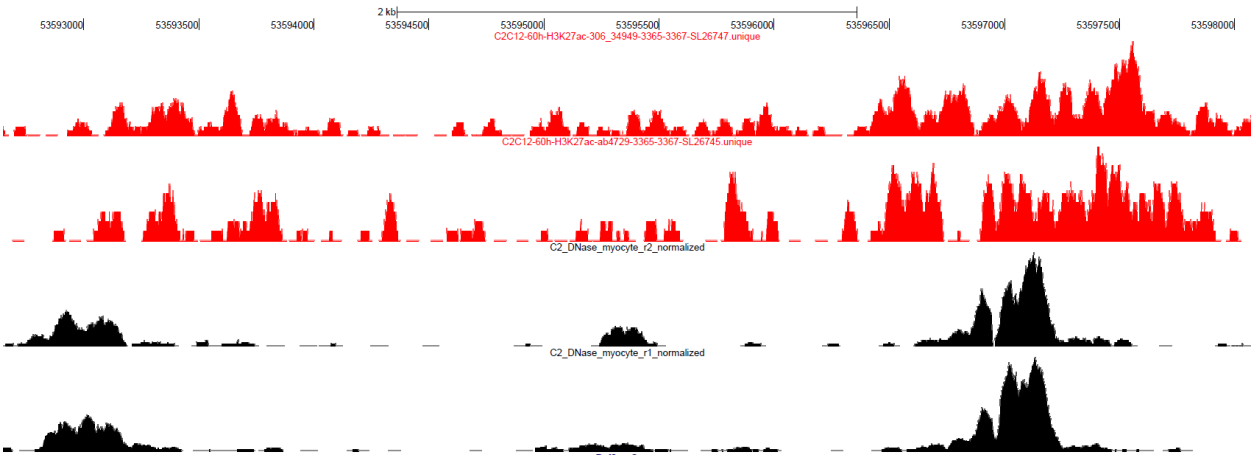
In the example provided above, we have shown a SNP, a deletion, another SNP, this time with two alternative alleles, and an insertion.

You can find more information about the format by visiting the following links: `http://vcftools.sourceforge.net/VCF-poster.pdf` and also at `http://www.1000genomes.org/wiki/Analysis/Variant%20Call%20Format/vcf-variant-call-format-version-41`, or reading the paper describing it

6 Genome browser tracks and genome annotation formats

The following describes the most common formats used for visualizing information on genome browsers, which can also double as annotation formats (GTF in particular). You can read more about them at `http://genome.ucsc.edu/FAQ/FAQformat.html`.

6.1 bedGraph/wiggle



The `bedGraph` format is used for display of continuous-value information along the genome, for example read coverage in ChIP-seq/RNA-seq/DNAse-seq/etc. experiments, sequence conservation and many other types of data. The `wiggle` format does the same but is formatted differently (and is more difficult to parse) so we will only talk about `bedGraph` here. Both formats associate a float value with each genomic position (but you do not need a separate line for each position, if a stretch of genomic positions has the

same value, they can be specified with a single line, see below). An example is shown above and here is what a portion of such a file looks like:

```
chr2    5152    5323    36.0
chr2    5323    5324    35.0
chr2    5324    5359    34.0
chr2    5359    5360    35.0
chr2    5360    8876    36.0
```

6.2 BED

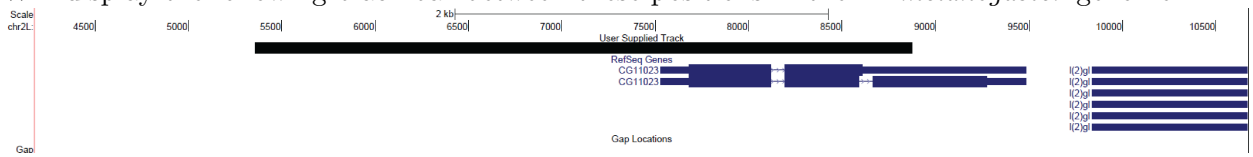
In contrast to the `bedGraph` format, which does not allow for overlapping features, the BED format can be used to specify an arbitrary number (and positions) of features. It is the most basic format for specifying a genomic region and in its minimum version (so called `BED3`) consists of the following fields:

```
#chr    left_coordinate    right_coordinate
```

For example:

```
chr2L    5360    8876
```

Will display the following black bar between these positions in the *D. melanogaster* genome:



There are a number of additional fields that can be added, for example, strand information, and several different versions of the BED format (`BED4`, `BED6`, `BED9`, `BED12`) exist, which you can read about on the UCSC website: `https://bison.cacr.caltech.edu/FAQ/FAQformat.html#format1`.

6.3 Conversion from bedGraph/wiggle and bed to bigBed and bigWig

Historically, the way to display tracks on the UCSC genome browser was to upload them onto it. However, the size of the track files can be quite large and as more and more data was generated, this became impractical. This necessitated the development of the `bigBed` and `bigWig` formats [6], which are broadly speaking, compressed binary version of the `bedGraph/wiggle` and `BED` files which can be accessed and displayed remotely by the browser. In practice, this works as follows: you generate your tracks, convert them to `bigBed/bigWig`, and put them in a location on your system that is visible to the outside world, you load them onto the browser (but not physically, just as links to them), and then the browser reads them and displays them over the internet.

6.3.1 Conversion from bedGraph/wiggle to bigWig

```
$wigToBigWig track.wig genome.chrom.sizes track.bigWig
```

Where the `chrom.sizes` files contain the name of each chromosome and its length, as follows (this is the female version of the human genome):

```
chr1    249250621
chr2    243199373
chr3    198022430
chr4    191154276
chr5    180915260
chr6    171115067
chr7    159138663
chr8    146364022
chr9    141213431
chr10   135534747
chr11   135006516
chr12   133851895
chr13   115169878
chr14   107349540
chr15   102531392
chr16   90354753
chr17   81195210
chr18   78077248
chr19   59128983
chr20   63025520
chr21   48129895
chr22   51304566
chrX    155270560
chrM    16571
```

6.3.2 Conversion from BED to bigBed

```
$bedToBigBed track.bed genome.chrom.sizes track.bigBed
```

6.3.3 Loading of custom tracks onto the UCSC Genome Browser

You can load the tracks onto the USCS Genome Browser by pasting the following into the custom track loading field:

```
track type=bigWig name=name_of_your_track description=description_of_your_track
visibility=full color=255,0,0 bigDataUrl=URL_of_your_track

track type=bigBed name=name_of_your_track description=description_of_your_track
visibility=full color=255,0,0 bigDataUrl=URL_of_your_track
```

6.4 GTF

The Gene Transfer Format (GTF) is the most commonly used format for storing and visualizing genome annotations (transcript models and certain features in them). Here are the specifications: `http://mblab.wustl.edu/GTF22.html` and here is what portion of one (in this case, the GENCODE V13 annotation of the human genome) looks like:

```
##description: evidence-based annotation of the human genome (GRCh37), version 13 (Ensembl 68)
##provider: GENCODE
##contact: gencode@sanger.ac.uk
##format: gtf
##date: 2012-08-09
chr1    HAVANA  gene    11869   14412   .       +       .       gene_id "ENSG00000223972.4";
transcript_id "ENSG00000223972.4"; gene_type "pseudogene"; gene_status "KNOWN"; gene_name "DDX11L1";
transcript_type "pseudogene"; transcript_status "KNOWN"; transcript_name "DDX11L1"; level 2;
```

```
havana_gene "OTTHUMG00000000961.2";
chr1    HAVANA  transcript      11869   14409   .       +       .       gene_id "ENSG00000223972.4";
transcript_id "ENST00000456328.2"; gene_type "pseudogene"; gene_status "KNOWN"; gene_name "DDX11L1";
transcript_type "processed_transcript"; transcript_status "KNOWN"; transcript_name "DDX11L1-002";
level 2; havana_gene "OTTHUMG00000000961.2"; havana_transcript "OTTHUMT00000362751.1";
chr1    HAVANA  exon    11869   12227   .       +       .       gene_id "ENSG00000223972.4";
transcript_id "ENST00000456328.2"; gene_type "pseudogene"; gene_status "KNOWN"; gene_name "DDX11L1";
transcript_type "processed_transcript"; transcript_status "KNOWN"; transcript_name "DDX11L1-002";
level 2; havana_gene "OTTHUMG00000000961.2"; havana_transcript "OTTHUMT00000362751.1";
chr1    HAVANA  exon    12613   12721   .       +       .       gene_id "ENSG00000223972.4";
transcript_id "ENST00000456328.2"; gene_type "pseudogene"; gene_status "KNOWN"; gene_name "DDX11L1";
transcript_type "processed_transcript"; transcript_status "KNOWN"; transcript_name "DDX11L1-002";
level 2; havana_gene "OTTHUMG00000000961.2"; havana_transcript "OTTHUMT00000362751.1";
```

Each entry in a GTF file consists of 8 fields.

The first field contains the chromosome.

The second field contains information about the annotation (the source of the feature).

The third indicates the type of feature. The only absolutely required type of feature is `'exon'`; a GTF file can contain only exons and is a fully valid GTF file, but a lot more information than that can be added, for example, you may find the following additional entries: `'gene'`, `'transcript'`, `'CDS'`, `'start_codon'`, `'stop_codon'`, and others. Note that it is extremely important to distinguish between the features and not treat everything as exons when parsing a GTF file. The GTF files you will be working with will contain CDS features specifying open reading frames (ORF), which means you will not have to do ORF annotation on your own.

The fourth and fifth fields contain the coordinates for the feature (start and end).

The sixth field is left for a score value indicating the confidence in the feature; this will usually be a dot `'.'`, and there is no need for you to pay attention to it. The seventh field specifies the strand of the feature ('+' or '-').

The next field contains information about the phasing of the feature with respect to the ORF (if there is an ORF associated with it, of course; a value of 0 indicates that the 5' end of the feature begins with a whole codon, a value of 1 that the 5' end of the feature begins begins with a basepair and then a whole codon, and a value of 2 that the 5' end of the feature begins begins with 2 basepairs and then a whole codon. This information is present in annotation GTF files but is by no means an universal feature (as ORF annotation is not done in the process of generation of all GTF files) so I do not advise you to rely on it for keeping track of codon phasing even if it is present.

The last field contains a number of "attribute" subfields containing information about the gene and transcript that contain the feature. Note the way they are separated from each other. The only absolutely necessary attribute fields are the `gene_id` and `transcript_id`, everything else is optional (in this case, the gene and transcript names, and the type of gene and trancsript). Of course, the more information there is, the better for downstream users, but if you are to write general code for parsing GTF files, you can not rely on the presence of such information (unless you absolutely need it and you are certain it is present).

7 SNV and small indel discovery pipeline

Several tools and pipelines for carrying out SNV and small indel discovery pipeline exist, in fact even the 1000 Genomes Project [1] used multiple pipelines (largely because most of the existing tools were developed as part of the project). In the last two years, a pipeline consisting of mapping reads with `BWA` [9, 10] and variant discovery and filtering by the Genome Analysis Tool Kit (`GATK`) [4, 12] has emerged as the most widely used one and a de facto field standard. Running `GATK`, however, is more computationally intensive

than practical for the purposes of this exercise, so we will run a different pipeline consisting of mapping reads with `Bowtie2` [7, 8], and variant discovery using `samtools`.

7.1 Mapping reads with Bowtie2

We will run Bowtie2 as follows:

```
bowtie2-align -p 8 --end-to-end --very-sensitive -x $bowtie2-indexes/hg19-male
-q --no-unal reads.fastq reads.bowtie2.very-sensitive.SAM
```

Where `-x $bowtie2-indexes/hg19-male` refers to the human genome index (it is an FM Index, but do not worry what that means) that Bowtie2 will align against, `reads.fastq` are your reads, and `reads.bowtie2.very-sensitive.SAM` is the output SAM file. In this example we ask for 8 CPUs on which the program will run in parallel in order to speed up the process (that is, assuming you have 8 CPUs on your system) with the `-p 8` option, `-q` indicates that reads are in FASTQ format, `--no-unal` tells the program not to output unaligned reads, the `--end-to-end` option means end-to-end rather than local alignment and `--very-sensitive` specifies the alignment parameters. Refer to the Bowtie2 website `http://bowtie-bio.sourceforge.net/bowtie2/index.shtml` for more information if you are curious.

7.2 Conversion of the SAM file to a BAM file

Next, we convert the SAM file into a BAM file using the `view` function in `samtools`:

```
samtools view -bT $sequence/hg19.fa - | samtools sort reads.bowtie2.very-sensitive.SAM
-o reads.bowtie2.very-sensitive.bam
```

7.3 Sorting the BAM file

We then sort the alignments by chromosome and chromosome position using the `sort` function in `samtools`:

```
samtools sort reads.bowtie2.very-sensitive.bam reads.bowtie2.very-sensitive.sorted
```

Note that the sorted bam file prefix (i.e. with the `.bam` extension) is given rather than the actual output file name; `samtools` will automatically append the extension.

7.4 Running all of the above at once

SAM files are large and take up a lot of space, while all you are really going to need is the final sorted bam file. Fortunately, you can run all of the above commands at once without writing anything but the final sorted bam file onto disk, by using the UNIX's standard input and output streams (note all the parameters that have been replaced by `'-'` symbols), as follows:

```
bowtie2-align -p 8 --end-to-end --very-sensitive -x $bowtie2-indexes/hg19-male
-q --no-unal reads.fastq | samtools view -bT $sequence/hg19.fa - | samtools sort -
reads.bowtie2.very-sensitive.sorted
```

In practice, fastq files are also very bulky and take up a lot of space so they are sorted as compressed `.gz` or `.bz2` files. You will also be given compressed files in this class, but there is no need to uncompress them. You can instead run the pipeline as follows:

```
gunzip -c reads.fastq.gz | bowtie2-align -p 8 --end-to-end --very-sensitive
-x $bowtie2-indexes/hg19-male -q --no-unal - | samtools view -bT $sequence/hg19.fa -
| samtools sort - reads.bowtie2.very-sensitive.sorted
```

or:

```
bzip2 -cd reads.fastq.bz2 | bowtie2-align -p 8 --end-to-end --very-sensitive
-x $bowtie2-indexes/hg19-male -q --no-unal - | samtools view -bT $sequence/hg19.fa -
| samtools sort - reads.bowtie2.very-sensitive.sorted
```

Note the addition of the `-c` parameter to `gunzip` and `bzip2` which sends the output of those programs to the standard output stream.

7.5 Removing duplicate reads

It is a standard practice when working with genome resequencing data to remove apparent duplicate reads, i.e. reads (or fragments, which is a pair of reads, if you are working with paired-end data) that map to exactly the same position in the genome. This is done in order to eliminate PCR artifacts from affecting results. We will "dedup" our data as follows:

```
samtools rmdup -s reads.bowtie2.very-sensitive.sorted.bam
                  reads.bowtie2.very-sensitive.sorted.ddup.bam
```

7.6 Indexing the BAM file

BAM files need to be indexed for fast access by subsequent applications. This is done as follows:

```
samtools index reads.bowtie2.very-sensitive.sorted.ddup.bam
```

7.7 Calling variants

We will call variants using the `mpileup` program in `samtools` and the `bcftools` program that is also part of the `samtools` package. This can be done at two separate steps, but can be also done in one step similar to the way we did the mapping, conversion from SAM to BAM and sorting the BAM file above:

```
samtools mpileup -uf $sequence/hg19.fa reads.bowtie2.very-sensitive.sorted.ddup.bam |
bcftools view -vcg - > reads.bowtie2.very-sensitive.sorted.ddup.vcf
```

References

[1] 1000 Genomes Project Consortium, Abecasis GR, Auton A, Brooks LD, DePristo MA, Durbin RM, Handsaker RE, Kang HM, Marth GT, McVean GA. 2012. An integrated map of genetic variation from 1,092 human genomes. *Nature* **491**(7422):56-65.

[2] Cock PJ, Fields CJ, Goto N, Heuer ML, Rice PM. 2010. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res* **38**(6):1767-1771.

[3] Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. 2011. The variant call format and VCFtools. *Bioinformatics* **27**(15):2156-2158.

[4] DePristo MA, Banks E, Poplin R, Garimella KV, Maguire JR, Hartl C, Philippakis AA, del Angel G, Rivas MA, Hanna M, McKenna A, Fennell TJ, Kernytsky AM, Sivachenko AY, Cibulskis K, Gabriel SB, Altshuler D, Daly MJ. 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat Genet* **43**(5):491-498.

[5] Keane TM, Goodstadt L, Danecek P, White MA, Wong K, Yalcin B, Heger A, Agam A, Slater G, Goodson M, Furlotte NA, Eskin E, Nellker C, Whitley H, Cleak J, Janowitz D, Hernandez-Pliego P, Edwards A, Belgard TG, Oliver PL, McIntyre RE, Bhomra A, Nicod J, Gan X, Yuan W, van der Weyden L, Steward CA, Bala S, Stalker J, Mott R, Durbin R, Jackson IJ, Czechanski A, Guerra-Assuno JA, Donahue LR, Reinholdt LG, Payseur BA, Ponting CP, Birney E, Flint J, Adams DJ. 2011. Mouse genomic variation and its effect on phenotypes and gene regulation. *Nature* **477**(7364):289-294.

[6] Kent WJ, Zweig AS, Barber G, Hinrichs AS, Karolchik D. 2010. BigWig and BigBed: enabling browsing of large distributed datasets. *Bioinformatics* **26**(17):2204-2207.

[7] Langmead B, Salzberg SL. 2012. Fast gapped-read alignment with Bowtie 2. LiDurbin2009 **9**(4):357-359.

[8] Langmead B, Trapnell C, Pop M, Salzberg SL. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol* **10**(3):R25.

[9] Li H, Durbin R. 2009. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* **25**(14):1754-1760.

[10] Li H, Durbin R. 2010. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics* **26**(5):589-595.

[11] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R; 1000 Genome Project Data Processing Subgroup. 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**(16):2078-2079.

[12] McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, Garimella K, Altshuler D, Gabriel S, Daly M, DePristo MA. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res* **20**(9):1297-1303.