Aligning reads & processing SAM files

Exercise for creating and processing short read alignments.

- **Contact:** alexander.kanitz@unibas.ch

Prerequisites

Operating system

For solving these exercises, you need to work in Bash on a Unix-type terminal, so if you are behind either a Linux distribution, a Mac or a BSD system etc., you should be good to go. If instead you only have access to a Windows 10/11 computer, you should be able to make use of the support the system offers for Linux/Ubuntu through their cooperation with Canonical. For older Windows versions or even better compatibility, you can also always run a virtual machine with any Linux distribution you like.

See here for some pointers for either approach.

Finally, if you have experience with Docker, one other possible option is to use a Docker image that contains all required software. For example, starting from a Linux image (e.g., latest Ubuntu), which provides Linux/GNU/Bash out of the box, you could install STAR and/or any other required Linux software by writing an appropriate Dockerfile and then building that image. You can also search online for available images that already contain these tools (e.g., this looks like one you could use for running STAR; see below). If you decide to go this route, be aware that Docker on Windows still runs in a VM and support is not always stable.

> **Note:** If you are planning to do more bioinformatics work in the future, it is definitely a good idea to have a stable Linux system at hand at all times. In this case, you might want to consider installing a Linux distribution side-by-side with your Windows OS (search for "Linux Windows dual boot" or similar).

Software

In the last exercise, you were asked to write a simple, naive short read aligner from scratch. While this is a good exercise, it is not suitable for actual analysis, because your code won't be optimized to handle the amounts of data that next-generation sequencing typically yields. Since the dawn of high-throughput sequencing techniques in the mid 2000's, a lot of effort has been put into designing and implementing very efficient methods at mapping short reads to reference sequences. For this exercise, we will use a popular option called STAR. Among many other features, STAR supports spliced alignments and optionally accepts gene annotations in GTF format next to a genome reference to increase the fidelity of mapping reads that cover splice junctions (if not provided, STAR, by default, will try to infer splice junctions from the genome reference and the reads). Please either install STAR (and any other third-party software you need) via the Conda package manager *OR* use Docker containers as suggested above.

Exercises

Part 1: Map reads

Exercise 1.1: Create index (2 points)

Follow STAR's manual to create an index of the provided genome `FASTA` file and `GTF` gene annotations. Note that to allow the mapping to be done on a laptop, only chromosome 19 and the corresponding gene annotations are provided. In a typical setting, indexing and mapping would be done on all chromosomes and an unfiltered set of annotations. Note also that files are provided in a compressed form (`GZIP`) for easy sharing and may need to be uncompressed before use (check the instructions whether passing `GZIP`ped files directly is accepted).

Required files:

- Genome: `Mus_musculus.GRCm38.dna_rm.chr19.fa.gz`
- Gene annotations: `Mus_musculus.GRCm38.88.chr19.gtf.gz`

Exercise 1.2: Align reads (2 points)

Follow STAR's manual to align reads to the reference, using the index you have created in exercise 1.1. Note that we are dealing here with (part of) a paired-end sequencing library, so you will need to provide both read library files for this step.

Required files:

- Reads mate 1: `reads.mate_1.fq.gz`
- Reads mate 2: `reads.mate_2.fq.gz`

Part 2: Process alignments

Exercise 2.1: Count reads (4 points)

Use Bash/GNU commands (see file `bash_basics.sh`) to find out from the STAR output (`SAM` file):

- How many alignments were reported? 108
- How many reads were uniquely mapped? 4
  Hint: check for the `NH` (number of hits) SAM tag
- How many reads were mapped to multiple loci? 26
- How many reads could not be mapped? 0

Compare the sum of uniquely mapped, multi-mapped and unmapped reads to the total number of reads in the FASTQ input files. Do the numbers match?

> **Note:** See the SAM specification for more info on SAM files.

Exercise 2.2: Run custom functions on STAR results (2 points)

`FASTQ` (for reads), `FASTA` (for reference sequences and reads if sequencing quality scores are disregarded or discarded), `GTF`/`GFF` (for gene annotations/features), `SAM` (for alignments; with the corresponding binary/compressed versions `BAM`, and, more recently, `CRAM`), `BED` (generic tabular format for representing genomic ranges) are the main file types that are used in the analysis of RNA-Seq data. Relevant tools in the field will nowadays almost always require input files and report their own outputs in any of these formats. However, not every tool accepting a `FASTA` file as input will also accept a `FASTQ` file, although it is trivial to convert `FASTQ` to `FASTA` in a non-lossy way. In addition, both for legacy and new tools, custom formats are still being used, occasionally, to represent specialized information. Therefore, writing and applying parsers and converters to convert outputs of one tool such that they can be used as inputs to another is a somewhat menial, but common task that bioinformaticians are often faced with.

To practice this and connect to the work you have done in the previous exercise:

- Write code that converts a `SAM` file to a `FASTA` file and apply it on the output of exercise 2.1. If you didn't manage to solve exercise 2.1, write code that converts `FASTQ` to `FASTA` instead.
- Convert your files to `FASTA` and then apply your functions from the previous session to the output.

Part 3: Differential gene expression analysis (optional)

One of the most common experiment types making use of RNA-Seq is a differential gene expression analysis where gene expression levels across different conditions (e.g., healthy vs disease) or compared to each other. This type of experiment is often the basis for discovering genes that are relevant to a given physiological or disease process. Following this relatively open-ended discovery phase of a study, a list of genes with a strikingly different expression pattern is often the basis for further, more detailed mechanistic studies. While performing an entire differential gene expression analysis is beyond the scope of this limited seminar, you are invited to dig further into this topic in an open-ended way, by reading on the following topics and either writing your own code or apply available tools/packages for these tasks:

- Create a table of counts for each gene (rows) and sample (columns)
  Hint: You can easily implement this yourself, but you can also make use of the HTSeq package for this task (can also be used to report counts for other features)
- Analyze gene expression across different conditions based on such a count table
  Hint: You can do this, e.g., with the edgeR package, written in R.

If you are doing any work in this regard, feel free to send it to me for some feedback.

> **Note:** As an alternative to alignment- and count-based approaches to a differential gene expression, nowadays probabilistic methods are increasingly being used for similar purposes. These have several advantages, such as (typically) requiring fewer steps and less compute resources. Importantly, they provide abundances for individual transcripts and thus enable analyses, such as differential transcript expression and isoform usage analysis, which the count-based methods do not readily allow. A downside for these methods is that they are not easy to understand in detail, making results harder to interpret and potentially more sensitive to biases. The two main tools for alignment-free estimation of transcript abundances are Salmon and kallisto.