PROTOCOL

BIOMEDICAL DATA ANALYSIS

Name: Vaidehi Ponkshe

Matriculation no: 22201071

Course: M.Sc. Life Science Informatics

Contents

** denotes use of ChatGpt on some level, not completely

INTRODUCTION:

BIOMEDICAL DATA ANALYSIS

Biomedical data analysis involves the analysis of large-scale biomedical datasets to understand how living systems function. The goal of biomedical data analysis is to advance our understanding of human health and disease and to inform the development of new diagnostic tools, treatments, and preventative measures. Performing biomedical data analysis effectively requires a combination of domain expertise, technical skills, and attention to ethical considerations, as well as a commitment to reproducibility and transparency in the research process. This may involve the use of various statistical, computational, and machine learning methods to analyze data from sources such as genomic sequencing, medical imaging, electronic health records, and clinical trials.

REPRODUCIBLE RESEARCH

Reproducibility refers to the ability to reproduce the results of a study using the same data and methods as described in the original study. Reproducibility involves having access to the original data, code, and documentation, as well as being able to recreate the computational environment used in the original study. Reproducible research is the system of documenting and publishing results of an impact evaluation. At the very least, reproducibility allows other researchers to analyze the same data to get the same results as the original study, which strengthens the conclusions of the original study.

REPLICATION AND REPRODUCIBILITY

Replication refers to the independent repetition of an experiment or study to confirm or validate its results. Replication is essential in science to establish the validity and reliability of research findings. Reproducibility, on the other hand, refers to the ability to reproduce the results of a study using the same data and methods as described in the original study. Reproducibility involves having access to the original data, code, and documentation, as well as being able to recreate the computational environment used in the original study. In summary, replication is the repetition of an experiment to validate its results, while reproducibility is the ability to recreate the results using the same data and methods. Both are important for establishing the credibility of scientific research.

TECHNICAL CHALLENGES

1. A critical barrier to reproducibility in many cases is that the computer code is no longer available.
2. Specifying the exact software, libraries, and their versions used in the research is important to ensure that others can recreate the results.
3. Keeping track of changes to the code and data throughout the research process is necessary to enable reproducibility.
4. Collaborating on research projects with others can lead to a loss of reproducibility if the project is not properly documented and managed.
5. With the increasing size and complexity of research data and models, it can be challenging to ensure reproducibility at scale.
6. The biggest barrier to reproducible research is the lack of a deeply ingrained culture that simply requires reproducibility for all scientific claim

NCBI SRA

NCBI SRA (Sequence Read Archive) is a publicly accessible database for high-throughput sequencing data generated by the scientific community. It is a resource for researchers who want to access raw sequencing data for further analysis and to ensure the reproducibility of results. Researchers can use the NCBI SRA to access data from a variety of organisms, including humans, plants, animals, and microbes. The data in the NCBI SRA can be used for a variety of purposes, including genome assembly, gene expression analysis, variant discovery, and more.

CONDA/BIOCONDA

Conda is an open-source package management system and environment management system that enables users to install and manage packages and dependencies for various programming languages. Biodonda is a specific distribution of conda specifically designed for bioinformatics applications. The use of conda/biodonda can ensure reproducibility of results by enabling users to recreate the same computational environment used in a study.

JUPYTERLAB

Jupyterlab is an open-source web-based interactive development environment (IDE) for working with Jupyter notebooks, which are documents that contain live code, equations, visualizations, and narrative text. Jupyterlab is often used in reproducible research to share and document code, results, and narratives in a single document, making it easier for others to understand and replicate the study.

TECHNICAL SETUP

INSTALLING CONDA

Steps for installing conda on linux:

1. Open the terminal window on Linux machine (remote or local)
2. Download the latest version of miniconda using the following command **wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh**
3. Install the downloaded miniconda file with the following command **bash Miniconda3-latest-Linux-x86_64.sh**
4. This will prompt on-screen installation instructions. Simply follow them to complete the installation process. Don't forget to initialize the conda in the installation process.

CONDA CHANNELS

Conda channels are important for package management, as they allow users to control where packages are installed from and what version of a package is installed. For example, you can use a specific channel to install a specific version of a package, or to install packages that are not available in the default channels.

By default, Anaconda Distribution includes a default channel named "Anaconda" that contains the most popular packages, as well as several other channels. However, users can also add custom channels to their conda configuration, including channels hosted by other organizations, personal channels, or channels for specific projects or applications.

By using channels, users can manage the packages and dependencies needed for their projects, ensuring that their projects are reproducible and that the correct version of a package is used for a specific project or application.

We need to add two channels *bioconda* and *conda-forge* with the following commands.
$ conda config --add channels **conda-forge**
$ conda config --add channels **bioconda**

CONDA ENVIRONMENTS

Conda environments are isolated spaces where you can install packages and their dependencies without interfering with other projects or your system. Each environment has its own set of packages and dependencies, which can be different from other environments. This allows you to create a specific environment for each project or task, with the exact packages and dependencies required for that project or task. You can also easily switch between environments or share them with others.

In order to proceed with system setup tasks, we need to a new environment with the name" bio" with a Python version of 3.9. For this we use the following command:

$ conda create -n bio python=3.9

To activate the newly made bio environment, the following command is used:

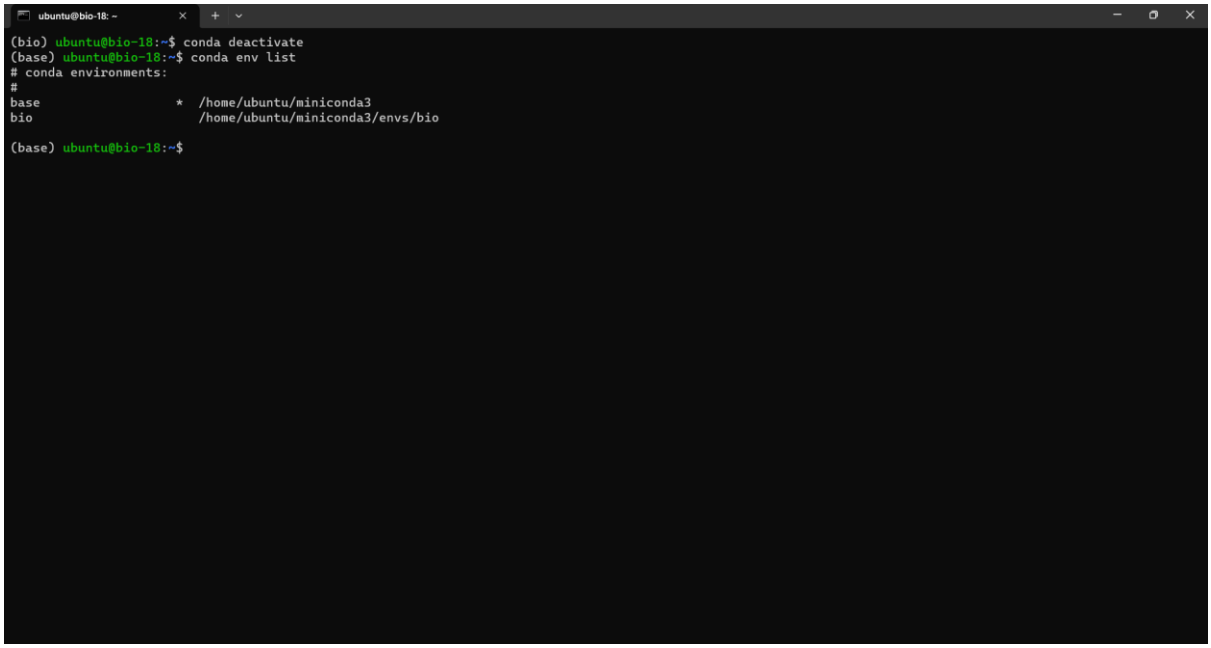
$ conda activate bio

To deactivate this environment and go back to the base envirnoment, the following command is used:

$ conda deactivate

To check all the availabe environment on a system the following commands can be used This list should contain the newly made "bio" environment if the installation was succesfull.

$conda env list



LINUX SHELL PATH VARIABLE

The PATH environment variable in Linux is used to define the directories in which the shell looks for executable files. The directories are separated by colons (:) in the PATH variable.
In a conda environment, the PATH variable is updated to include the bin directories of the conda environment. For example, when you activate the base conda environment, the PATH variable may look like this:

$ echo $PATH

/home/user/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

After creating a new conda environment, the PATH variable is updated to include the bin directory of the new environment. For example, after creating a new environment named "myenv", the PATH variable may look like this:

$ conda create --name myenv

$ conda activate myenv

(myenv) $ echo $PATH

/home/user/anaconda3/envs/myenv/bin:/home/user/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

As you can see, the PATH variable now includes the bin directory of the new environment, **"/home/user/anaconda3/envs/myenv/bin"**, before the bin directory of the base conda environment, **"/home/user/anaconda3/bin"**.

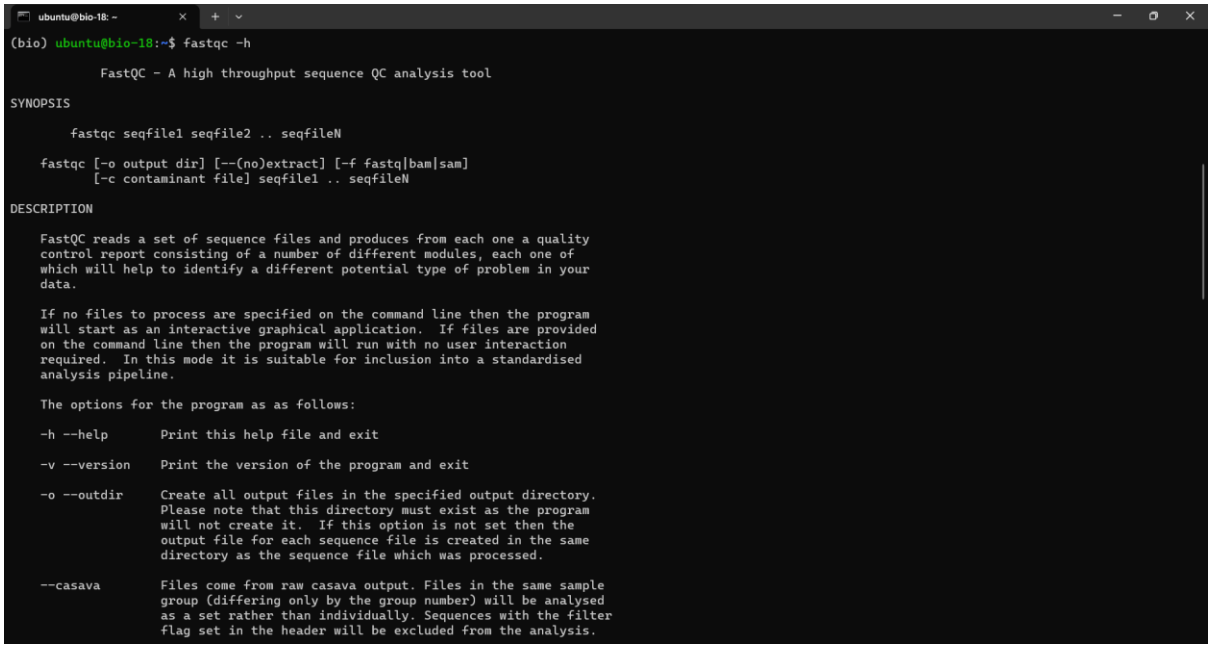
This change in the PATH variable has consequences for package management. When you run an executable from the command line, the shell will first search the directories in the PATH variable, in the order they are listed. By having the bin directory of the new environment listed first, you ensure that executables installed in the new environment are used, instead of executables from the base conda environment or other system directories. In this way, you can use different conda environments to manage different sets of packages and dependencies for different projects, ensuring that each project uses the correct version of a package, and that packages do not interfere with each other.

INSTALLING FASTQC

FastQC is a quality control tool for high throughput sequence data. To install fastqc, we first need to activate **bio** environment and then use the following command:

$ conda install -c bio fastqc

$ fastqc -h



INSTALLING JUPYTERLAB

JupyterLab is built on top of the Jupyter Notebook server and can be run on a local machine or remote server. It is designed to be extensible and customizable through third-party extensions and plugins.

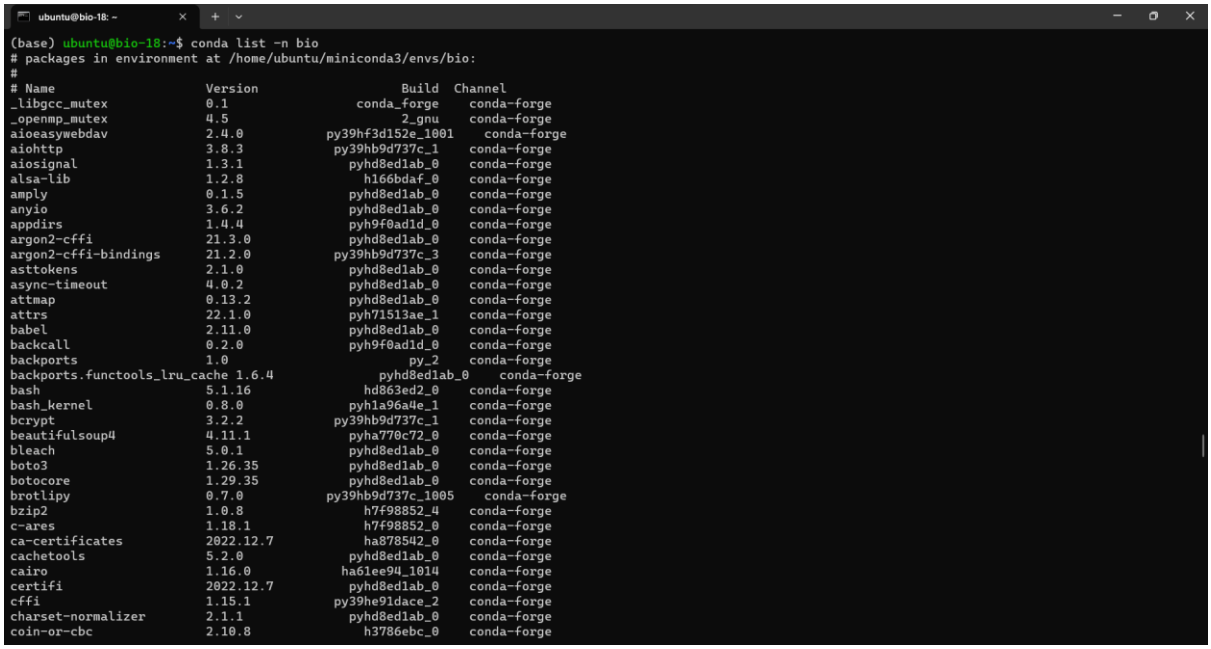
$ conda install -c bio jupyterlab

This step is followed by installing jupyter kernel for bash with the following command.

$ conda install -c bio bash_kernel

ALL INSTALLED PROGRAMS OF BIO ENVIRONMENT

$ conda list -n bio



Connect Jupyter and R

- Before we start Jupyter, we need to call one line that will register the current installation of R so that it can be used from Jupyter.
- Make sure that you are in your "bio" environment. If you are not in the "bio"
- environment, activate the "bio" environment first.
- Use the following command to be able to use R from a Jupyter notebook:
  $ R -e "IRkernel::installspec()"

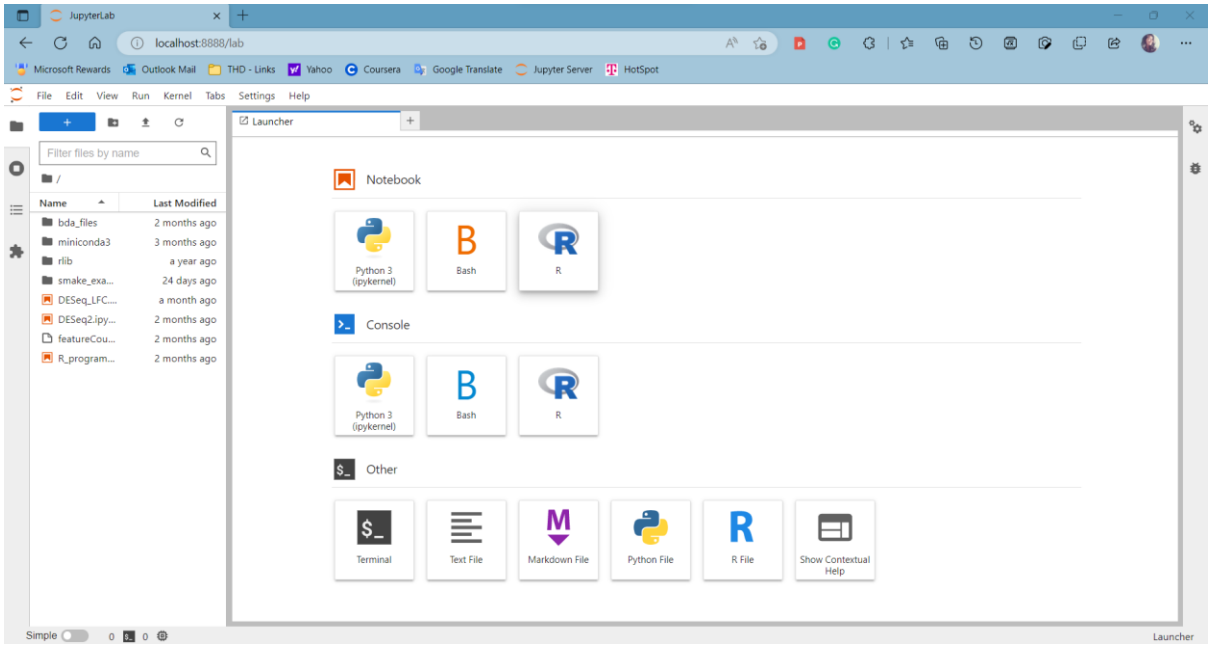
Start a Jupyter Server

- Activate your "bio" environment in case it is not already active. First, we configure the Jupyter server.
  $ jupyter server --generate-config
  $ jupyter server password
- The terminal will ask you for a password. Type in a password and remember it. Next, the program asks to confirm the password.
- Start a server by using the following command:
  $ jupyter lab --no-browser --ip "*"
- You will see lots of print outs from the starting server. You also won't see the prompt again, because the server is still active.

Connect to a Remote Jupyter Server

- To be able to connect to the remote Jupyter server, we need to redirect requests to the remote machine which is known as port forwarding.
- Forwarding can be established using a slightly different **ssh** call.
- Open a new (Windows) terminal to be able to establish another ssh connection. Instead of your normal ssh connection command you will use an additional **-L option**.
  $ ssh -i /path/to/your/keyfile -L 8888:localhost:8888 ubuntu@xxx.xxx.xxx.xxx

- Any request to your local port 8888 is forwarded via a secure connection to **localhost:8888** at the specified remote machine.
- Use the command above, but replace the path to your key as well as your IP address. Press enter.
- You should see your terminal as you are used to. Open a browser on your computer and type in the address localhost:8888 and press enter.
- Your browser should connect to the Jupyter server and display Jupyter lab like this:



- Stop the Jupyter server by pressing **ctrl + c** (if you press the key combination twice, Jupyter will shut down immediately).

TMUX

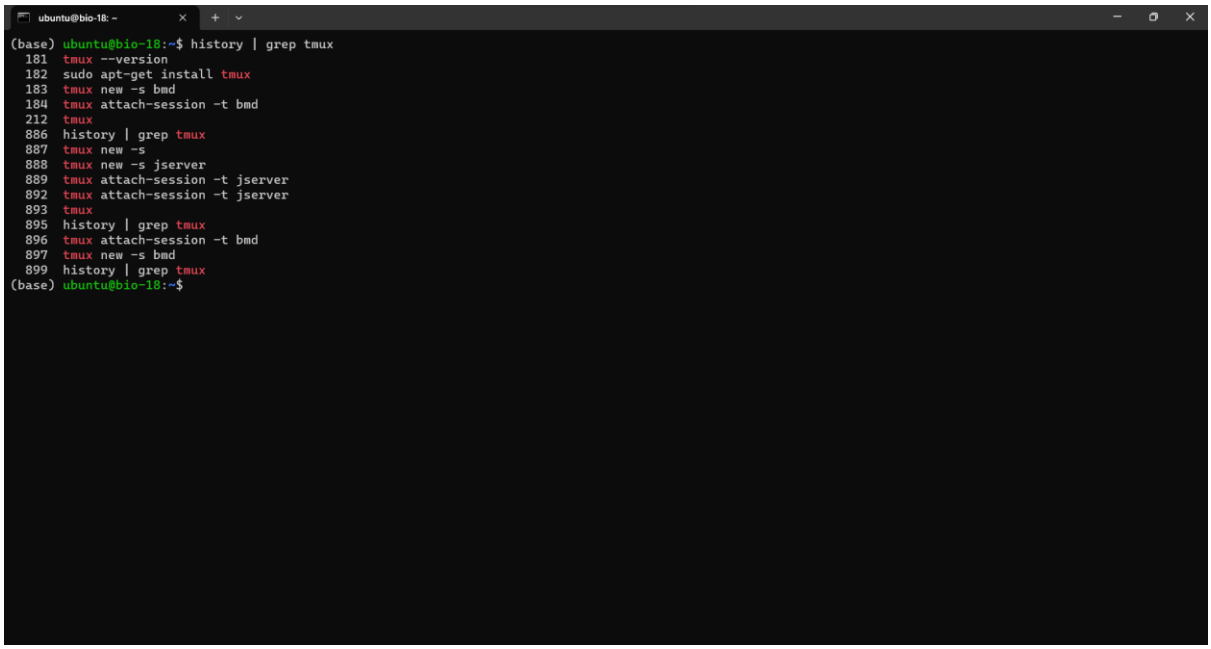
Tmux is a terminal multiplexer. It allows multiple terminal sessions to be attached to a single terminal window and be managed from a single terminal session. This allows for easy organization, management and persistence of terminal sessions. With tmux, you can switch between multiple terminal windows, start and stop processes, split windows into panes, and more, all within a single terminal session.

CREATING NEW SESSION - $ tmux new -s <session-name>

DETACHING FROM THE SESSION - Ctrl-b followed by d.

REATACHING TO A SESSION - $ tmux attach-session -t <session-name>

Where <session-name> is the name of the tmux session you want to reattach to.



CREATING MULTIPLE PANES AND SWTICHING THEM

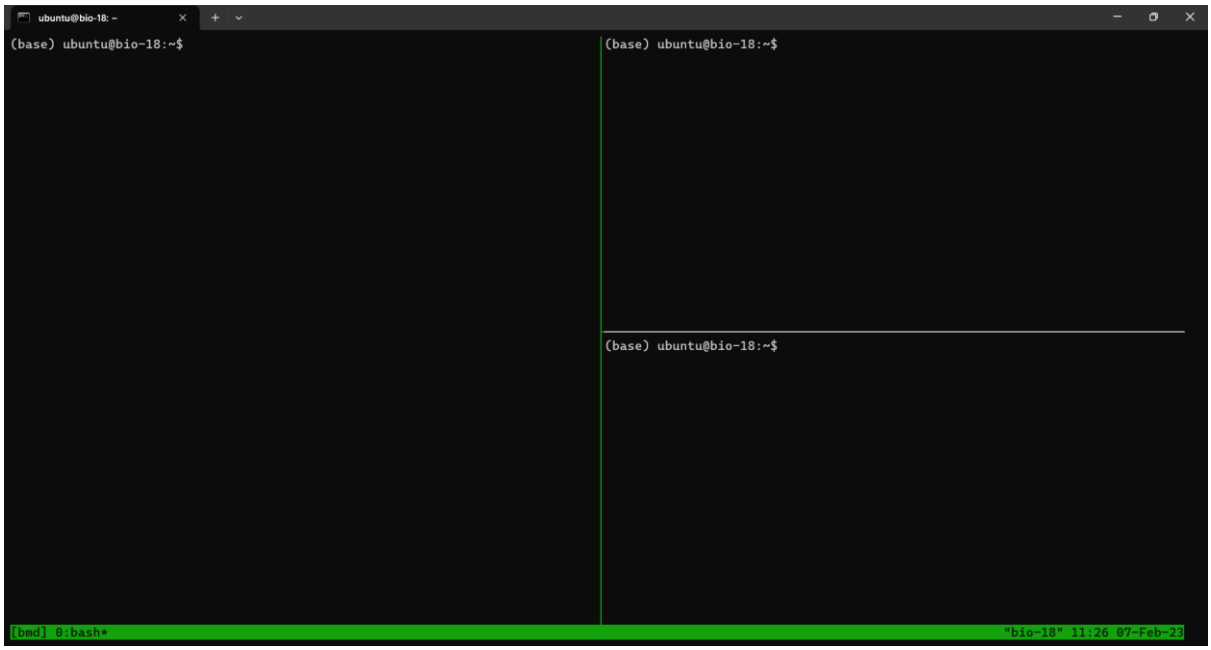
To create multiple panes within a tmux session, you can use the following key combinations:

- **Ctrl-b** followed by **%** to split the window vertically into two panes
- **Ctrl-b** followed by **"** to split the window horizontally into two panes

To switch between panes, you can use the following key combinations:

1. **Ctrl-b** followed by the **Up Arrow**, **Down Arrow**, **Left Arrow**, or **Right Arrow** key to switch to the pane in the corresponding direction.
2. **Ctrl-b** followed by **o** to switch to the next pane in the clockwise direction.
3. **Ctrl-b** followed by **{**or**}** to switch to the next or previous pane, respectively, in the current window.

Note: **Ctrl-b** is the default tmux key binding prefix, but it can be changed to something else in the tmux configuration file.



SESSION, WINDOW, PANE IN TMUX

In tmux, a **session** is a collection of terminal windows and their associated processes. A tmux session can persist even after the terminal window that started it is closed, allowing you to pick up right where you left off when you reattach to the session.

A **window** in tmux is a single terminal window that displays the output of one or more processes. Each window has its own shell and can contain multiple panes.

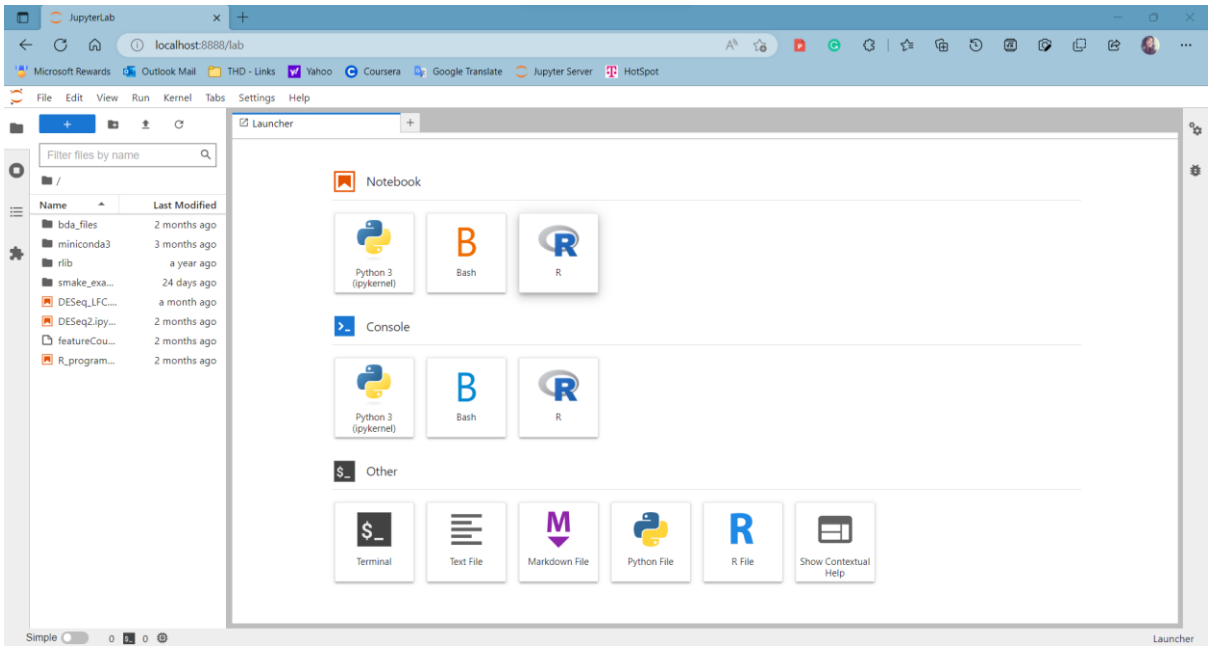
A **pane** in tmux is a rectangular area within a window that contains a shell and displays the output of a single process. Panes can be split into multiple panes, allowing you to see the output of multiple processes in the same window. This allows you to organize your terminal sessions in a flexible and efficient manner.

STARTING JUPYTER IN TMUX SESSION

$ tmux new -s jserver

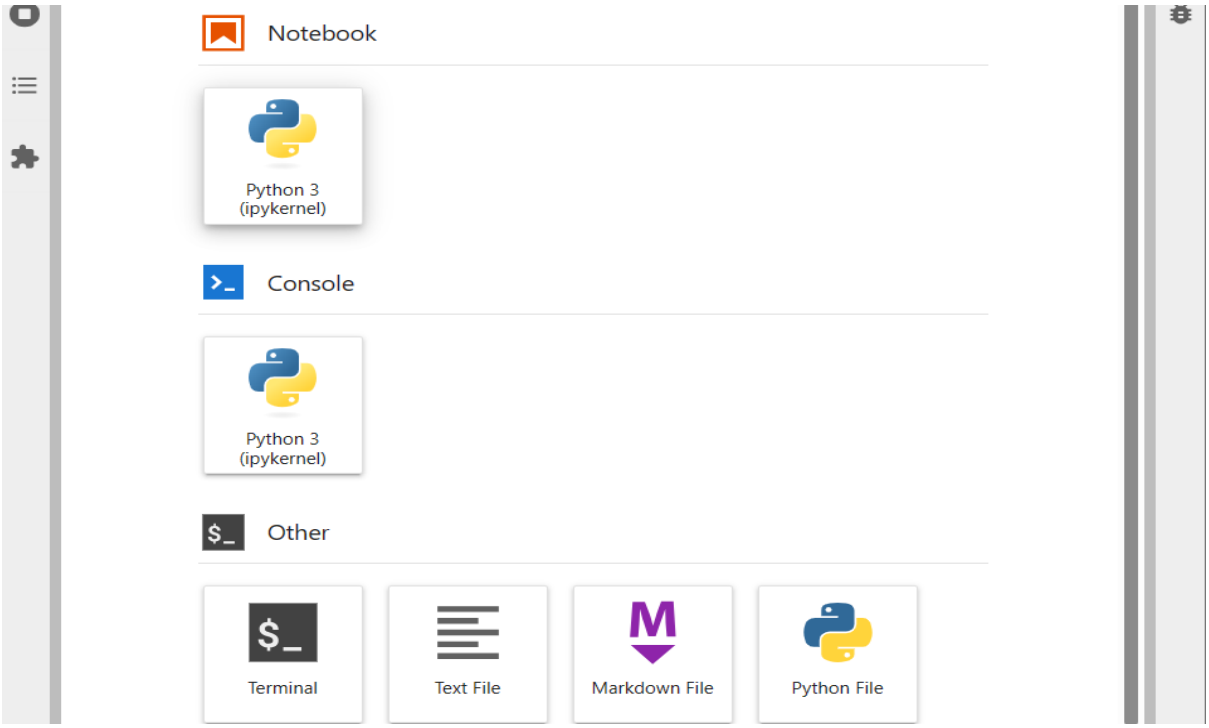
$ conda activate bio

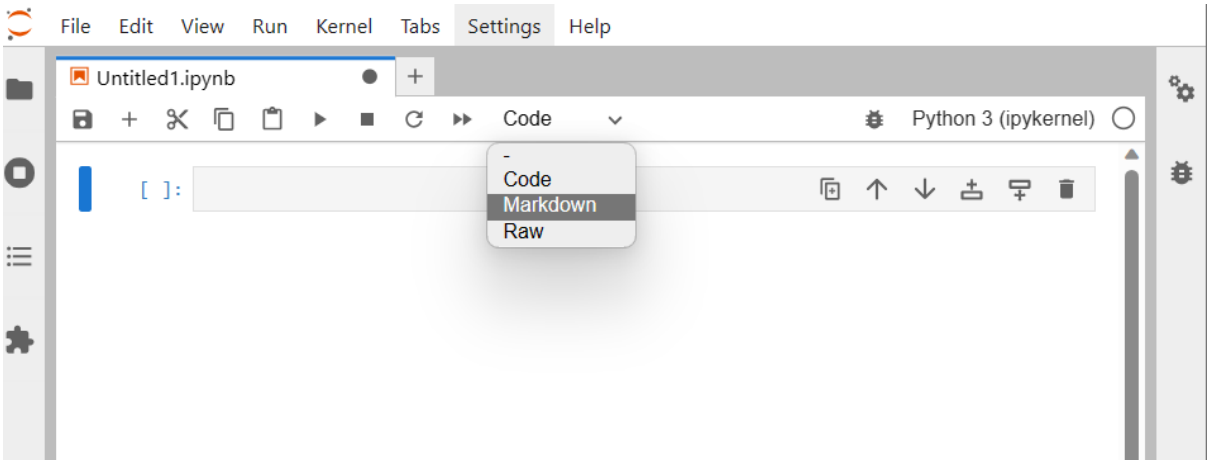
$ ssh -i C:\Users\hp\Downloads\key_vm_12.txt -L 8888:localhost:8888 ubuntu@10.20.102.54



JUPYTER NOTEBOOK/ MARKDOWN

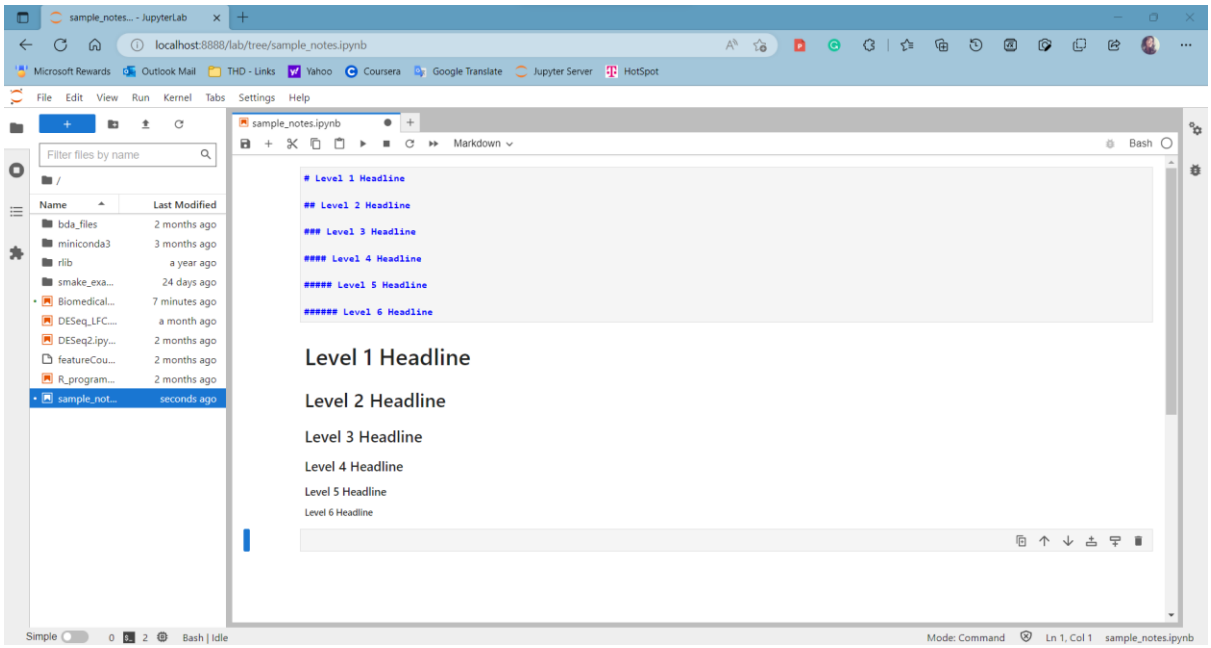
Jupyter supports Markdown, a lightweight and popular Markup language that you can use to add formatting elements to plaintext text documents. It is often converted into the corresponding HTML which by the Markdown processor which allows it to be easily shared between different devices and people

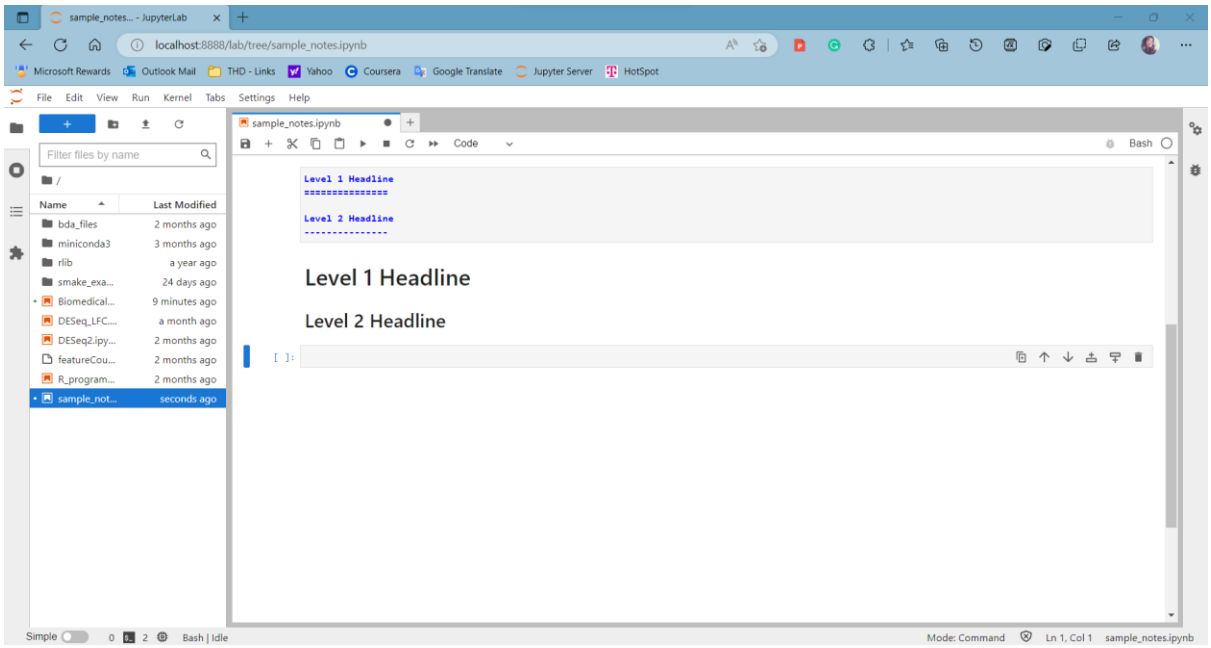




HEADINGS

In markdown, headlines can be created by using either # symbols or by underlining the text with = or - symbols. The size of the headings can be adjusted by using a different count of '#'. Higher the count, lower the heading size is general. There are six Headings with the largest heading only using one hash symbol and the smallest titles using six hash symbols.

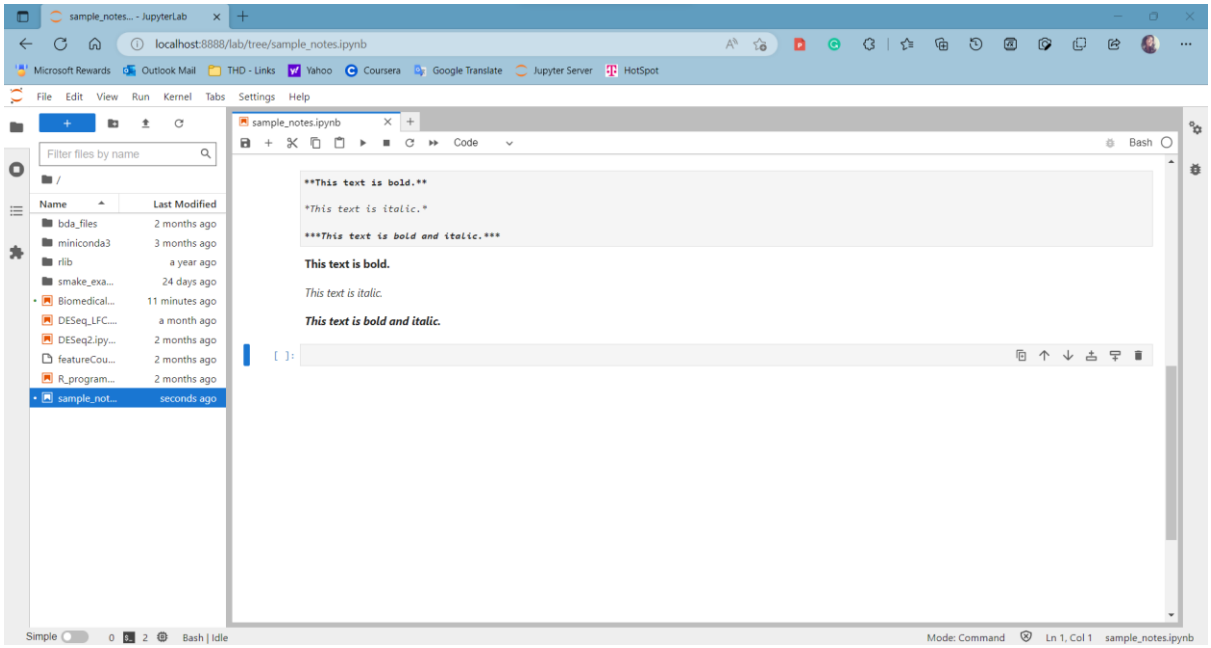




BOLD/ITALIC

Text can be formatted as bold, italic, or bold and italic using specific characters. The following symbols are used to format text:

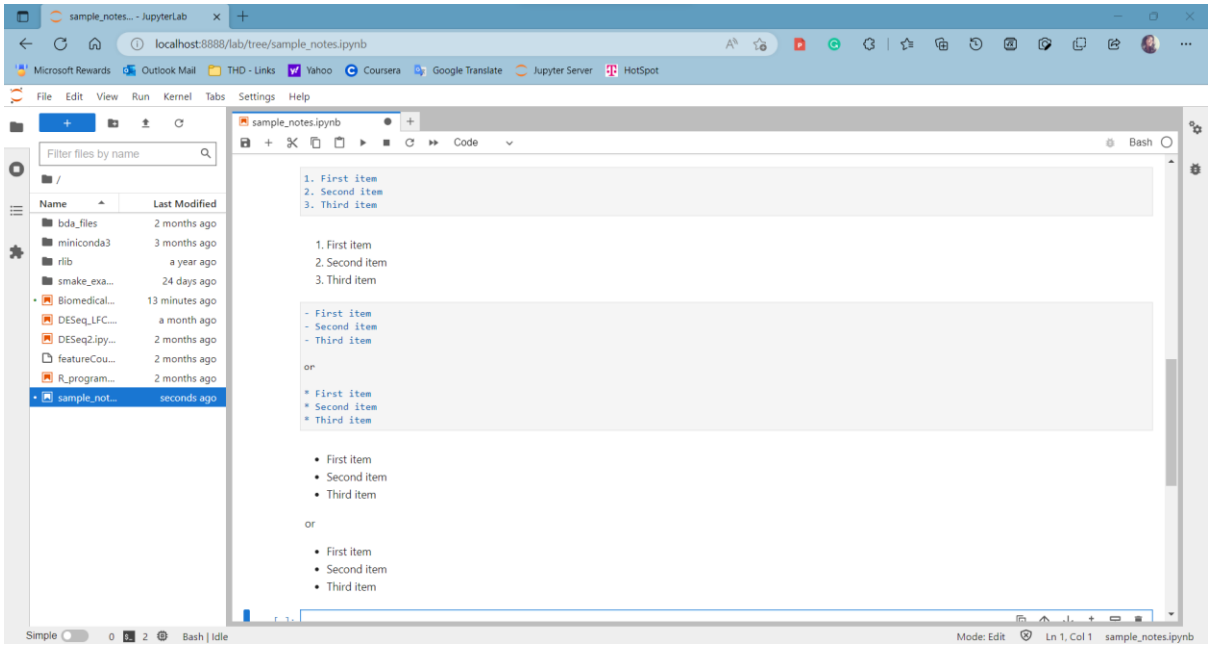
1. Bold: ****text**** or **__text__**
2. Italic: ***text*** or **_text_**
3. Bold and italic: *****text***** or **___text___**



NUMBERED/UNNUMBERED LISTS

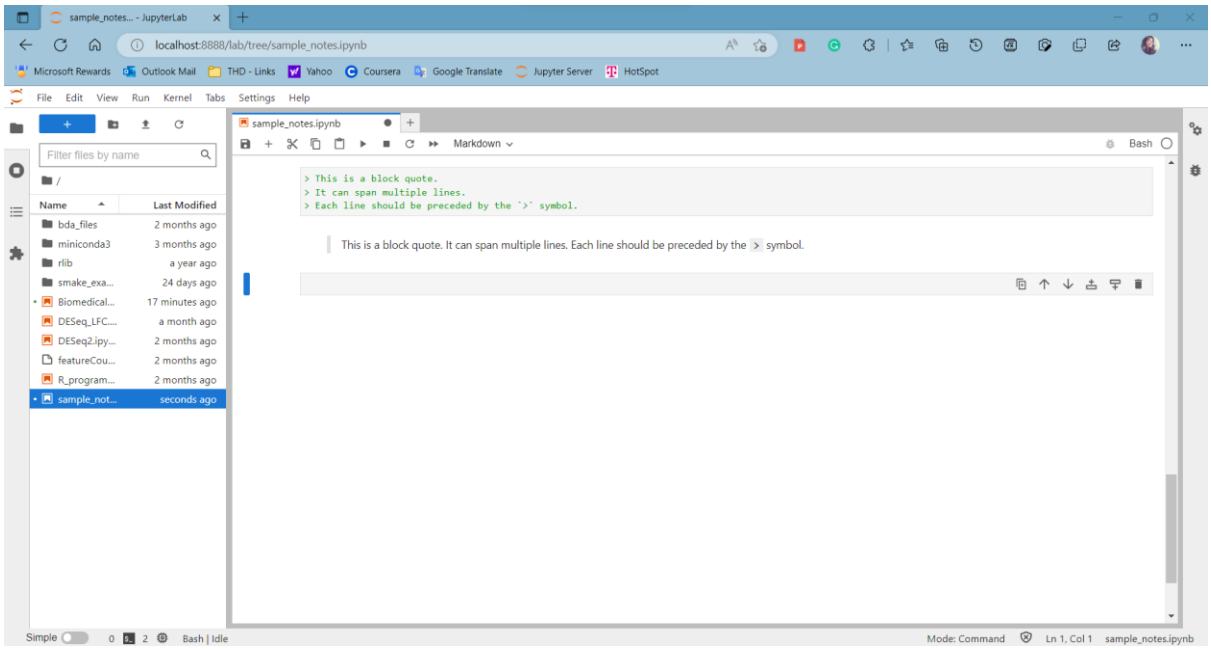
Numbered and unnumbered lists can be created using either numbers and periods, or hyphens and asterisks in markdown.

- To create a numbered list, use numbers followed by periods.
- To create an unnumbered list, use hyphens or asterisks.



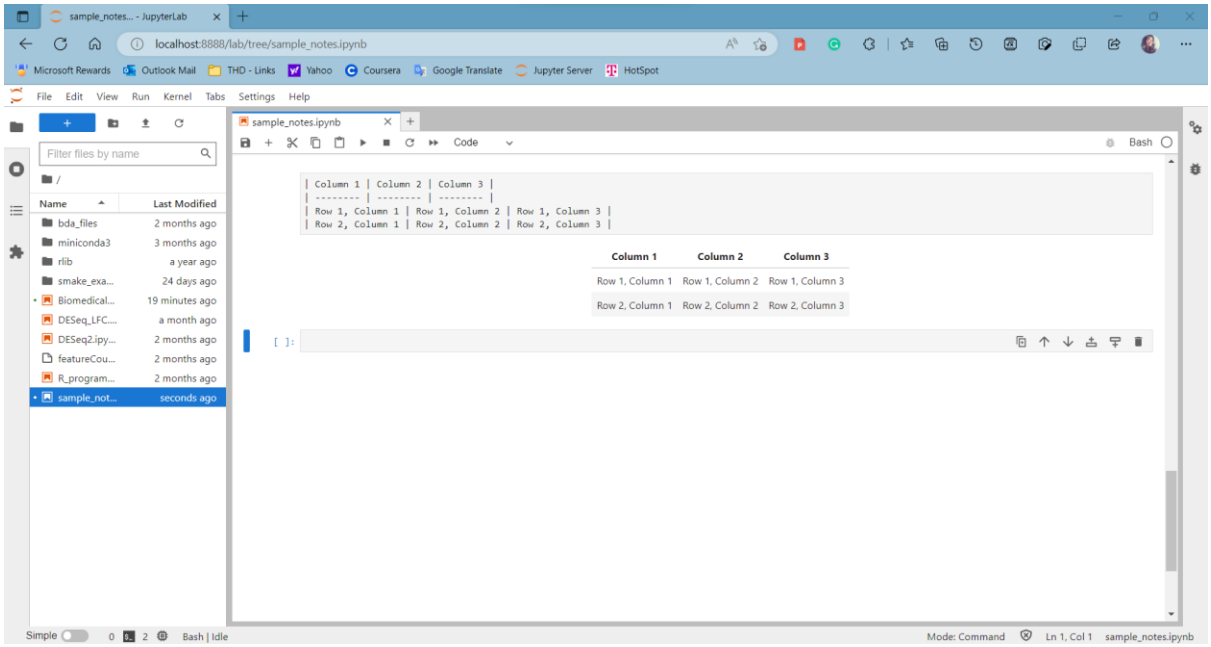
BLOCK QUOTE

For creating block quote, the text on a separate/new line should starts with greater than symbol '>' followed by space and the result would look like something like below.



ADDING TABLES

To add a table, use three or more hyphens (---) to create each column’s header, and use pipes (|) to separate each column. For compatibility, you should also add a pipe on either end of the row. Here is an example table created:



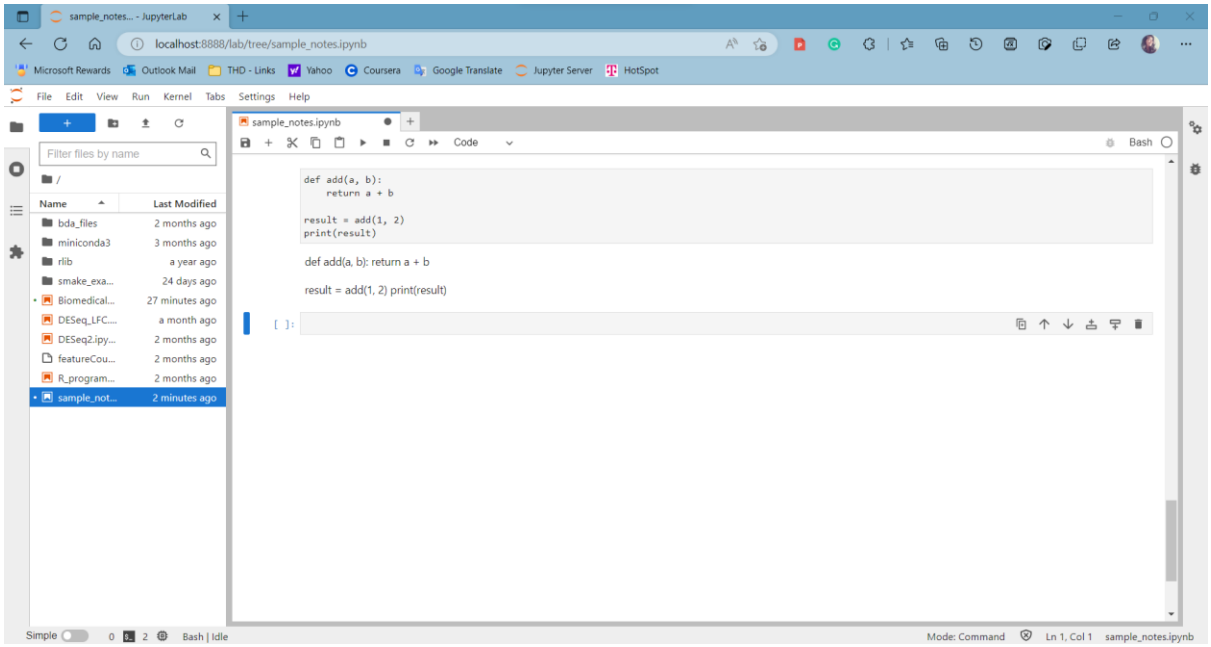
FOOTNOTES

Jupyterlab does not natively support footnotes, so they will not be rendered in the final output of a markdown cell in Jupyterlab. However, some Jupyterlab extensions, such as the Jupyterlab LaTeX extension, may allow you to add footnotes.

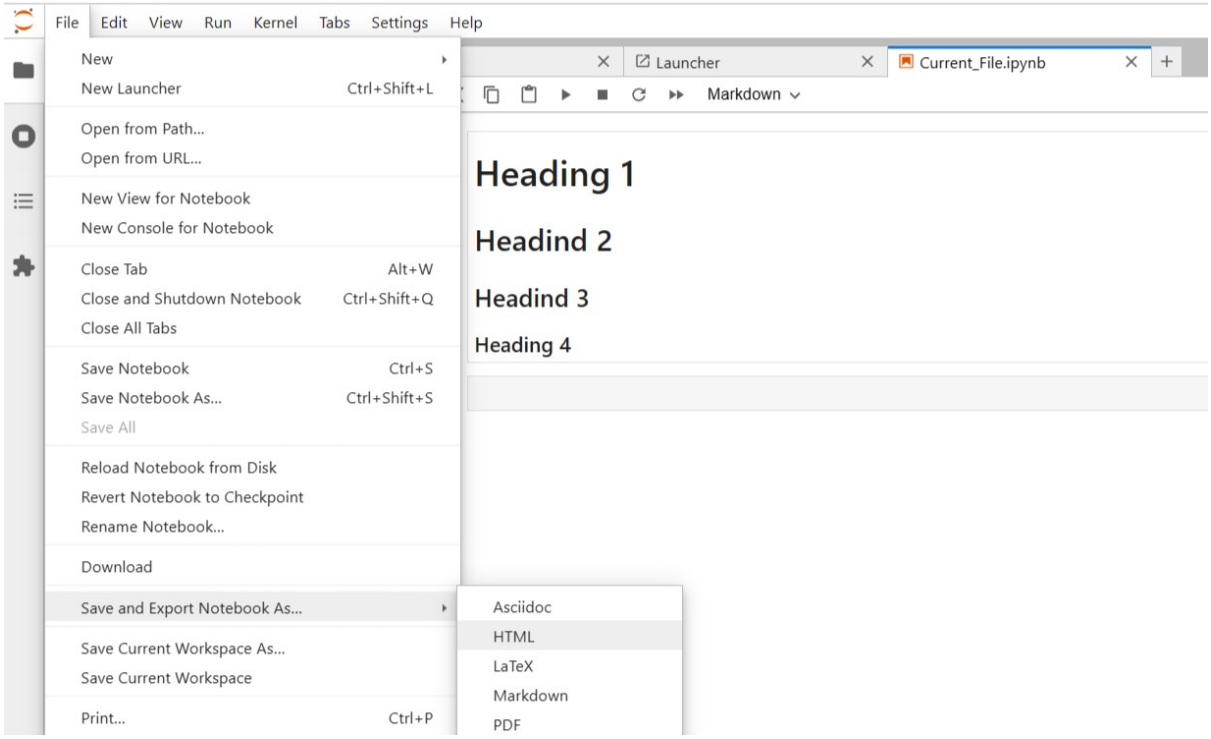
SYNTAX HIGHLIGHTING

Code blocks can be created by surrounding the code with three backticks (```) and specifying the language for syntax highlighting. But, Jupyter notebook displays do not use the syntax

highlighting theme in Preferences to display code blocks. The display is the same no matter what the syntax highlighting selection



EXPORTING YOUR CURRENT NOTEBOOK TO AN HTML FILE

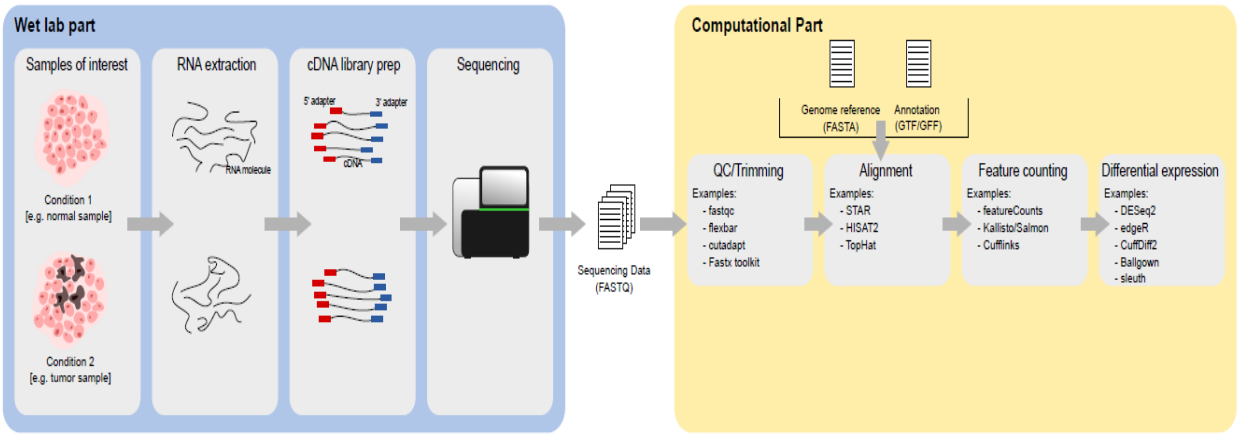


RNA-SEQ ANALYSIS: HIGH LEVEL OVERVIEW

The Goal of Sequencing

Abstracting from the whole wet lab part, the overall goal of the laboratory part is to obtain accurate sequence data. Accurate means that both, the sequence information itself (the nucleotides) as well as the quantification of the sequences shall be as precise and bias-free as possible. Sequence information can be obtained by various sequencing technologies (Illumina, PacBio, Oxford Nanopore) and different sources (WGS, Exom-seq, RNA-seq, ChIP-seq, CLIP, etc.).

From RNA-Seq Data to Differential Expression Analysis



Shown is only an exemplary high-level workflow assuming using a short-read sequencing approach (Illumina). Other technologies can be used. The choice of the technology and the chosen input material specifies the needed steps in the laboratory.
Besides sequencing-data a reference genome and the corresponding annotation is needed to perform a typical differential expression analysis.

FILE FORMATS

1. FASTA

The FASTA file format is a text-based format storing either nucleotide or protein sequences. A new sequence entry is indicated by the ">" sign typically followed by a sequence name/identifier and maybe additional information.

>BD300317.1 Attenuated influenza virus useful as vaccine
TGTGGTGTGAATGGTGATACTGTAGATTGGTCTAGGCCAGGCGGTGCTGAGTTGCCGTTC
ACCATTGACAAGTAGTTTCTTCTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTNGANCCCNGGGCAANNANTNN
NGNGGCCCCC

2. FASTQ

The FASTQ file format is the typical output format from DNA sequencing machines and contains at least the obtained nucleotide sequence as well as a quality score for each nucleotide. Each entry consists of four lines:

1. "@" followed by the sequence identifier and the additional information of the sequence.
2. The second line has the sequence of nucleotides.
3. A line that separates the sequence from quality score called the separator line that has a "+" sign on it and nothing else.
4. A quality score in ASCII format for each nucleotide

@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!"((((**+))%%%++)(%%%%).1***-+*"))**55CCF>>>>>>CCCCCCC65

3. GTF

General Transfer Format or GTF is a file format used for representing gene annotations in genomic data. It is commonly used for representing gene and transcript models, as well as other types of genomic features such as exons, introns, and promoter regions. The GTF file format is specified using a tab-delimited text file, where each line represents a separate feature. Each line in the GTF file contains nine fields that provide information about the feature, such as its chromosomal location, its type, and any associated gene or transcript identifier.

The group field has been expanded into a list of attributes. Each attribute consists of a type/value pair. Attributes must end in a semi-colon, and be separated from any following attribute by exactly one space.

The attribute list must begin with the two mandatory attributes:

• gene_id value - A globally unique identifier for the genomic source of the sequence.

• transcript_id value - A globally unique identifier for the predicted transcript.

The figure below shown an example of GTF file:

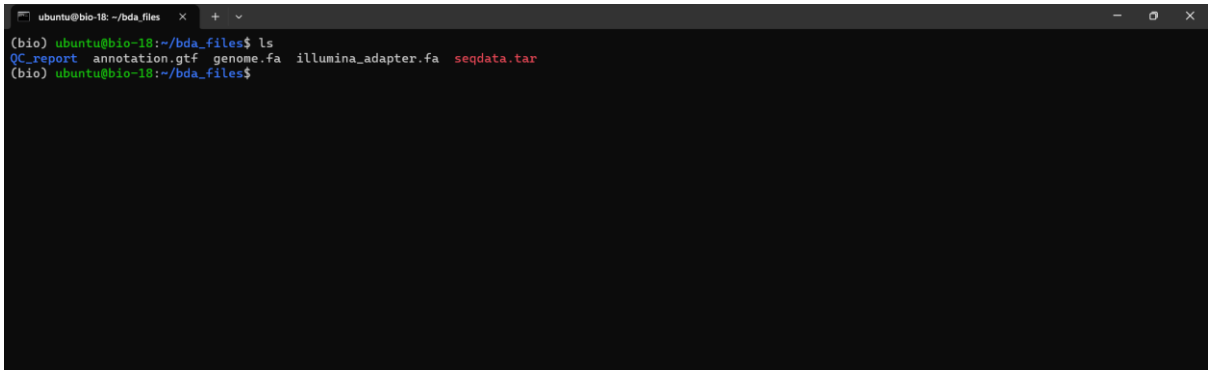
```
ctg123 .  gene            1000  9000  .  +  .  ID=gene00001;Name=EDEN
ctg123 .  TF_binding_site 1000  1012  .  +  .  Parent=gene00001
ctg123 .  mRNA            1050  9000  .  +  .  ID=mRNA00001;Parent=gene00001
ctg123 .  mRNA            1050  9000  .  +  .  ID=mRNA00002;Parent=gene00001
ctg123 .  mRNA            1300  9000  .  +  .  ID=mRNA00003;Parent=gene00001
ctg123 .  exon            1300  1500  .  +  .  Parent=mRNA00003
ctg123 .  exon            1050  1500  .  +  .  Parent=mRNA00001,mRNA00002
ctg123 .  exon            3000  3902  .  +  .  Parent=mRNA00001,mRNA00003
ctg123 .  exon            5000  5500  .  +  .  Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 .  exon            7000  9000  .  +  .  Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 .  CDS             1201  1500  .  +  0  ID=cds00001;Parent=mRNA00001
ctg123 .  CDS             3000  3902  .  +  0  ID=cds00001;Parent=mRNA00001
ctg123 .  CDS             5000  5500  .  +  0  ID=cds00001;Parent=mRNA00001
ctg123 .  CDS             7000  7600  .  +  0  ID=cds00001;Parent=mRNA00001
ctg123 .  CDS             1201  1500  .  +  0  ID=cds00002;Parent=mRNA00002
ctg123 .  CDS             5000  5500  .  +  0  ID=cds00002;Parent=mRNA00002
ctg123 .  CDS             7000  7600  .  +  0  ID=cds00002;Parent=mRNA00002
ctg123 .  CDS             3301  3902  .  +  0  ID=cds00003;Parent=mRNA00003
ctg123 .  CDS             5000  5500  .  +  1  ID=cds00003;Parent=mRNA00003
ctg123 .  CDS             7000  7600  .  +  1  ID=cds00003;Parent=mRNA00003
ctg123 .  CDS             3391  3902  .  +  0  ID=cds00004;Parent=mRNA00003
ctg123 .  CDS             5000  5500  .  +  1  ID=cds00004;Parent=mRNA00003
ctg123 .  CDS             7000  7600  .  +  1  ID=cds00004;Parent=mRNA00003
```

OBTAINING DATA

The data for RNA-seq analysis in this workflow is downloaded from the links shown in the links below.

- https://nextcloud.th-deg.de/s/BZdNP4BQqRLae7L/download/genome.fa
- https://nextcloud.th-deg.de/s/DFEkkW8aArwFPWN/download/annotation.gtf
- https://nextcloud.th-deg.de/s/zrjyWijeAjekRr7/download/illumina adapter.fa
- https://nextcloud.th-deg.de/s/jsbPzAmNfYRBgpk/download/seqdata.tar

Using the ***wget*** command we retrieved content from the above following links and download the data for the workflow.

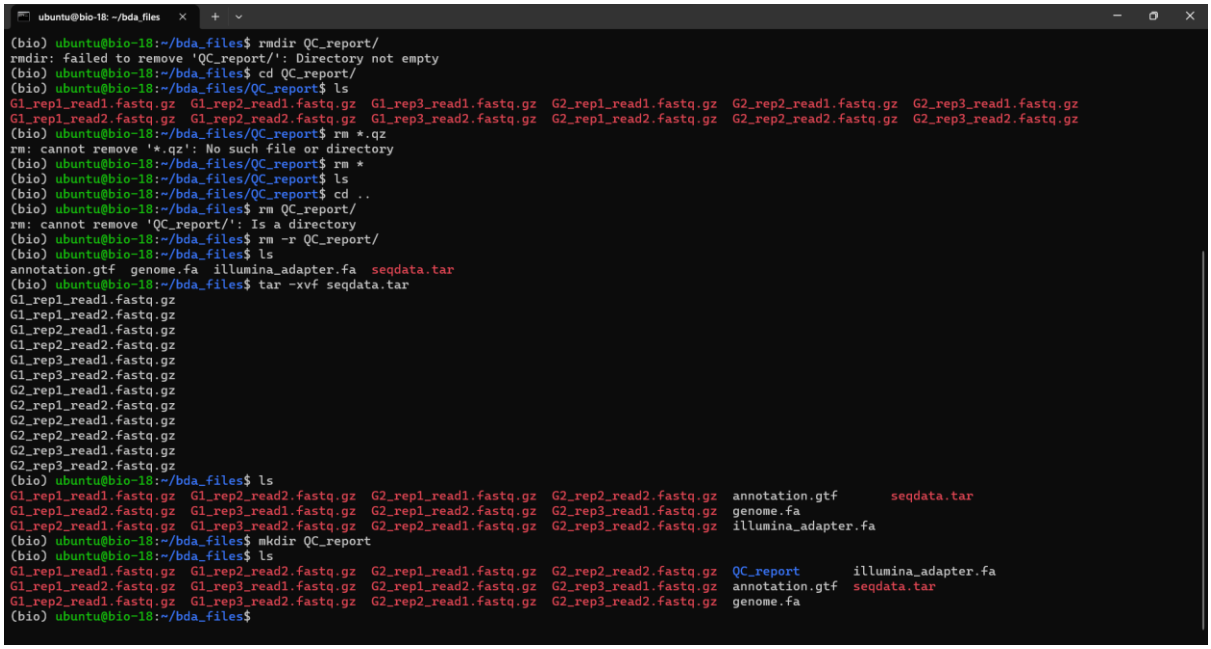


Tape archives (tar)

Tape archives are created and extracted using the Linux command "tar" which stands for tape archive. One of the crucial commands in Linux that provide archiving functionality is the tar command. Compressed or uncompressed Archive files can be created, maintained, and modified using the Linux tar program.

The following command is used to uncompressed the seqdata downloaded in the previous step:

$ tar -xvf seqdata.tar -C /home/ubuntu/seqdata



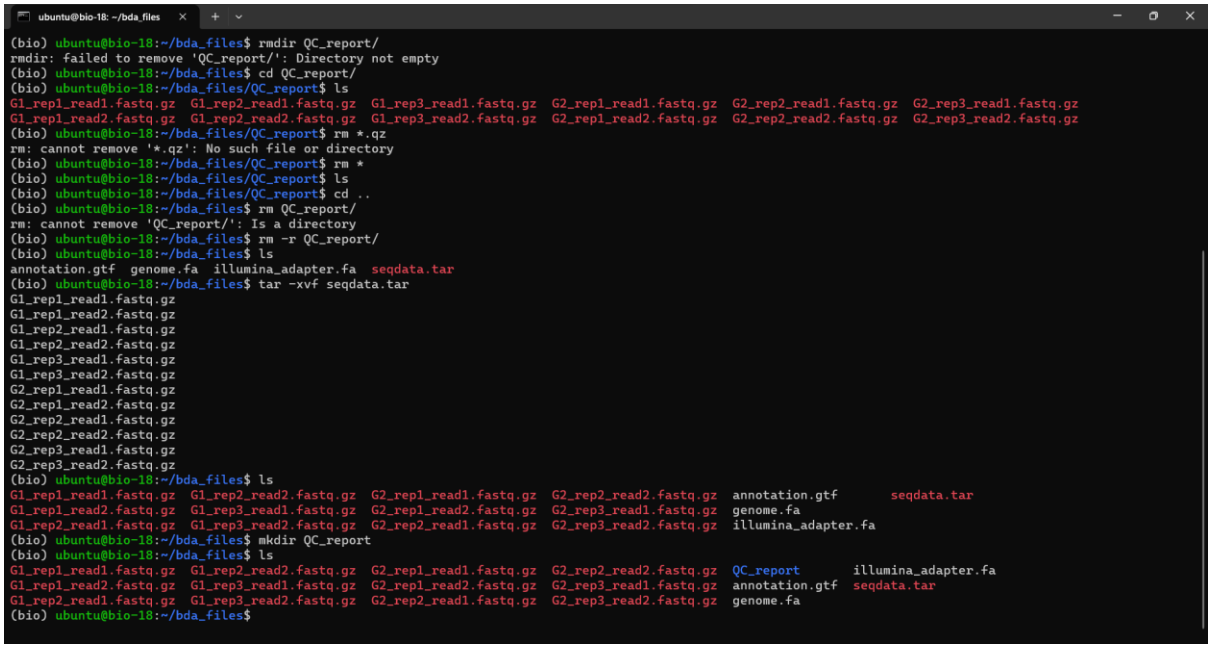
- The **-x** flag in this command extracts file from an archive.
- The **-v** flag verbosely lists the which are processed.
- The **-f** flag use archive file whereas;
- The **-C** option is to specify the path of the directory to which the files are being extracted.

This gives us 12 files for the data used in the current analysis.

QUALITY CONTROL WITH FASTQC

To determine quality of the raw sequencing data, quality control is performed. This frequently entails significant time savings during further analysis. In order to organize the QC report for all of the seqdata files, new directory **QC_report** is created using the command:

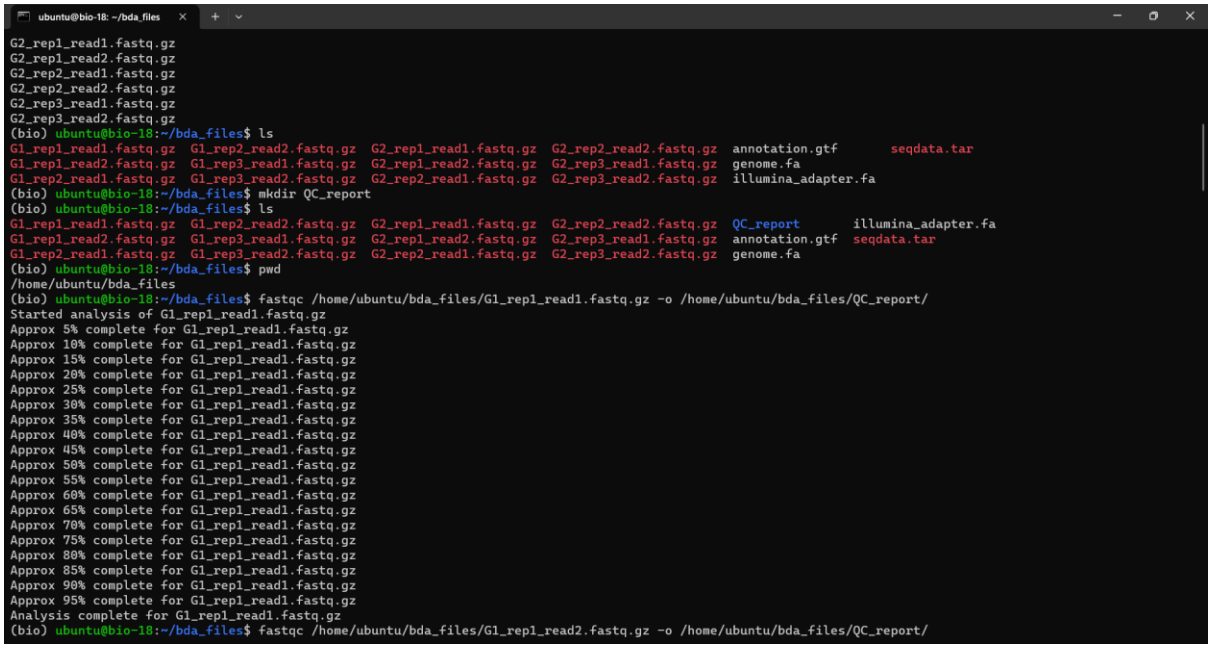
$ mkdir QC_report

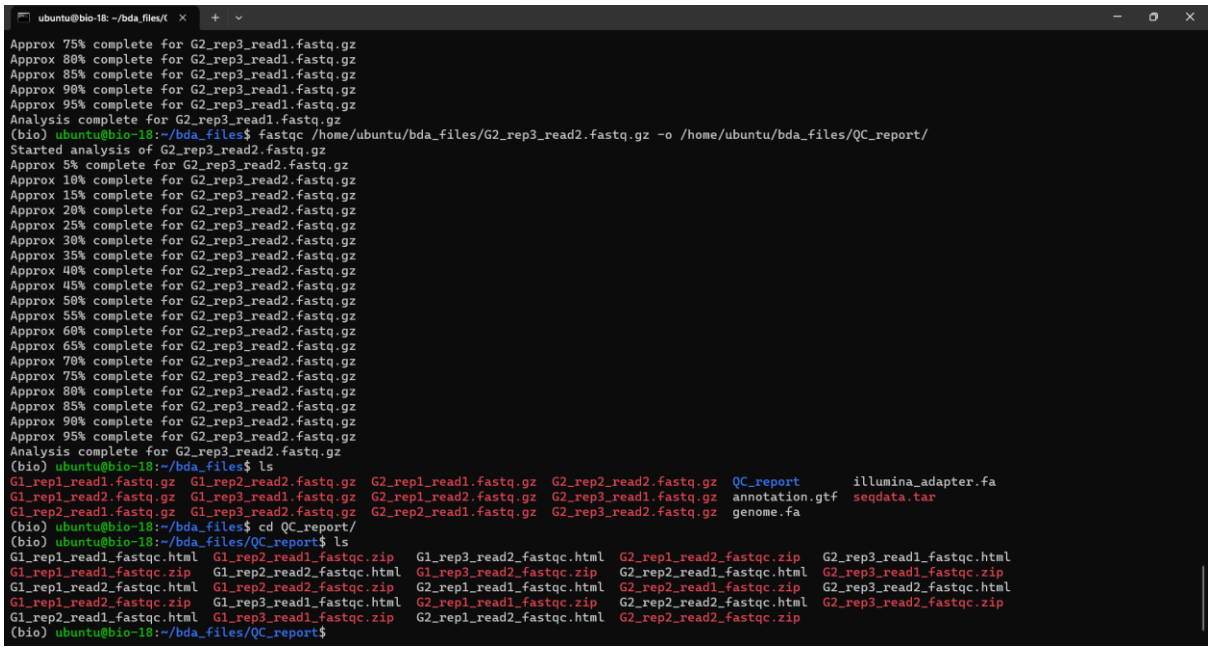


The following command is performed to obtain the QC report:

$ fastqc /home/ubuntu/bda_files/G1_rep1_read1.fastq.gz -o /home/ubuntu/bda_files/QC_report

The -o flag/option followed by the path specifies the directory in which the QC reports needs to be generated.



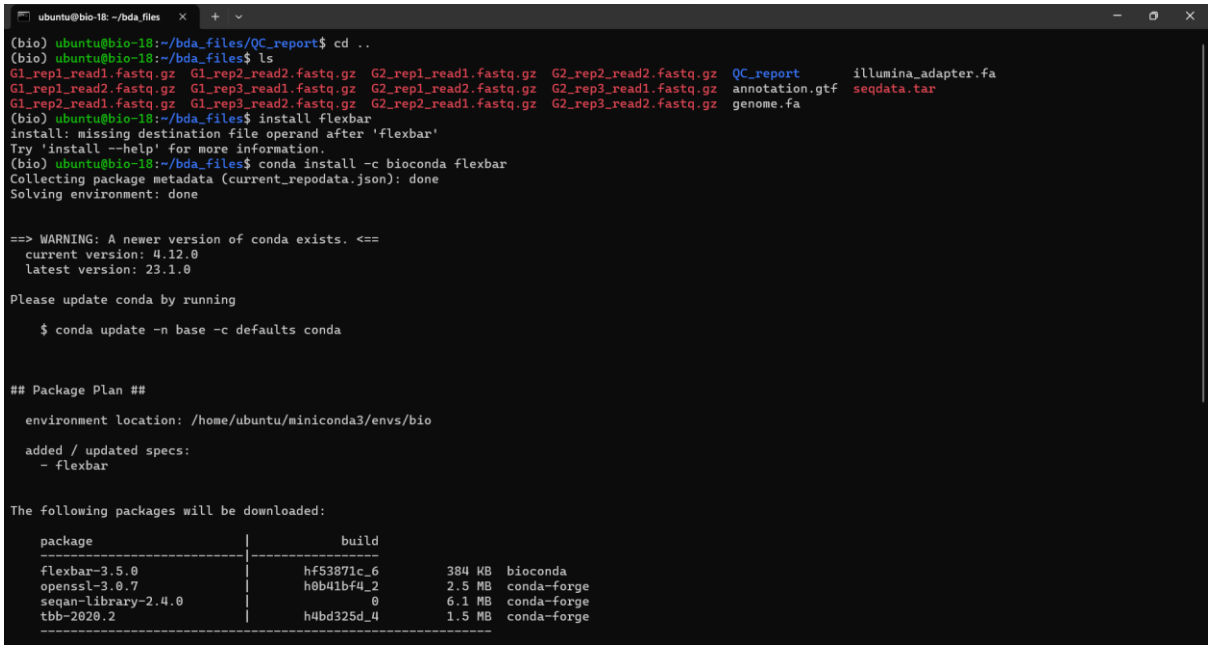


This quality control step on raw sequencing data is important because in addition to highlighting data concerns, regulating the quality of raw data makes it easier to immediately identify low-quality samples. It is quite clear that the sequencing quality is decreasing at the end of the sequence. This is typical for short read sequencing data and this trend is seen in other reads of the data also.

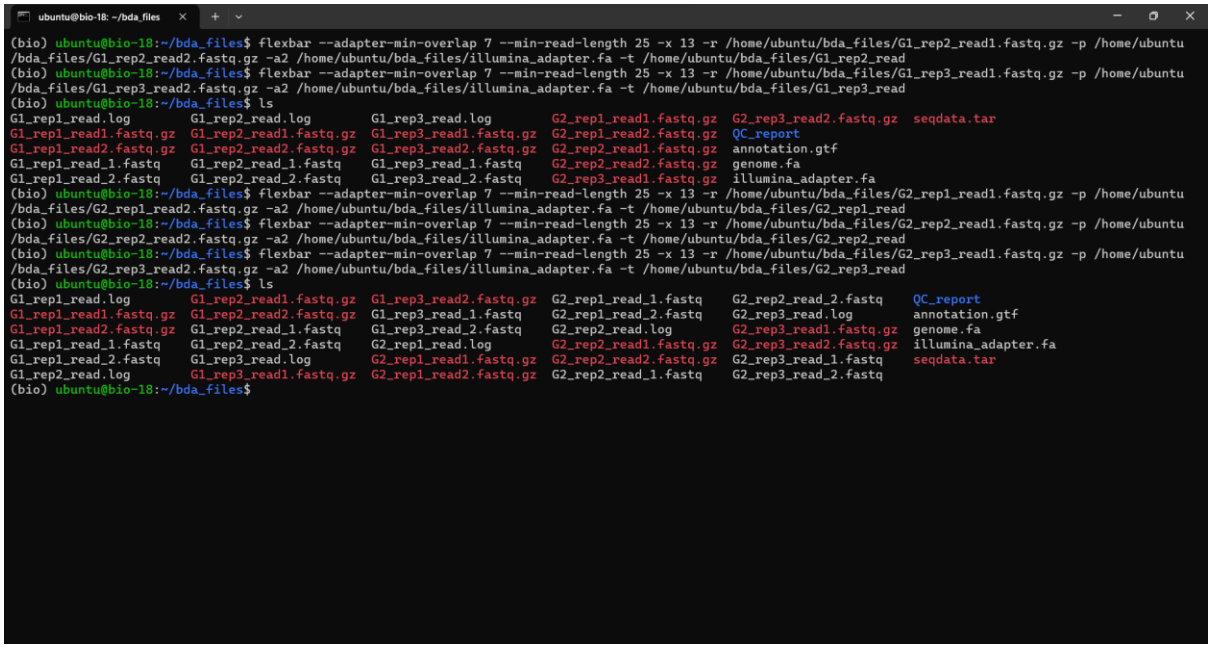
ADAPTER TRIMMING WITH FLEXBAR

Preprocessing is done on data obtained from high-throughput sequencing using the application Flexbar. Demultiplexing of barcode runs, and elimination adapter sequences is done. There are a number of adapter removal presets for Illumina libraries. Flexbar uses multicore parallelism and SIMD to compute precise overlap alignments. In addition, features for trimming and filtering are included, such as trimming of homopolymers at read ends. Flexbar boosts genome and transcriptome assembly and read mapping rates. A flexible method can be used to derive distinctive molecular IDs. Data in the fasta and fastq formats from several sequencing platforms are supported by the software.

Flexbar is installed in the **bio** environment of conda using the following command:
$ conda install -c bioconda flexbar



It is important for this program that the adapter sequences are at the end as **Flexbar** only leaves the sequence that is between two adapters and discard everything else. Therefore, if the adapters are in the middle, important sequence information could be lost.



The options used in this command are as follows:

- **--adapter-min-overlap 7**: Used to specify the minimum overlap length between reads and adapters for them to be considered as adapter-trimmed.
- **--min-read-length 25**: Determines the minimum length a read must have after trimming to be kept for further analysis.
- **-threads 1**: Used to define the number of threads to use during the trimming process.
- **-x 13**: Specifies the error rate in base-pair substitutions allowed during the adapter search.
- **-r /home/ubuntu/bda_files/G1_rep1_read1.fastq.gz:** Used to state the location and name of the first read file (read1) in the Fastq format.
- **-p /home/ubuntu/bda_files/G1_rep1_read2.fastq.gz:** Determines the location and name of the second read file (read2) in the Fastq format.
- **-a2 /home/ubuntu/bda_files/illumina_adapter.fa:** Used to state the location and name of the adapter file in Fasta format.
- **-t /home/ubuntu/bda_files/"sample_name":** Defines the location and prefix for the output files as it will generate two output files 1 and 2.

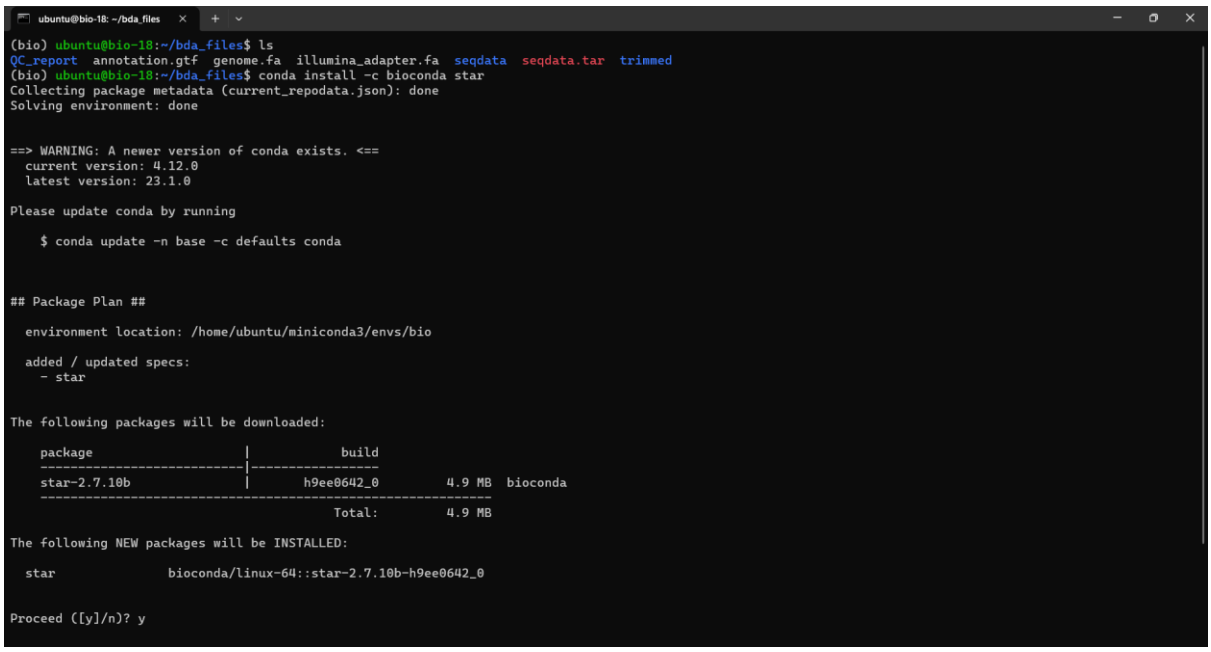
READ ALIGNMENT WITH STAR

STAR (Spliced Transcripts Alignment to a Reference) is a fast alignment algorithm albeit having relatively higher memory requirements. It is based on maximum mappable length approach that involves splitting individual reads into pieces or seeds and identifying best segments that can be mapped for each seed. Next step, segments are stitched together and mapped using the genome sequence as an uncompressed suffix array including information of splice sites.

STAR provides two options to either create a custom genome index using suffix arrays or directly utilize available STAR references indices. To include SJ annotation into the mapping process, a separate splice junction reference file is required while constructing a customized reference index.

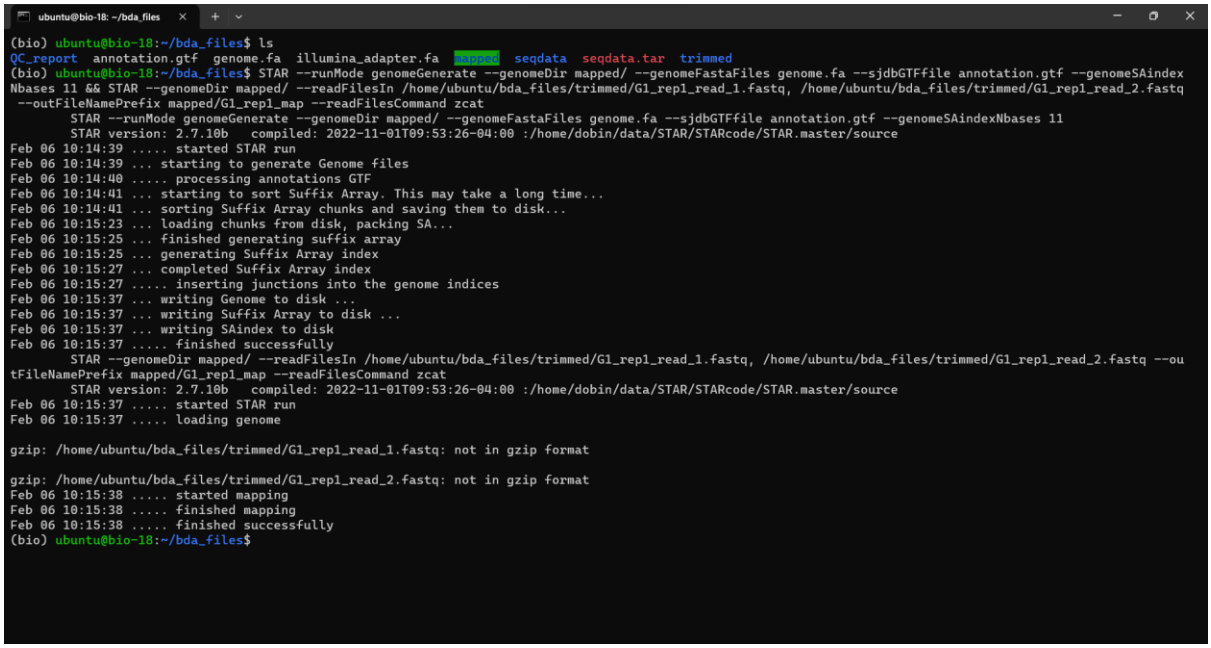
The following command is used in order to install the STAR into the **bio** environment:

$ conda install -c bioconda star



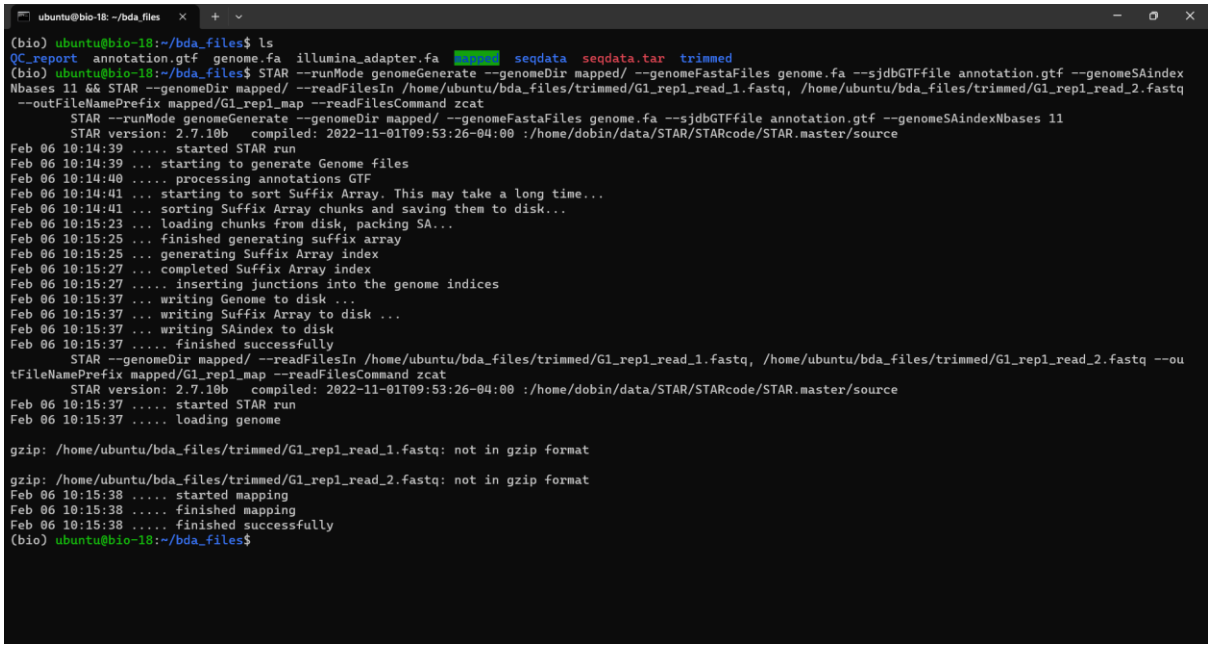
After installation, a new directory to store the alignment file is made and the reference index is created by following command:

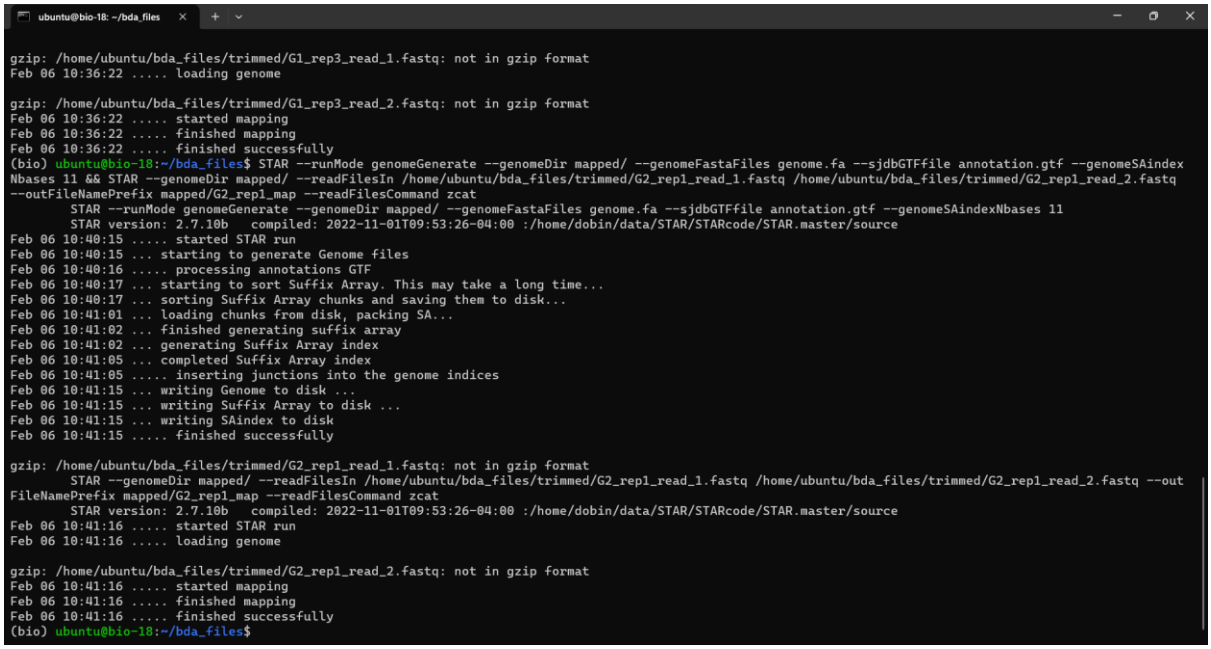
$ STAR --runMode genomeGenerate --genomeDir mapped/ --genomeFastaFiles genome.fa --sjdbGTFfile annotation.gtf --genomeSAindexNbases 11



Here's an explanation of each option in the command:

- **--runMode genomeGenerate:** To specify the run mode of the STAR program. In this case, it is set to genomeGenerate mode, which generates a genome index from the reference genome and annotation file.
- **--genomeDir /home/ubuntu/bda_files/mapped/:** It identifies the directory where the genome index will be stored.
- **--genomeFastaFiles /home/ubuntu/bda_files/genome.fa:** This option is used to specify the FASTA file of the reference genome.
- **--sjdbGTFfile /home/ubuntu/ bda_files /annotation.gtf:** To describe the GTF (Gene Transfer Format) file of the annotation.
- **--genomeSAindexNbases 11:** It defines the number of bases for constructing the suffix array index of the genome.
- **--genomeDir /home/ubuntu/bda_files/mapped/:** To define the directory where the previously generated genome index is stored.
- **--readFilesIn:** This option defines the paired-end RNA-seq read files that will be aligned to the reference genome.
- **--outFileNamePrefix: /home/ubuntu/bda_files/mapped/G1_rep1_map:** Used to state the prefix for the output file names.
- **--readFilesCommand zcat:** To map the paired-end reads for each sample





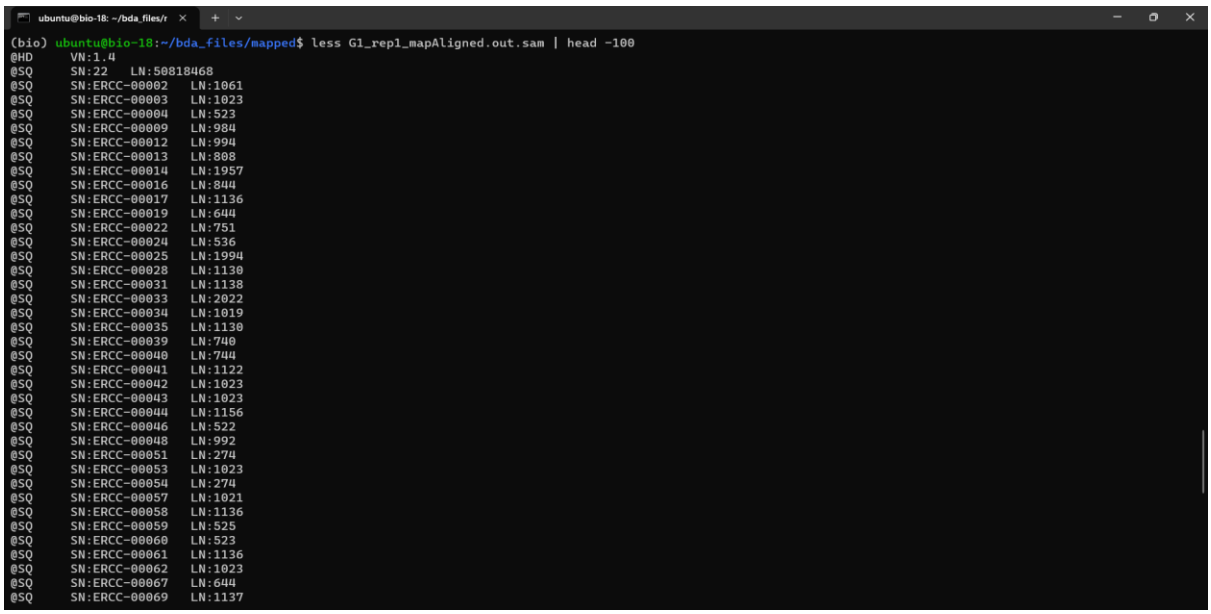
STAR produces multiple output files but the final log file is often used for quality assurance and contains summary statistics about the mapping job. For example,

- to determine the percentage of uniquely mapped reads.
- to check how many reads were unmapped.

SAM FILE

Sequence Alignment/Map format (SAM) is the general output format used by NGS-read alignment algorithms. SAM file is made of an alignment section and an optional header section. The alignment section has 11 necessary columns and a TAB-delimited style that offers comprehensive alignment information.

The format of a SAM file by running one of the SAM files generated in the last step is shown:



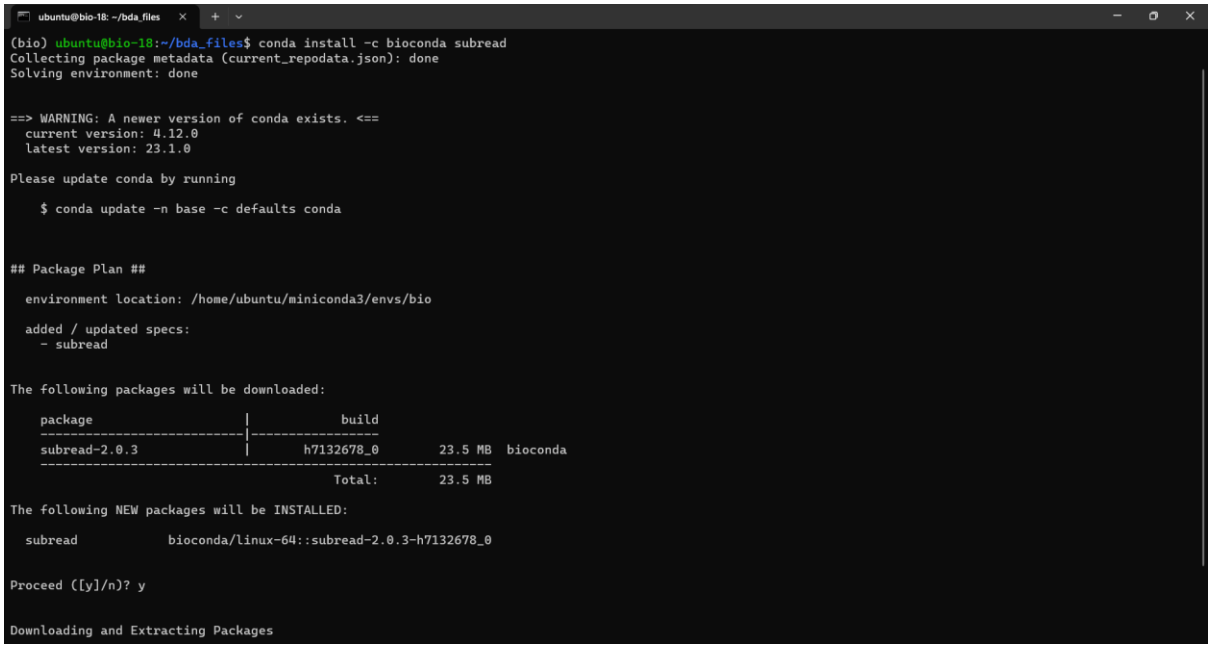
QUANTIFICATION WITH FEATURECOUNTS

The program featureCounts will be used to count the number of mapped reads per gene.

FeatureCounts is a software program that performs read quantification for high-throughput sequencing data, such as RNA-Seq and ChIP-Seq data. It provides an efficient and accurate way to count the number of reads that map to genomic features, such as genes, exons, and introns. The output of featureCounts can be used for downstream differential expression analysis, gene expression profiling, and epigenetic studies. Additionally, it provides a variety of options for read summarization and can handle multi-mapping reads and paired-end sequencing data.

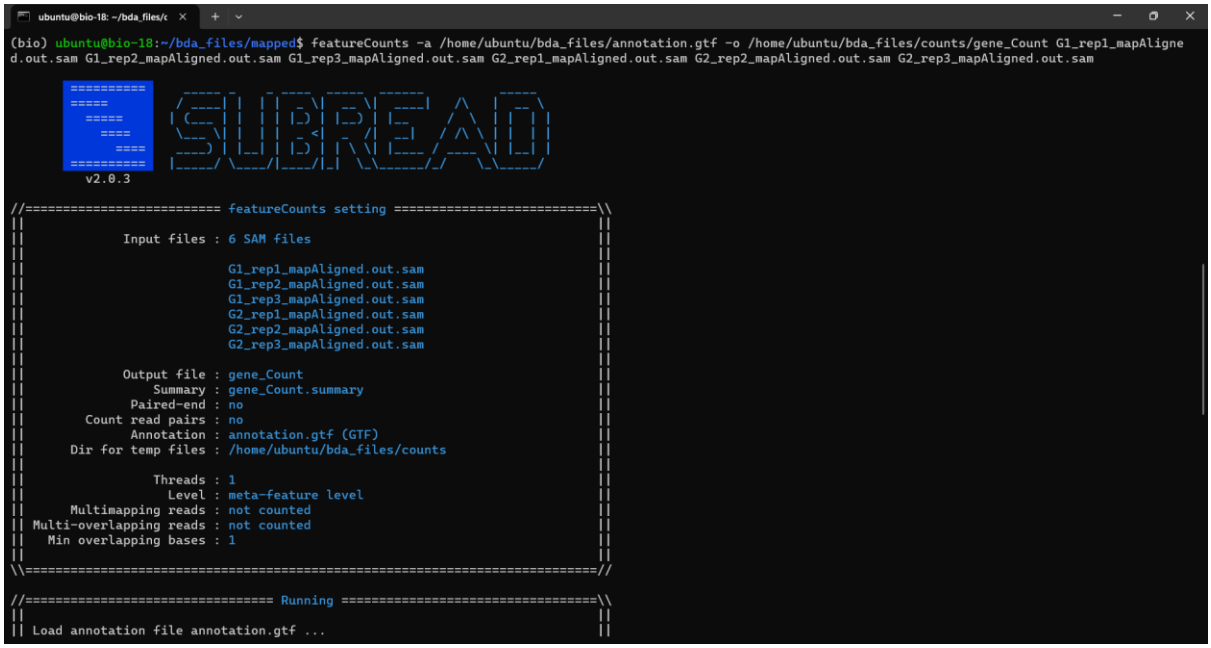
The "featureCounts" program can be found in the "subread" package in the Bioconda channel. To install it into the **bio** environment, the following command is used:

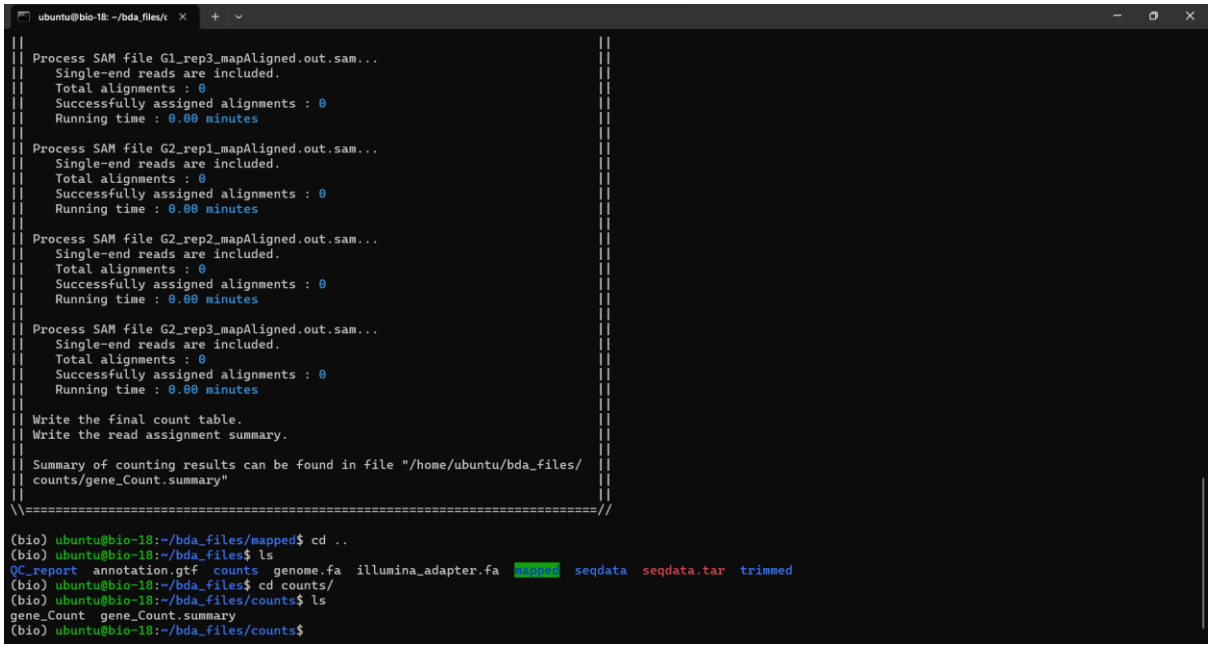
$ conda install -c bioconda subread



A new directory called **counts** is created to store the output files of the featureCounts.

To align featureCounts to the SAM files generated in the last step, command used is:





Parameters used are as follows:

- The **-a** options specify the annotation file for the genomic features to be counted.
- The **-o** option on the other hand specify the directory in which the output files need to be generated.

THE R PROGRAMMING LANGUAGE

R is free and open-source language for data analysis and statistics. It is the de-facto standard for data analysis. R is used in notebooks together with markdown to make reproducible data analysis possible. Using R, the gene counts file can be arranged in table as well as visualized in plot. The following steps organize the data in the gene counts file into table and then plots the data.

```
In [7]: data = read.table("/home/ubuntu/featureCounts_output.txt", header=T, stringsAsFactors = T)
```

```
In [8]: head(data)
        colnames(data)
```

	Geneid	Chr	Start	
	<fct>	<fct>	<fct>	
1	U2	22	10736171	
2	FRG1FP	22;22;22;22;22;22;22;22;22	10939388;10940597;10941691;10944967;10947304;10949212;10950049;10959067;10961283	10939423;10940
3	CU104787.1	22;22	11065974;11067335	
4	BAGE5	22;22	11066501;11067985	
5	ACTR3BP6	22;22	11124337;11124508	
6	5_8S_rRNA	22	11249809	

'Geneid' · 'Chr' · 'Start' · 'End' · 'Strand' · 'Length' · '...results.align_STAR.G1_rep1_Aligned.out.sam' · '...results.align_STAR.G1_rep2_Aligned.out.sam' · '...results.align_STAR.G1_rep3_Aligned.out.sam' · '...results.align_STAR.G2_rep1_Aligned.out.sam' · '...results.align_STAR.G2_rep2_Aligned.out.sam' · '...results.align_STAR.G2_rep3_Aligned.out.sam'

```
In [23]: count = data[,7:12]
         colnames(count) = 1:6
         count
```

A data.frame: 1378 × 6

1	2	3	4	5	6
<int>	<int>	<int>	<int>	<int>	<int>
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	2	0
0	0	0	0	1	0
0	0	0	0	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	2	0
0	0	0	0	1	0
0	0	0	0	2	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	0	0	0
0	0	0	1	2	0
0	0	0	0	0	0
0	0	1	7	5	10
0	0	0	0	2	0
0	0	0	0	0	0
0	0	0	2	1	2
⋮	⋮	⋮	⋮	⋮	⋮

143	139	135	989	656	940
0	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
3	3	5	12	2	11
3783	4906	4045	33528	20755	26500
19	31	20	143	80	125
0	0	1	0	0	0
235	284	225	2136	1413	1678
0	0	0	1	2	0
1	0	0	0	1	0
0	0	0	0	1	0
0	3	1	4	5	3
1	1	4	10	5	4
449	633	515	748	462	572
0	0	0	0	0	0
1	0	5	3	1	4
0	3	1	2	4	0
1	0	1	1	7	4
0	0	0	0	0	0
9	7	13	13	12	11
0	0	2	1	1	1
7	19	13	11	9	8
26	22	24	50	36	28
4	7	13	4	4	9
0	1	0	0	0	0
32	60	37	49	26	33
2	1	1	0	1	0
1	2	3	17	14	15
864	1139	985	1831	1226	1557

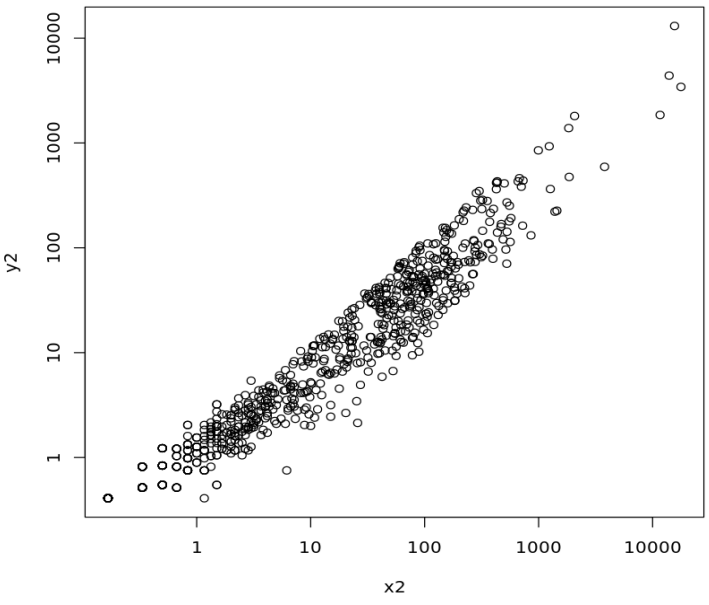
In [26]:
```
x = rowMeans(count)
range(x)
```

0 · 17737.5

In [34]:
```
y = numeric(nrow(count))
for(i in 1:nrow(count)){
    y[i] = sd(count[i,])
    }
```

In [36]:
```
selx = x > 0
sely = y > 0
x2 = x[selx&sely]
y2 = y[selx&sely]
```

In [44]:
```
plot(x2,y2, log="xy")
```



WORKFLOW MANAGER: SNAKEMAKE

The general idea of Snakemake is based on the GNU. Make principles: workflows are defined in terms of rules that define how to create output files from input files. Dependencies between the rules are determined automatically, creating a DAG (directed acyclic graph) of jobs that can be automatically parallelized. In other words, a workflow (pipeline) is decomposed into rules whereas each rule defines how to obtain output files from input files. Snakemake is an extension of the Python programming language. Thus, normal Python code can be used in Snakemake workflows.

To install Snakemake via Conda in your bio environment, the following command was used:

$ conda install snakemake

Snakemake can run a workflow in a" dry-run" mode specified by the -n option. Using this option only shows which steps would be performed without running the jobs. The -p option prints the shell commands used and -s specifies the snakefile:

$ snakemake -np -s snm0

Rules or outputfiles can also be used to run only specific parts of a workflow:

$ snakemake -np -s snm0 count_lines
$ snakemake -np -s snm0 ../lcounts/G1_rep1_read1

To finally run the workflow, we call:

$ snakemake --cores 1 -s snm0

RNA-SEQ WORKFLOW WITH SNAKEMAKE

The script for the snakemake workflow that automates all of the step done manually before in this report for RNA-seq raw sequence data analysis. A workflow that performs quality control, adapter trimming, mapping, and counting of gene expression data:

ubuntu@bio-14: ~/smake_exa

```
SAMPLES = ["G1_rep1", "G1_rep2", "G1_rep3", "G2_rep1", "G2_rep2", "G2_rep3"]

rule all:
    input:
        expand("/home/ubuntu/smake_example/RNA_Seq_Analysis/counts/gene_counts.txt", sample = SAMPLES)

rule fastqc:
    input:
        input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/fastq_raw/{sample}_read1.fastq.gz",
        input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/fastq_raw/{sample}_read2.fastq.gz"
    output:
        output1="/home/ubuntu/smake_example/RNA_Seq_Analysis/{sample}_read1_fastqc.html",
        output2="/home/ubuntu/smake_example/RNA_Seq_Analysis/{sample}_read2_fastqc.html"
    shell:
        "fastqc -o /home/ubuntu/smake_example/RNA_Seq_Analysis {input}"

rule adapter_trimming:
    input:
        input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/fastq_raw/{sample}_read1.fastq.gz",
        input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/fastq_raw/{sample}_read2.fastq.gz",
        input3="/home/ubuntu/smake_example/RNA_Seq_Analysis/illumina_adapter.fa"

    params:
        prefix = "/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read"

    output:
        "/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_1.fastq"
    shell:
        "flexbar --adapter-min-overlap 7 --min-read-length 25 -x 13 -r {input.input1} -p {input.input2}
        -a2 {input.input3} -t {params.prefix}"

```

The workflow defines 6 samples (G1_rep1, G1_rep2, G1_rep3, G2_rep1, G2_rep2, G2_rep3) which are listed in the **SAMPLES** variable.

- **Rule all** declares that the input for the workflow is the "genes_counts.txt" file, which is expanded for all 6 samples.

- **Rule fastqc** performs quality control on the fastq files (read1 and read2) of each sample and generates the fastqc html reports.

- **Rule adapter_trimming** uses flexbar to trim adapters from the reads and generates the trimmed fastq files.

```
30
31 rule mapping:
32     input:
33         input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_1.fastq",
34         input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_2.fastq"
35
36     output:
37         "/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/{sample}Aligned.out.sam"
38
39     shell:
40         "STAR --runMode genomeGenerate --genomeDir /home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/
41         --genomeFastaFiles /home/ubuntu/smake_example/RNA_Seq_Analysis/genome/genome.fa
42         --sjdbGTFfile /home/ubuntu/smake_example/RNA_Seq_Analysis/genome/annotation.gtf
43         --genomeSAindexNbases 11 && STAR --genomeDir /home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/
44         --readFilesIn {input.input1} {input.input2}
45         --outFileNamePrefix /home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/{wildcards.sample}"
46
47 rule feature_counts:
48     input:
49         input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep1Aligned.out.sam",
50         input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep2Aligned.out.sam",
51         input3="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep3Aligned.out.sam",
52         input4="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep1Aligned.out.sam",
53         input5="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep2Aligned.out.sam",
54         input6="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep3Aligned.out.sam"
55     output:
56         "/home/ubuntu/smake_example/RNA_Seq_Analysis/counts/gene_counts.txt"
57     shell:
58         "featureCounts -a /home/ubuntu/smake_example/RNA_Seq_Analysis/genome/annotation.gtf -o {output}
59         -p {input.input1} {input.input2} {input.input3} {input.input4} {input.input5} {input.input6}"
60
sma
-- INSERT --
```

- **Rule mapping** maps the trimmed reads to a reference genome using STAR aligner and generates the mapped SAM files.

  The params argument can be used to provide parameters to a rule that can change the behaviour of the rule or provide additional input to the rule's commands. The params argument is a dictionary that can contain key-value pairs that can be used in the rule's shell commands or scripts.

  The params was used to add prefix to the output files generated by STAR.

- **Rule feature_counts** perform gene counting using the featureCounts tool on the mapped SAM files and generates the gene expression counts file.

OUTPUT: COUNT FILES