BIOMEDICAL DATA ANALYSIS

SEMINAR PROTOCOL

STUDENT NAME : SHEFALI BADRE

IMMATRICULATION NUMBER : 22203735

OUTLINE :

1. **Introduction ****
    a. Biomedical Data Analysis
    b. Reproducible Research
2. **Technical Setup**
    a. Conda/Bioconda
    b. Environments / Virtual Machines / Containers
    c. Notebooks / Jupyterlab / Markdown
3. **RNA-Seq Analysis**
    a. Overview
    b. File formats
    c. Quality control and adapter trimming
    d. Read alignment
    e. Feature counting
    f. The R Programming Language
4. **Workflow Manager**
    a. General overview about workflow managers
    b. Snakemake
    c. RNA-Seq Analysis pipelines with Snakemake

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

INTRODUCTION : BIOMEDICAL DATA ANALYSIS

TASK : Briefly summarize the term "Biomedical Data Analysis".

Biomedical data analysis refers to the process of extracting meaningful insights and knowledge from large and complex sets of biomedical data. 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. 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. Effective biomedical data analysis 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.

INTRODUCTION : REPRODUCIBLE RESEARCH

Briefly summarize the course findings about "Reproducible Research".

Download and read the paper "**Reproducible Research in Computational Science**" by Roger Peng published in Science 2011.

TASK : Answer the following questions:

1. What's "reproducible research" and why is it needed?

Answer:

Replication and reproducibility are related but distinct concepts in scientific research.

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.

2. What's the difference between replication and reproducibility?

Answer:

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.

3. What are the technical challenges to enable "reproducible research"?

Answer:

There are several technical challenges to enable reproducible research:

i. Data management: The availability and accessibility of the raw data and its documentation is crucial for reproducibility.
ii. Computational environment: Specifying the exact software, libraries, and their versions used in the research is important to ensure that others can recreate the results.
iii. Version control: Keeping track of changes to the code and data throughout the research process is necessary to enable reproducibility.
iv. Collaboration: Collaborating on research projects with others can lead to a loss of reproducibility if the project is not properly documented and managed.
v. Scalability: With the increasing size and complexity of research data and models, it can be challenging to ensure reproducibility at scale.
vi. Human error: Even with the best tools and practices, human error can still occur, leading to incorrect or non-reproducible results.
vii. Implementation: Implementing reproducibility effectively requires effort and expertise, as well as a culture that values and encourages reproducibility.

TASK : Briefly answer the following questions and explain their connection to "reproducible research":

1. What is NCBI SRA?

Answer:

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.

2. What is conda/biodonda?

Answer:

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.

3. What is Jupyterlab?

Answer:

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 : CONDA / BIOCONDA

OVERALL GOAL: Set up your system using conda/bioconda. Create a new environment for this course and install a couple of programs into the new environment.

1. **Find documentation(s) about how to install conda on your remote machine. Follow the instructions and install conda. (Hint, there is an official conda documentation)**

Answer:

Steps for installing conda on Linux as mentioned in the official documentation of conda.

https://docs.conda.io/projects/conda/en/stable/user-guide/install/linux.html

Steps:

i. In your terminal window, run:
`$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh`

ii. Install the downloaded miniconda file with the following command:
`$ bash Miniconda3-latest-Linux-x86_64.sh`

iii. Follow the prompts on the installer screens.

iv. Install the downloaded miniconda file with the following command
`$ bash Miniconda3-latest-Linux-x86_64.sh`

v. Install the downloaded miniconda file with the following command:
`$ bash Miniconda3-latest-Linux-x86_64.sh`

2. **What are conda channels?**

Answer:

Conda channels are sources for packages in the Anaconda Distribution. When you use conda to install packages, it searches for the packages in the channels you have configured. Channels can be thought of as repositories or sources of packages.

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.

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 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.

3. Add the channels "bioconda" and "conda-forge".

Answer:

From the command line use **channels** by passing the argument:

```
$ conda config --add channels conda-forge
$ conda config --add channels bioconda
```

4. Create a new environment with the name "bio" with a Python version of 3.9.

Answer:

To create an environment with a specific version of Python:

```
$ conda create -n bio python=3.9
```

5. How can you activate/deactivate environments? Familiarize with activating and deactivating conda environments.

Answer:

To activate an environment, type:

```
$ conda activate bio
```

To deactivate an environment, type:

```
$ conda deactivate
```

6. **What is the content of the Linux shell PATH variable for the (base) conda environment and for your newly generated environment? Which differences do you observe and explain the consequence of the change?**

Answer:

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.

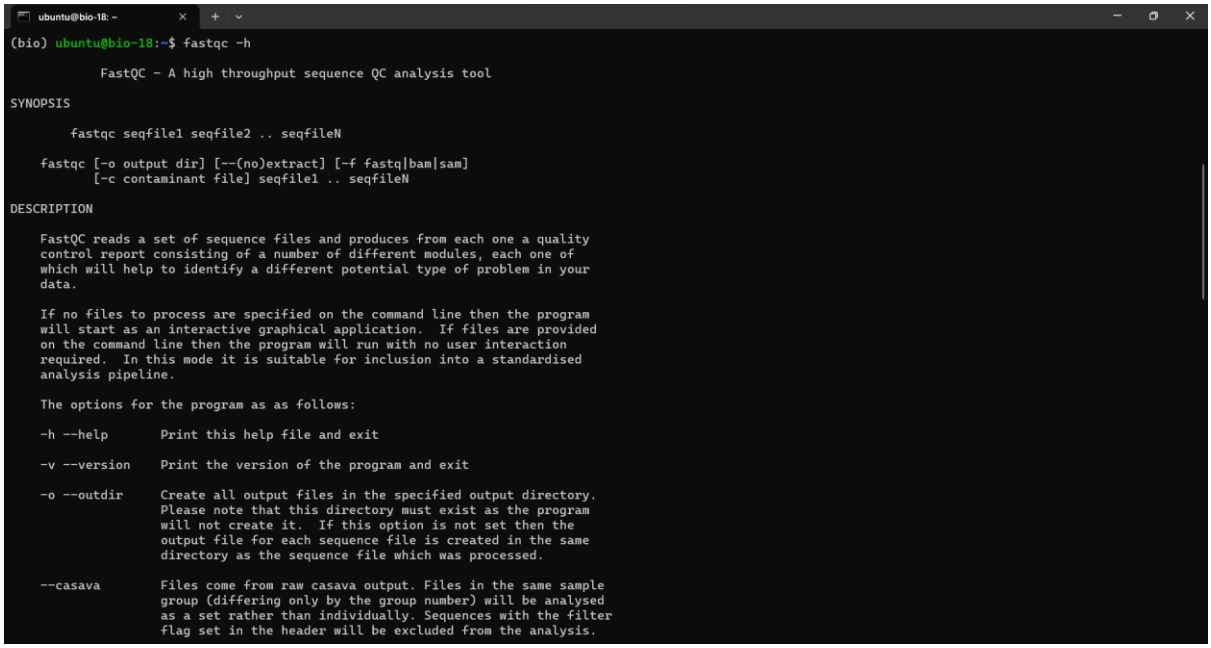
7. **Install "fastqc" via conda to your "bio" environment using conda. Test if you can start the program by typing "fastqc -h" into your shell. Report the output that you get.**

Answer:

Type the command:

```
$ conda install -c bio fastqc
```

Output of "fastqc -h":



8. Install "jupyterlab" and "bash kernel" via conda to your "bio" environment.

Answer:

Type the commands in the prompt:

```
$ conda install -c bio jupyterlab
$ conda install -c bio bash_kernel
```

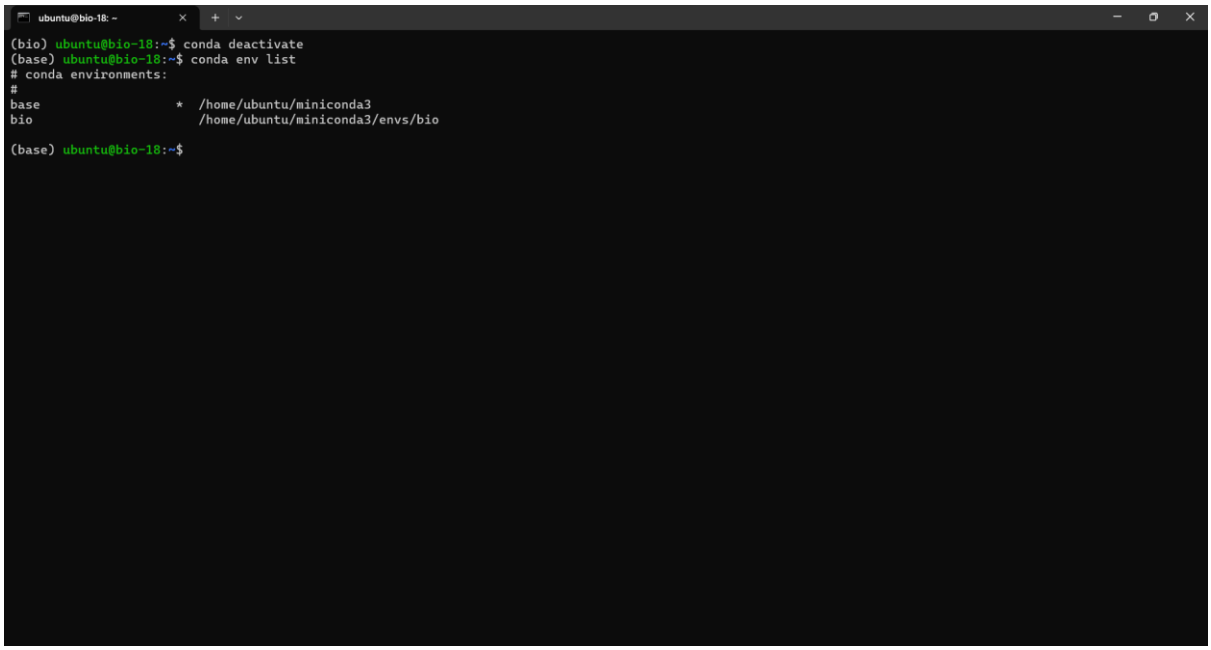
9. Find out how you can list all conda environments and report all your environments.

Answer:

Type the following command:

```
$ conda env list
```

Output:



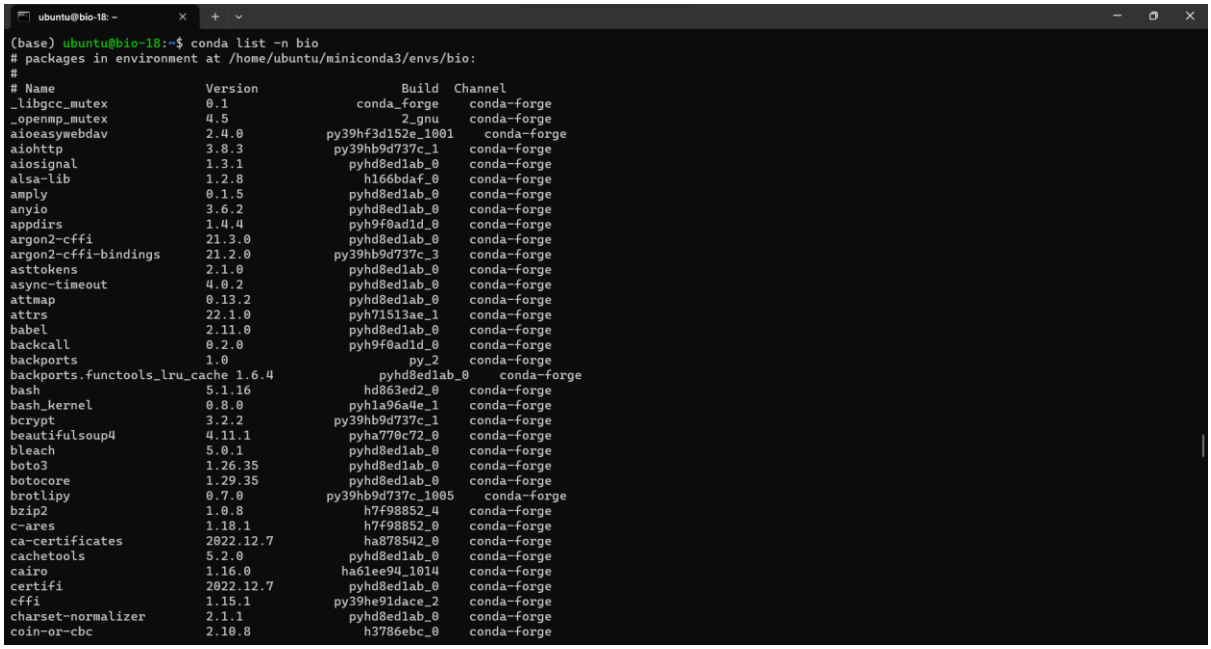
10. Find out how to report all installed programs of your "bio" environment and report the results.

Answer:

Type the command in the prompt:

```
$ conda list -n bio
```

Output:



TECHNICAL SETPUT : NOTEBOOKS / JUPYTERLAB / MARKDOWN

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 :

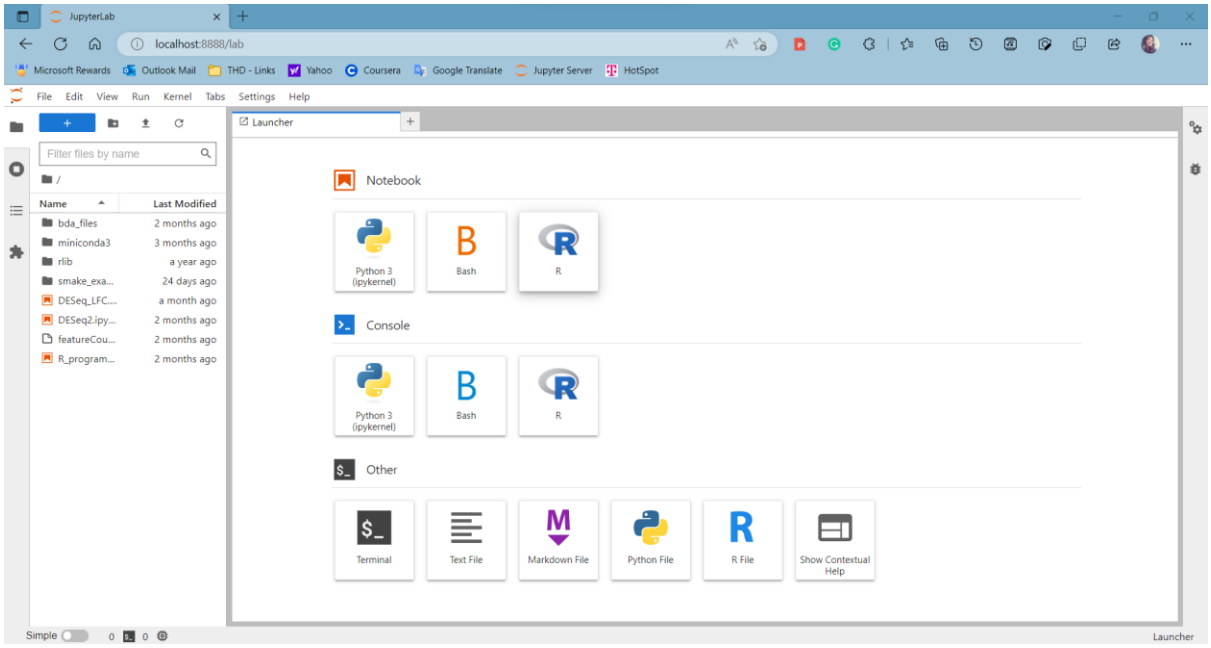
- Activate your "bio" environment in case it is not already active. First, we configure the Jupyter server.
  `$ jupyter server --generate-config`
  `$ jupyter server password`
- The terminal will ask you for a password. Type in a password and remember it. Next, the program asks to confirm the password.
- 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).

A short detour and introduction of helpful command line program for tmux :

1. **Find out what tmux is.**

Answer:

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.

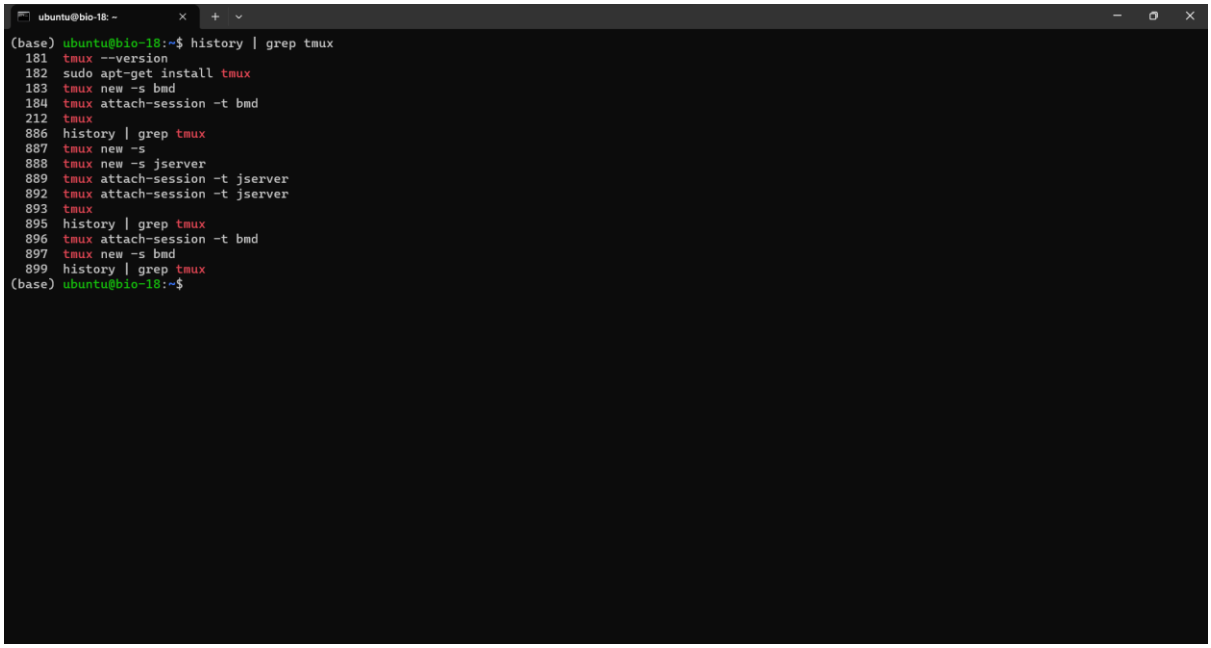
2. **Find out how to create a new session with a specified name and learn how to detach and retach to a session.**

Answer:

- To create a **new tmux session** with a specified name, run the following command in your terminal:
  `$ tmux new -s <session-name>`
- To **detach** from a tmux session, press Ctrl-b followed by d.
- To **reattach** to a tmux session, run the following command in your terminal:
  `$ tmux attach-session -t <session-name>`
  Where <session-name> is the name of the tmux session you want to reattach to.

3. **Create multiple sessions (no nested sessions) and familiarize with detach and attach to those.**

Answer:



4. **Within a session, and out how to create multiple panes and how to switch between them. What are the commands to create and to switch?**

Answer:

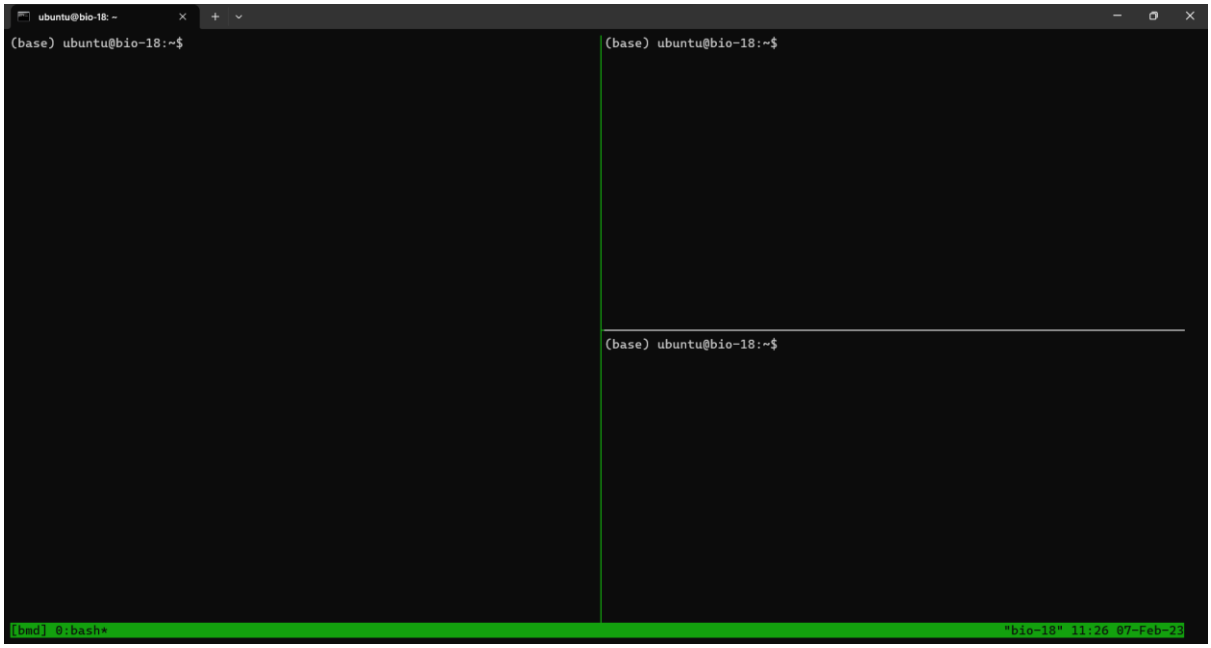
To create multiple panes within a tmux session, you can use the following key combinations:

- **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.



5. What is meant by the term's session, window, and pane in tmux?

Answer:

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.

TASKS :

1. **Create a new tmux session with the name "jserver". Activate the "bio" environment in that newly generated tmux session. Start the Jupyter server in that session. Detach from the session.**

Answer:

Type the command on the command line:

```
$ tmux new -s jserver
```

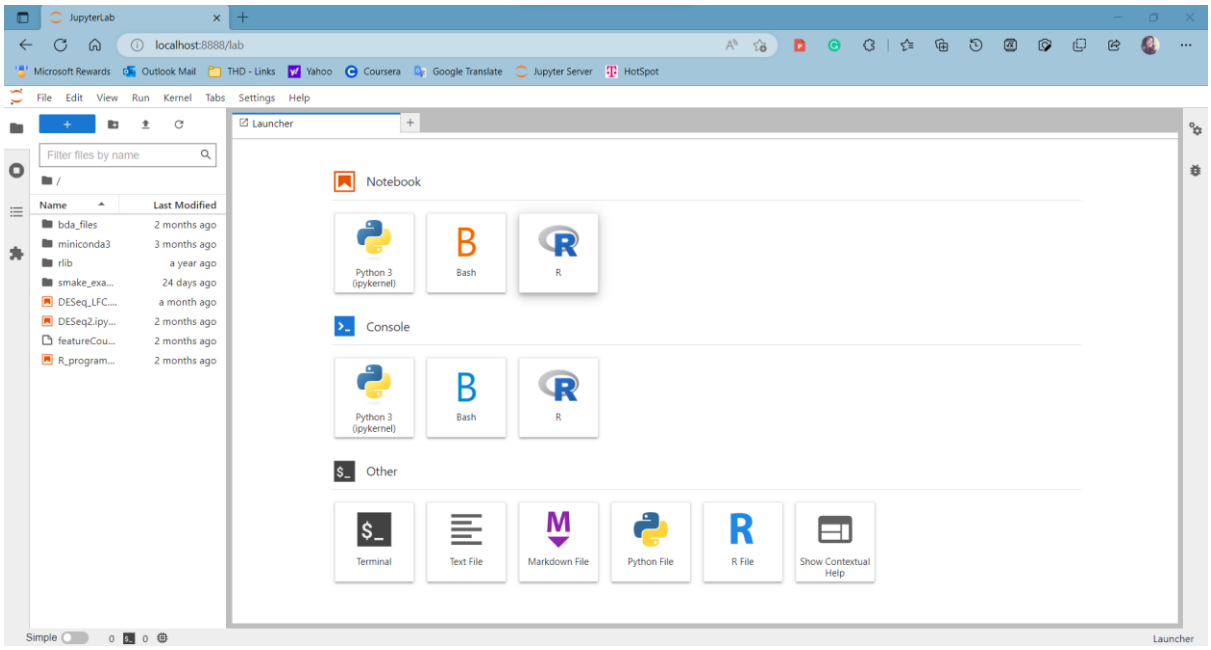
Inside the session start the Jupyter server:

```
$ conda activate bio
```

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

2. **Open a browser and try to connect to your Jupyter server.**

Answer:

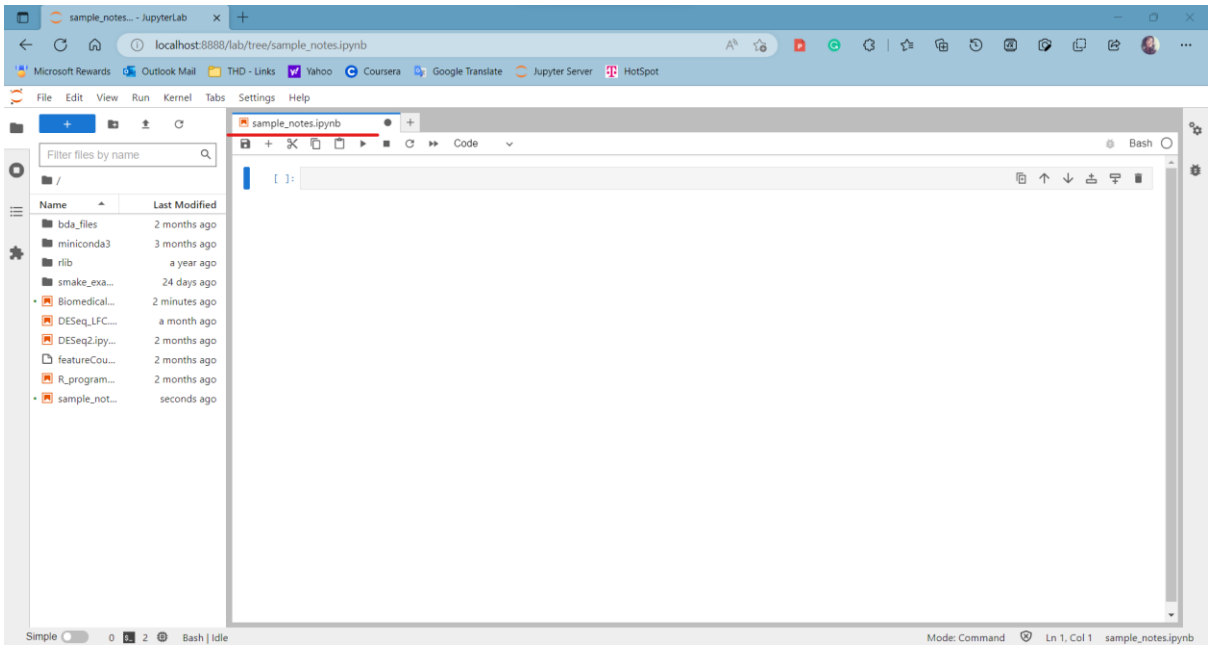


First Steps with Jupyter Notebooks

TASKS :

1. **Create a new notebook with any kernel (Python, bash or R).**

Answer:



2. **Change your first cell to a markdown cell. Have a look at https://www.markdownguide.org/basic-syntax/ and figure out how you change between the markdown (source) view of the cell and the rendered version. Try out some basic markdown syntax.**

Answer:

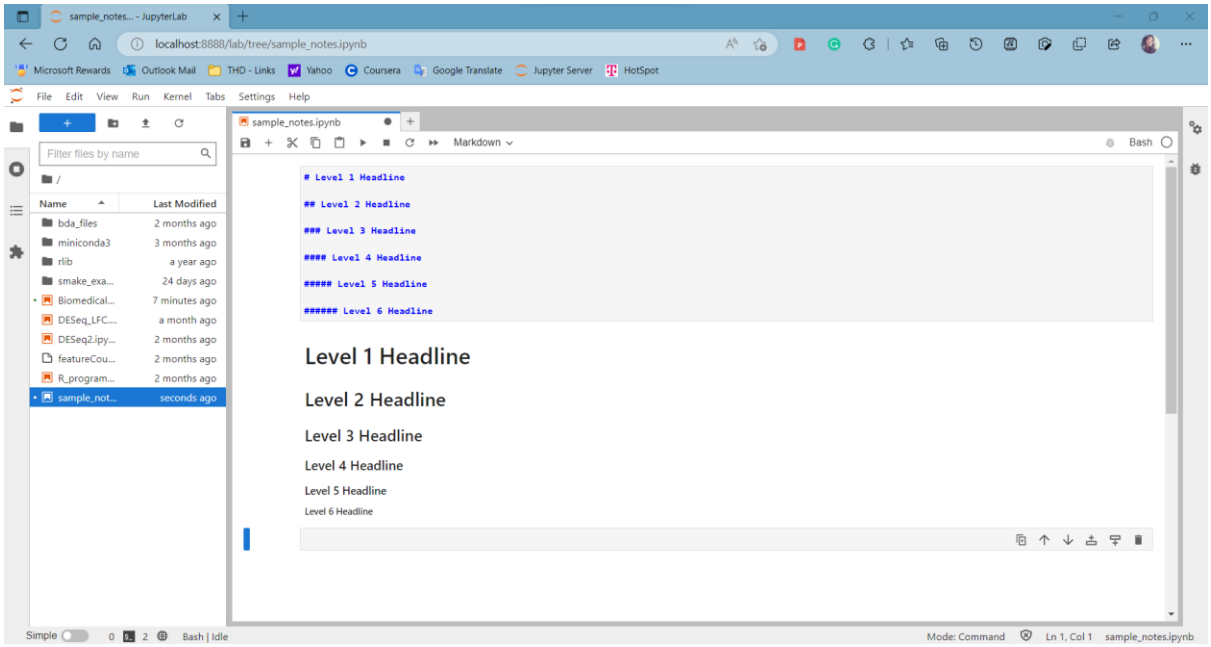
The markdown source view and the rendered view of a cell in Jupyter notebooks can be changed using the **Cell** menu in the Jupyter interface or by using keyboard shortcuts.

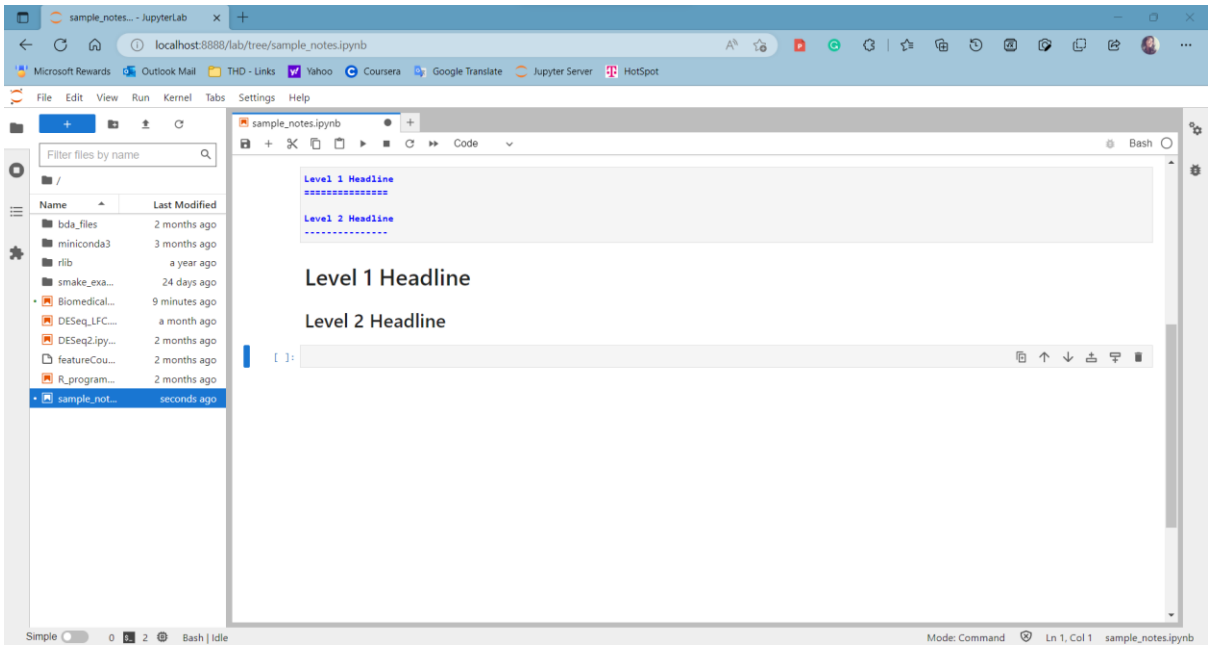
- To change to the markdown source view, you can either select **Cell** > **Cell Type** > **Markdown** from the menu, or press **Esc** followed by **m**.
- To change to the rendered view, you can either select **Cell** > **Run Cells** or press **Shift** + **Enter**.

i. **How can you create different levels of headlines?**

Answer:

In markdown, headlines can be created by using either **#** symbols or by underlining the text with **=** or **-** symbols. The number of **#** symbols or the length of the underlining determines the level of the headline.



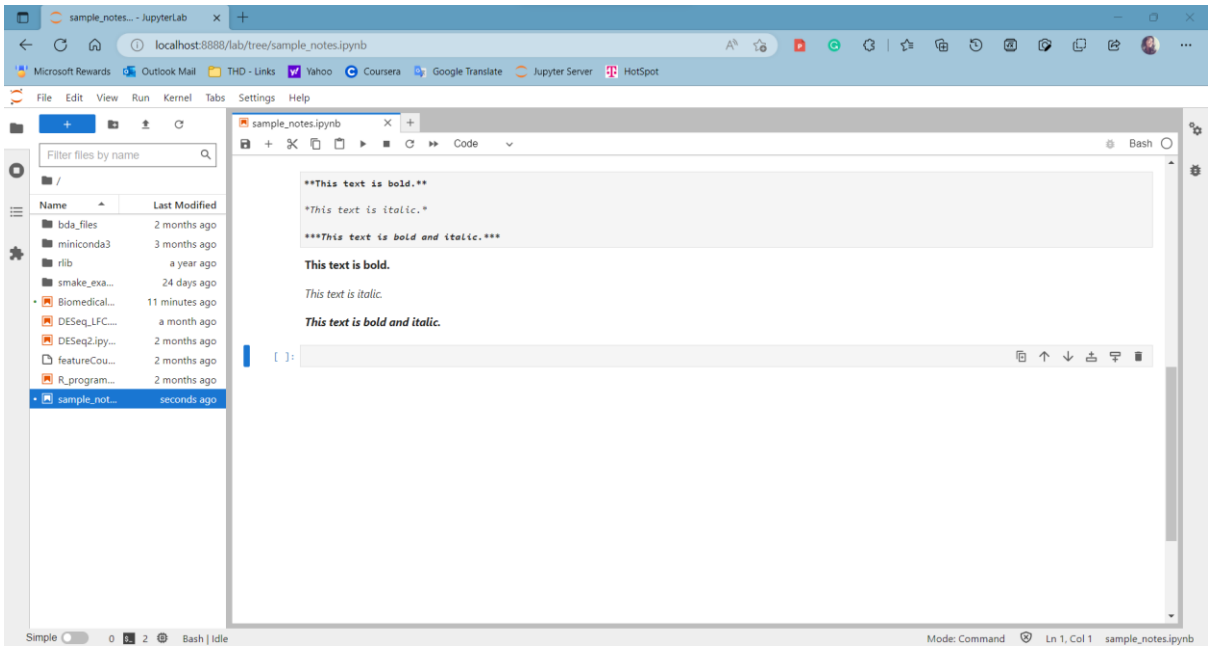


ii. How can you make text bold, italic or bold and italic?

Answer:

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

- Bold: ****text**** or **__text__**
- Italic: ***text*** or **_text_**
- Bold and italic: *****text***** or **___text___**

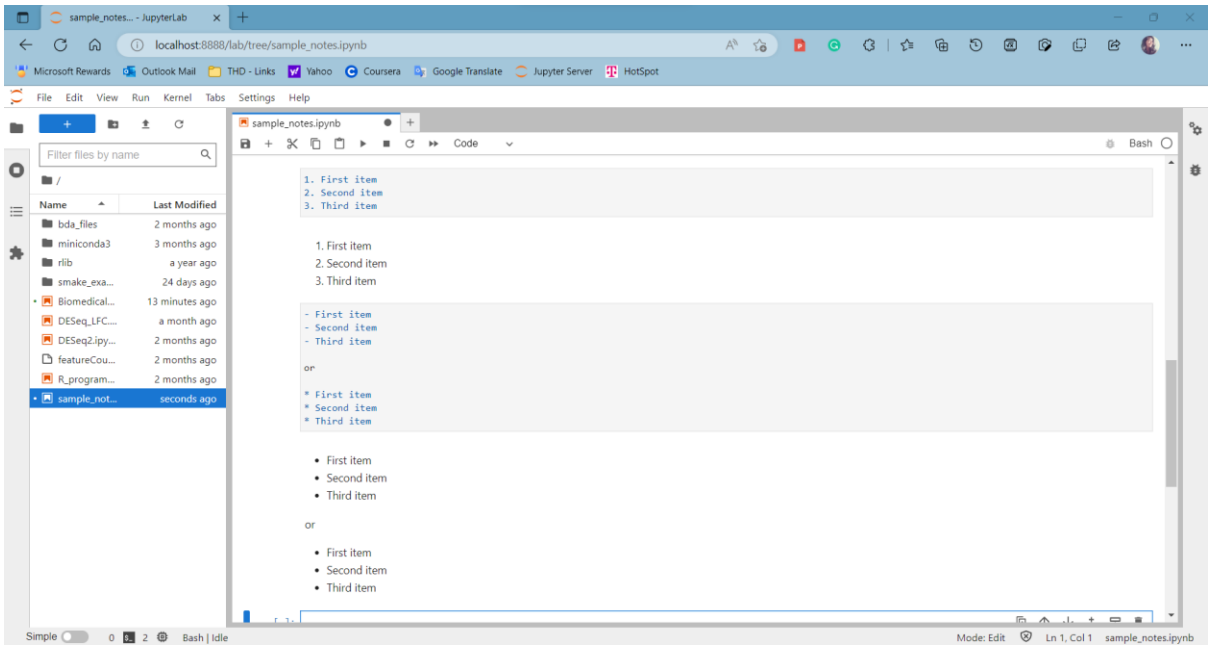


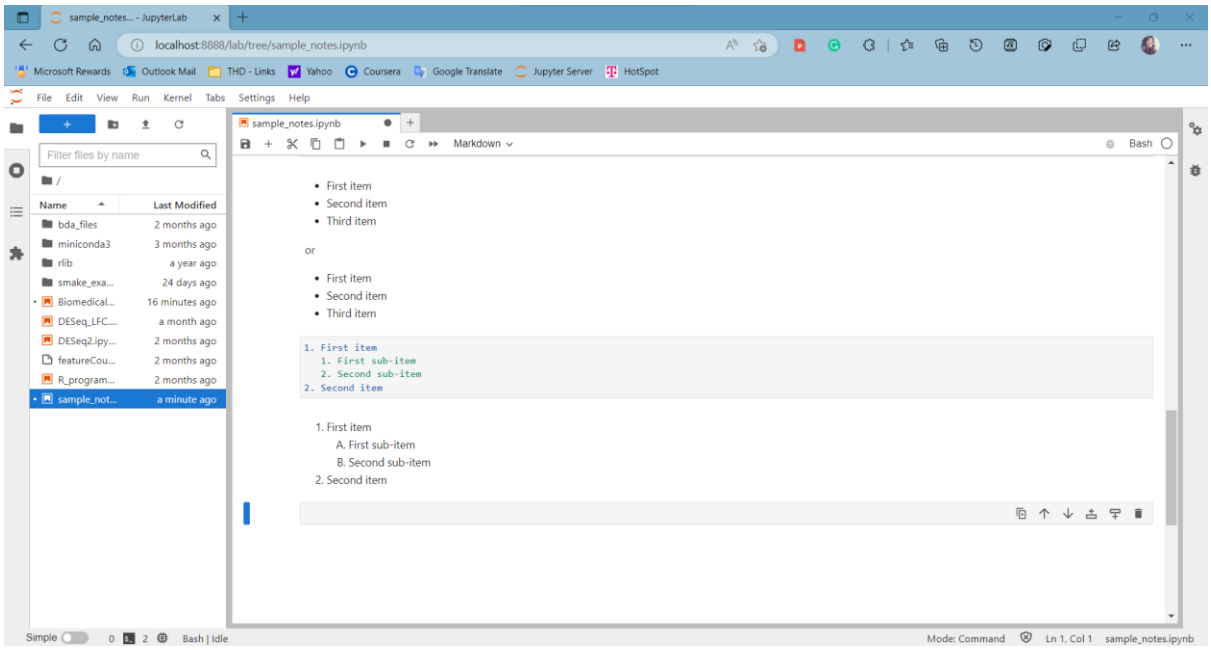
iii. How can you create numbered and unnumbered lists?

Answer:

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

- To create a numbered list, use numbers followed by periods.
- To create an unnumbered list, use hyphens or asterisks.
- Nested lists can be created by indenting items.



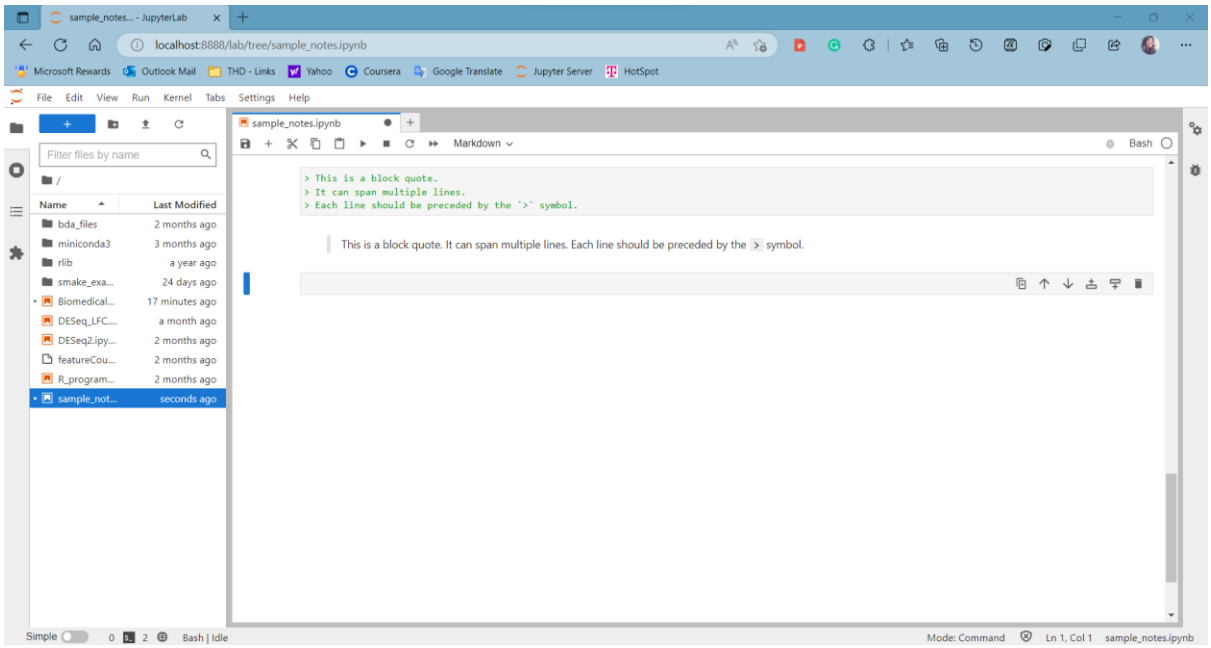


iv. How can you create a block quote?

Answer:

In markdown, a block quote can be created by using the **>** symbol before each line of the quote.

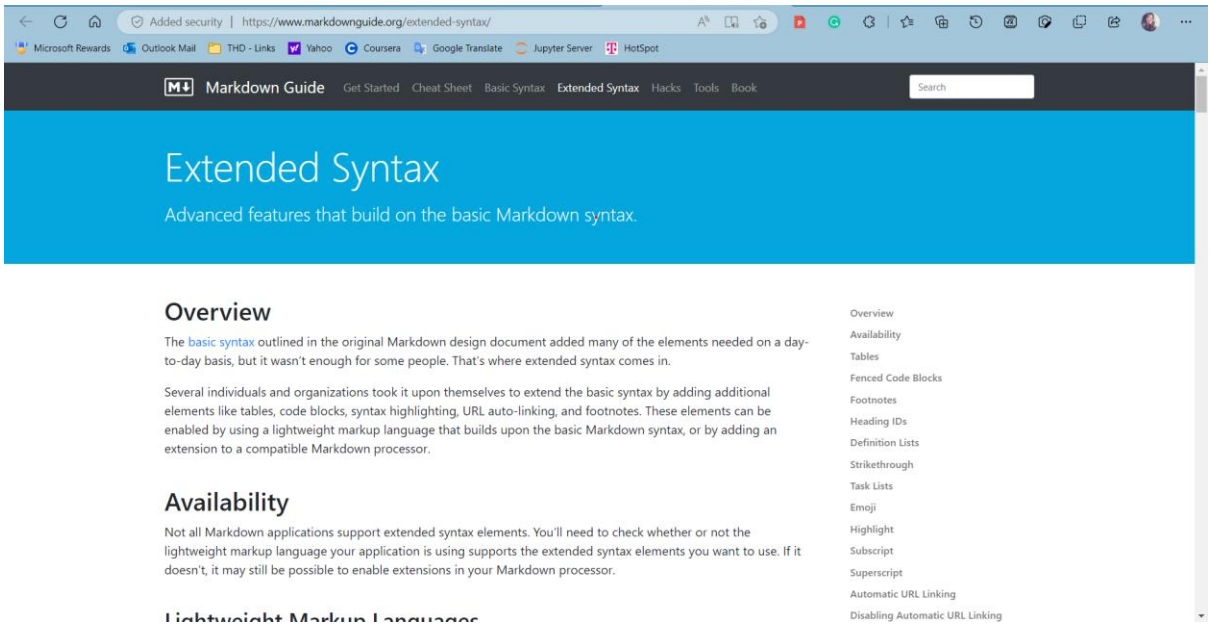
The block quote will be indented and set apart from the rest of the text in the final output, making it easy to distinguish from other text.



3. **Visit the extended markdown syntax site at:**

   https://www.markdownguide.org/extended-syntax/

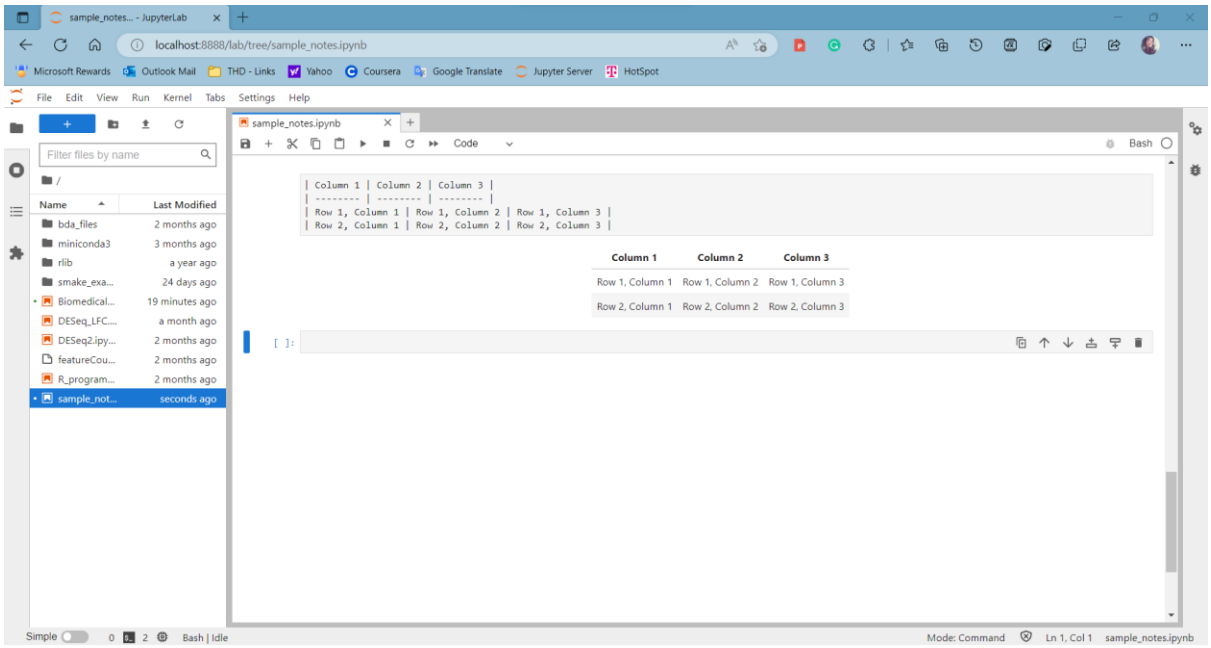
   Answer:



i. **Try to add a table using the extended markdown syntax. Can Jupyterlab render a table?**

Answer:

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.



Yes, Jupyterlab can render tables in markdown cells. When you run the cell, the table will be displayed in the rendered output.

ii. Try to create footnotes. Can Jupyterlab render footnotes as described?

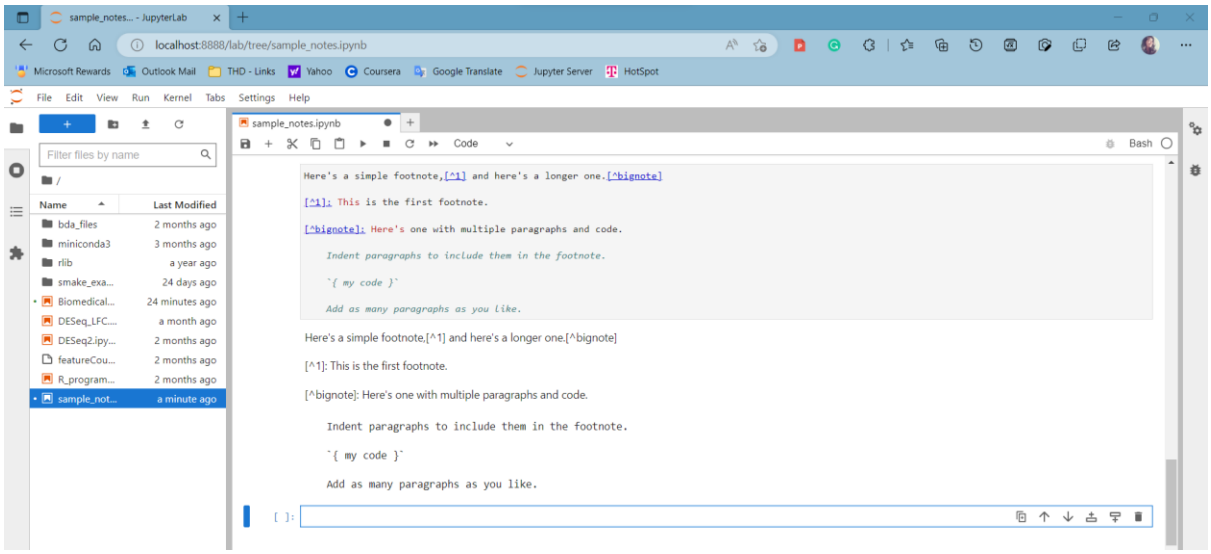
Answer:

Footnotes allow you to add notes and references without cluttering the body of the document. When you create a footnote, a superscript number with a link appears where you added the footnote reference. Readers can click the link to jump to the content of the footnote at the bottom of the page.

To create a footnote reference, add a caret and an identifier inside brackets ([^1]). Identifiers can be numbers or words, but they can't contain spaces or tabs. Identifiers only correlate the footnote reference with the footnote itself — in the output, footnotes are numbered sequentially.

Add the footnote using another caret and number inside brackets with a colon and text ([^1]: My footnote.). You don't have to put footnotes at the end of the document. You can put them anywhere except inside other elements like lists, block quotes, and tables.

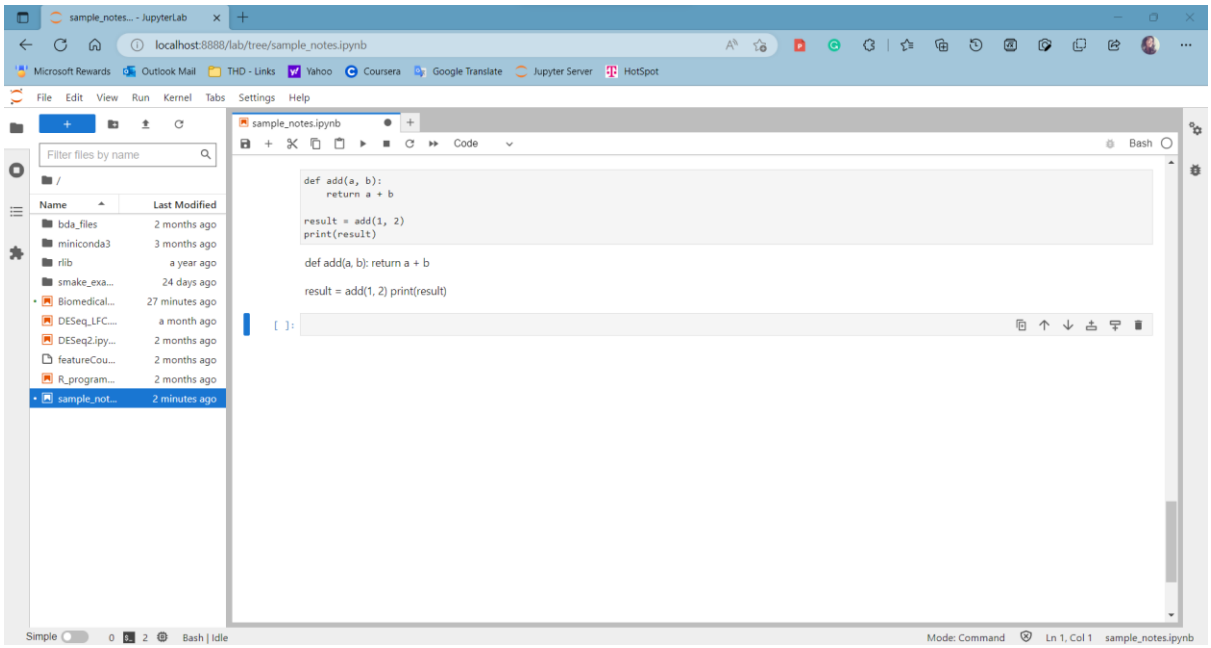
Jupyterlab does not natively support footnotes, so they will not be rendered in the final output of a markdown cell in Jupyterlab. However, some Jupyterlab extensions, such as the Jupyterlab LaTeX extension, may allow you to add footnotes using LaTeX syntax.



iii. Try the syntax highlighting blocks and write some dummy Python code. Can Jupyterlab display syntax highlighted code?

Answer:

In markdown, code blocks can be created by surrounding the code with three backticks (```) and specifying the language for syntax highlighting.



Yes, Jupyterlab can display syntax highlighted code in markdown cells. The code blocks will be displayed with the appropriate syntax highlighting and formatting, making it easier to read and understand the code. When you run the markdown cell, the code will not be executed, but will be displayed in the rendered output of the cell.

4. **Figure out how to export your current notebook to an html file.**

   Answer:

   In Jupyterlab, you can export a notebook to an HTML file by using the "File" menu and selecting "Export Notebook As...", then "Export Notebook to HTML". This will generate an HTML file that includes the markdown cells, code cells, and output of the notebook. The exported HTML file can be opened in any web browser and can be shared or used for offline viewing.

   Alternatively, you can also use the following command in the Jupyterlab terminal or command prompt:

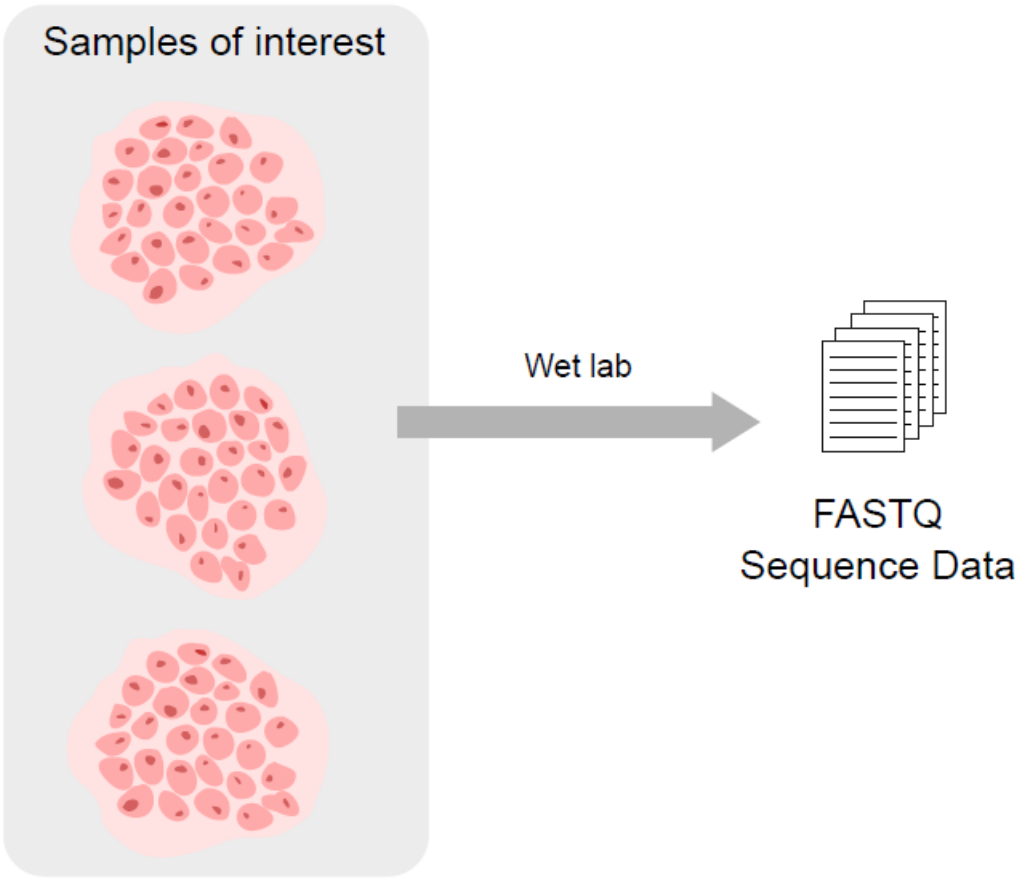
   ```
   $ jupyter nbconvert <notebook_name>.ipynb --to html
   ```

   This will generate an HTML file with the same name as the notebook, but with the **.html** extension, in the same directory as the original notebook.

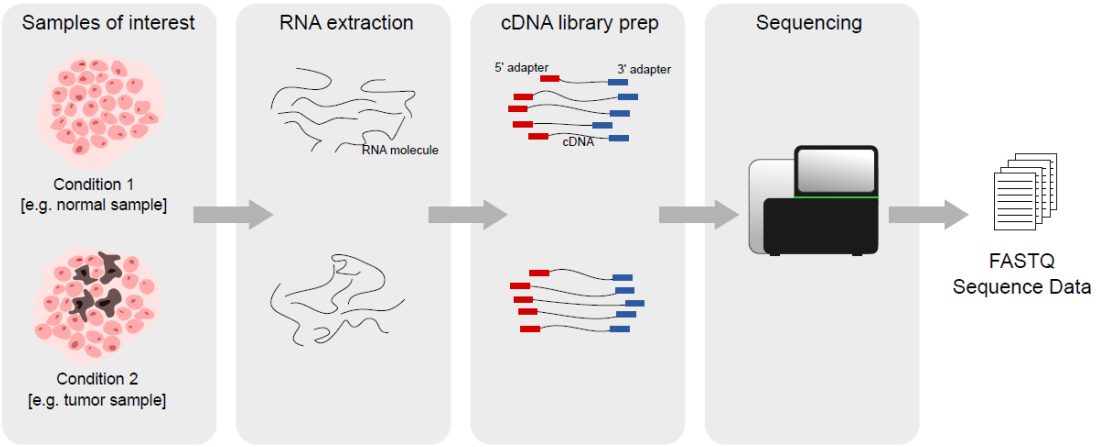
RNA-SEQ ANALYSIS : HIGH LEVEL OVERVIEW

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.).

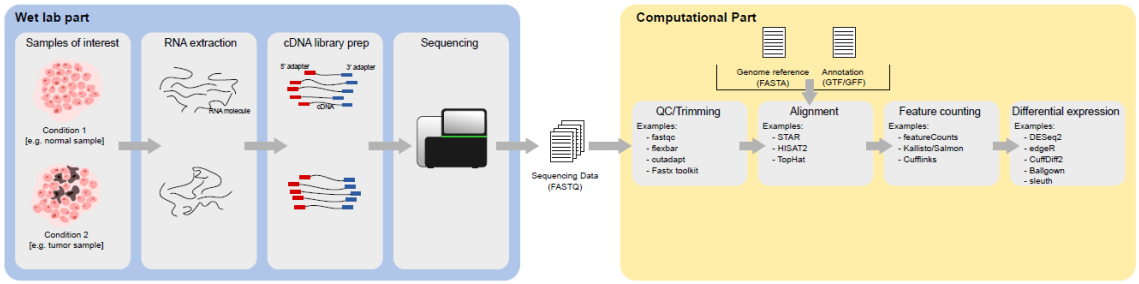


Exemplary RNA-Seq Wet Lab Part :



Note, 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.

From RNA-Seq Data To Differential Expression Analysis :



Besides sequencing-data a reference genome and the corresponding annotation is needed to perform a typical differential expression analysis.

RNA-SEQ ANALYSIS : FASTA FILE FORMAT

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.

Example fasta file is shown below –

```
>crab_anapl  ALPHA  CRYSTALLIN  B  CHAIN  (ALPHA(B)-CRYSTALLIN).
MDITIHNPLIRRPLFSWLAPSRIFDQIFGEHLQESELLPASPSLSPFLMR
SPIFRMPSWLETGLSEMRLEKDKFSVNLDVKHFSPEELKVKVLGDMVEIH
GKHEERQDEHGFIAREFNRKYRIPADVDPLTITSSLSLDGVLTVSAPRKQ
SDVPERSIPITREEKPAIAGAQRK
```

RNA-SEQ ANALYSIS : FASTQ FILE FORMAT

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. A sequence identifier/name indicated by the "**@**" sign. Often additional information can be found in the sequence identifier string.
2. The nucleotide sequence.
3. A separator line that only contains a "**+**" sign.
4. The quality scores for each nucleotide encoded in an ASCII format.

Example fastq file is shown below –

```
@HWI-ST718_146963544:8:1309:12781:23619/1
TCTGAATAAGTGCTCTTGGTATTCTGCCTTACGAGGGGACGCTGGCTACGTCTCCAAAACTT
AGGCAGGCCGTTCC
+
CCCFFFFFHHHHHJJJIJJCIIJJJJJJJJJJJJJJJJIJJJIJJJJJIJJJIHHHHHFFFF
FCEEEDBBDDBDDC
@HWI-ST718_146963544:8:2307:10477:98520/1
AAAATATGGAGGCAAGTTGAAGGCCCTGAATCATTTTCATAGTAAAAGGACAGCGGGGCAAG
AAAAGAGCACATTC
+
CC@FFFDAHHHGHHHGHIHGBBCHGHGAGIGEHIIIIEEGGC?DHIIEDHIIIIGIEDDDDD
DDDD?B<?CBDDD@
```

RNA-SEQ ANALYSIS : GTF/GFF FILE FORMAT

TASK :

Briefly describe the GTF file format. How is it specified? What is it used for?

Here are some links that might help:

- https://www.gencodegenes.org/pages/data format.html
- https://genome.ucsc.edu/FAQ/FAQformat.html

Answer:

GTF (General Transfer Format) 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 fields in a GTF file are:

1. seqname: The name of the chromosome or scaffold.
2. source: The source of the feature, such as a database or a program.
3. feature: The type of feature, such as gene, exon, or intron.
4. start: The starting position of the feature.
5. end: The ending position of the feature.
6. score: A score for the feature, typically set to ".".
7. strand: The strand of the feature, either "+" or "-".
8. frame: The frame of the feature, set to "0", "1", or "2".
9. attribute: Additional information about the feature, such as gene names or transcript identifiers.

GTF files are used in a variety of applications, including gene expression analysis, comparative genomics, and genome assembly validation. They are supported by many popular tools in the bioinformatics community, including the UCSC Genome Browser, the Ensembl genome browser, and the GENCODE database of gene annotations.

The figure below shows 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
```

RNA-SEQ ANALYSIS : OBTAINING DATA AND QUALITY CONTROL

TASK :

1. **Find out what you can do with the Linux command line program *wget*.**

   Answer:

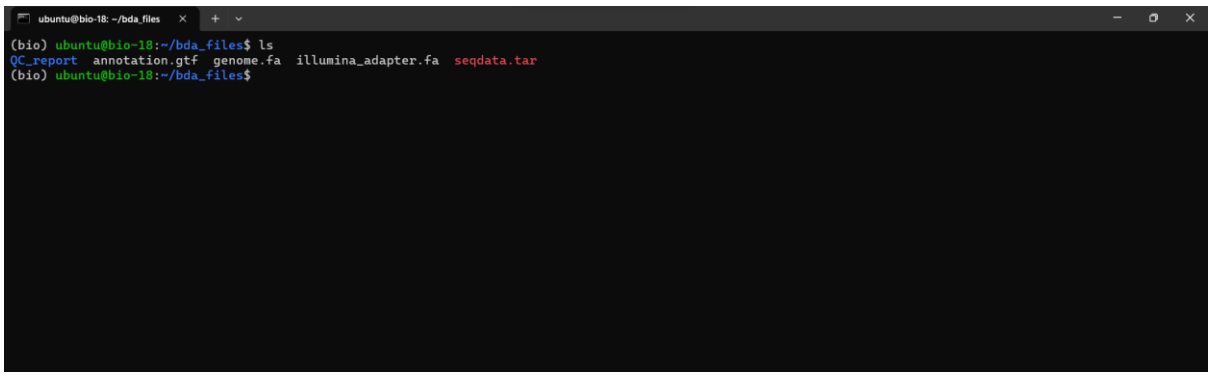
   The ***wget*** command retrieves content from web servers and used to download the data for the workflow.

2. **Download the following files to your remote machine:**

   - 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

   Answer:

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



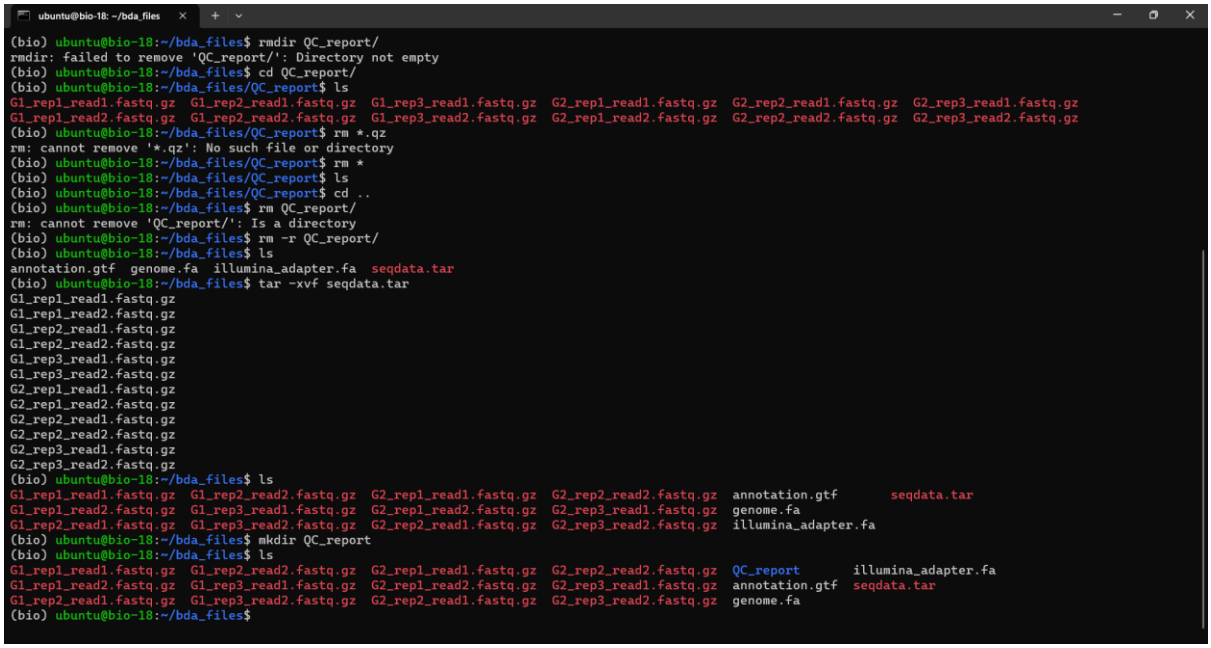
3. **Find out how to unpack the seqdata.tar. Describe what tar does and which options are needed to unpack the sequencing data. Note, you should get 12 files out of your seqdata.tar file. The 12 files will end with ".fastq.gz"**

Answer:

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.

RNA-SEQ ANALYSIS : QUALITY CONTROL

To determine quality of the raw sequencing data, quality control is performed.

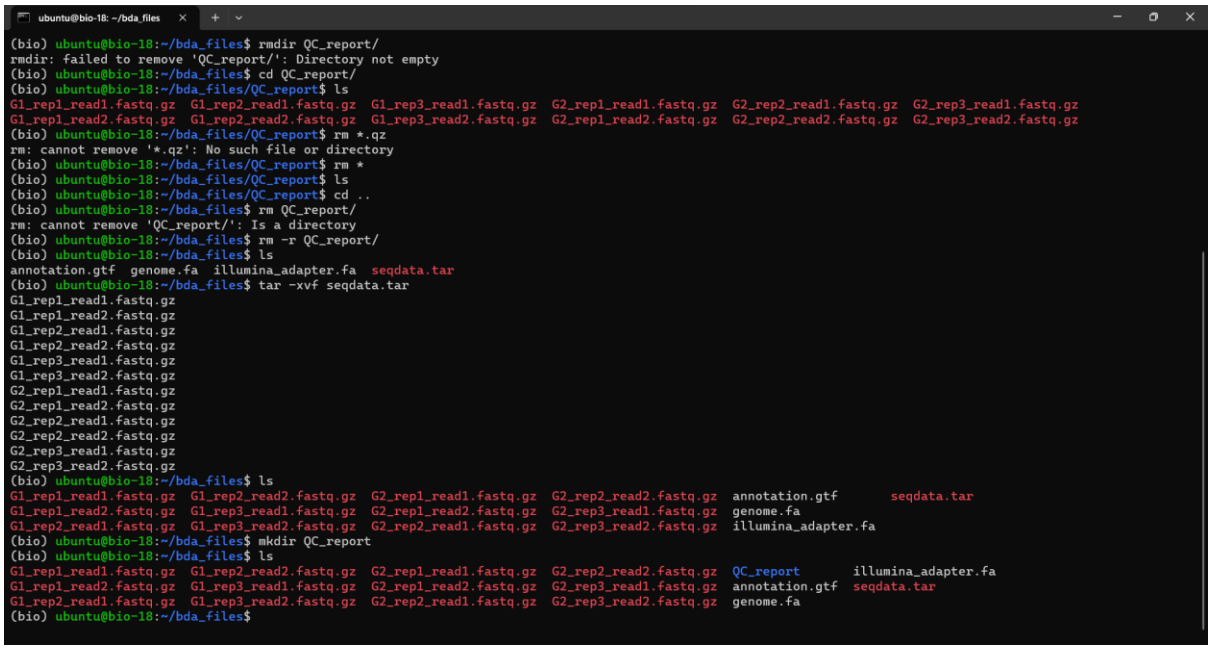
TASKS :

1. **Create a new folder to store your QC reports.**

   Answer:

   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`



2. **Run fastqc on the sequencing data.**

   i. **Have a look at the *fastqc* QC reports. Familiarize with the output of the *fastqc* program and understand what is reported. Why is quality control performed on raw sequencing data?**

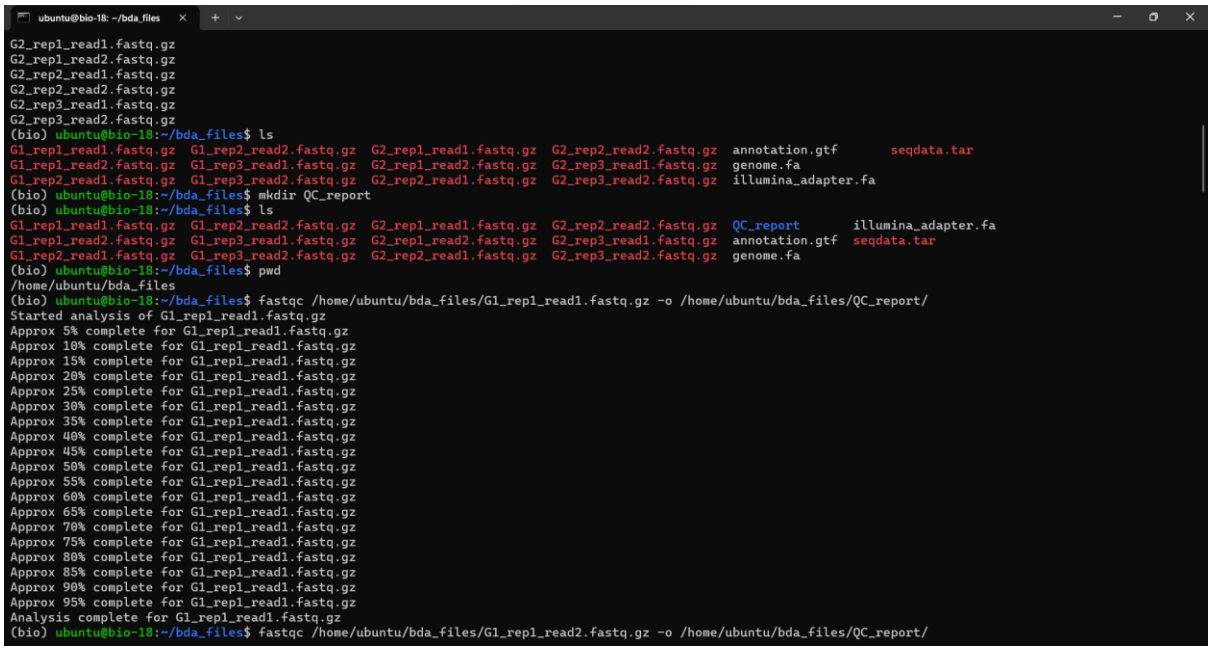
   Answer:

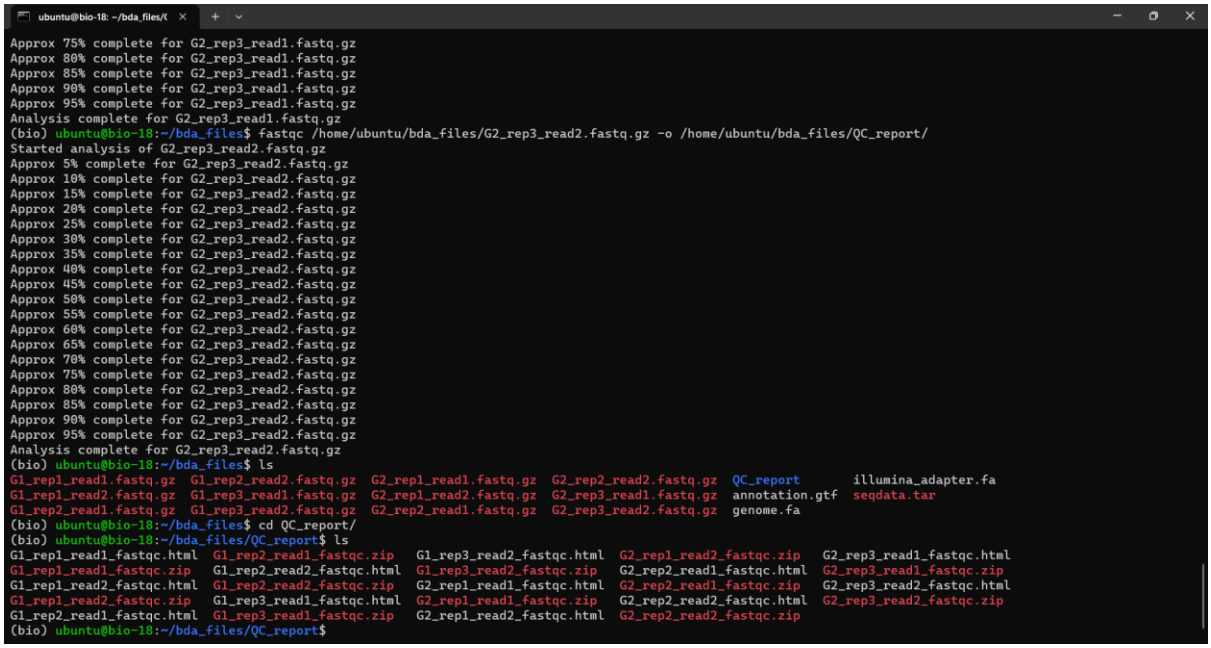
   The following command is performed to obtain the QC report:

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

   **** same command is used for all the other files**

   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. This frequently entails significant time savings during further analysis.

ii. Have a closer look at the "Per base sequence quality section". What are your observations?

Answer:

For per sequence base for one of the read of the data:

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.

Remove Adapter Sequences : Adapter Trimming With *flexbar*

Data from high-throughput sequencing are efficiently preprocessed using the application *Flexbar*. Barcoded runs are demultiplexed, and adapter sequences are eliminated. 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.

TASKS :

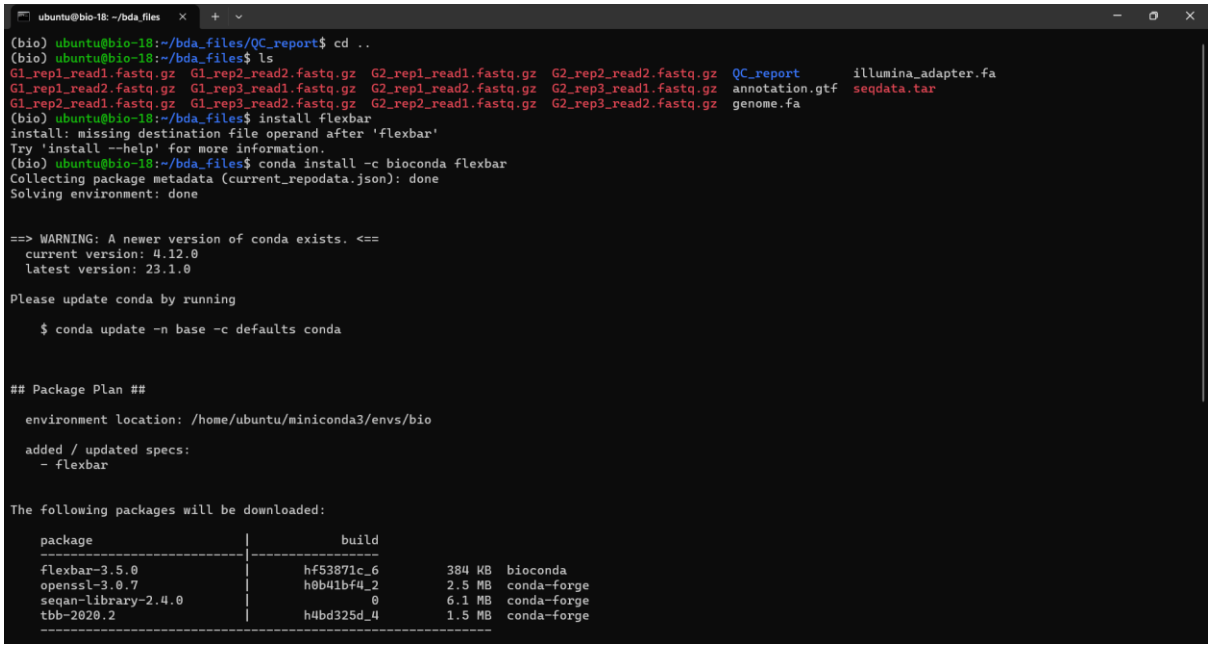
1. **Find the documentation for *flexbar* and install *flexbar* via conda.**

   Answer:

   GitHub - seqan/flexbar: flexible barcode and adapter removal

   Flexbar is installed in the **bio** environment of conda using the following command:

   ```
   $ conda install -c bioconda flexbar
   ```



2. Create small examples to understand how *flexbar* removes adapter sequences:

i. Create a *fasta* file that contains 2 adapter sequences and a *fastq* file that contains a few test sequences. Some of your sequences should contain one of your adapter sequences either at the beginning, in the middle or at the end of your sequence.

Answer the following questions:

a. Are the adapters removed?

Answer:

No, the adapters were not removed.

b. Are partial adapter matches removed and is it important where the adapter sequence occurs?

Answer:

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.

c. **Is anything else removed?**

   Answer:

   No.

d. **Is the program searching for the adapter only or also for reverse or reverse complement versions as well?**

   Answer:

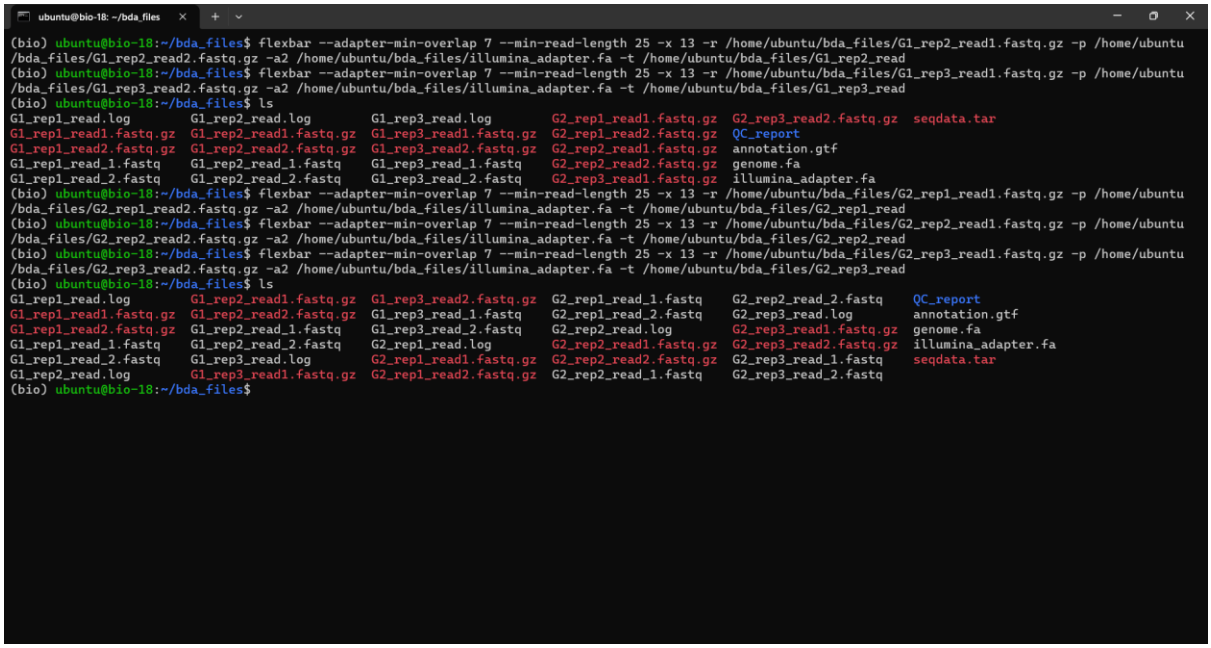
   Only for adapter.

TASK :

Apply *flexbar* to the downloaded paired-end sequencing data.

Notes:

- `$ flexbar -h` is your friend.
- You might want to create a separate directory to keep your work structured and find the flexbar option to save your results into a specified directory.
- Use the "illumina adapter.fa" file as adapter file.
- How can you use *flexbar* for paired-end data? (Hint: you need to provide read1 and read2 as input)
- Add the option to remove the first 13 bases from the reads.
- Your sequence data is in a compressed .gz format. flexbar can take .gz files as input. Find the *flexbar* option to directly store the trimmed sequence data in a .gz compressed format.
- Use the following options in addition to the ones mentioned above:
  `--adapter-min-overlap 7 - -min-read-length 25 - -threads 1`



The options specified in this command are as follows:

- **--adapter-min-overlap 7**: Specifies the minimum overlap length between reads and adapters for them to be considered as adapter-trimmed.
- **--min-read-length 25**: Specifies the minimum length a read must have after trimming to be kept for further analysis.
- **-threads 1**: Specifies 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:** Specifies the location and name of the first read file (read1) in the Fastq format.
- **-p /home/ubuntu/bda_files/G1_rep1_read2.fastq.gz:** Specifies the location and name of the second read file (read2) in the Fastq format.
- **-a2 /home/ubuntu/bda_files/illumina_adapter.fa:** Specifies the location and name of the adapter file in Fasta format.
- **-t /home/ubuntu/bda_files/"sample_name":** Specifies the location and prefix for the output files as it will generate two output files 1 and 2.

RNA-SEQ ANALYSIS : READ ALIGNMENT

Alignment And Quantification (part I) : Align NGS Reads Using STAR

The **Spliced Transcripts Alignment to a Reference (STAR) program** will be used to map the trimmed and quality-controlled RNA-Seq reads to the human genome (transcriptome).

STAR is a software tool for aligning RNA-Seq data to a reference genome. It is widely used in transcriptome analysis due to its fast speed, high sensitivity, and ability to handle alternative splicing events.
In STAR, the RNA-Seq reads are first transformed into short reads, called "seeds", which are then aligned to the reference genome. STAR then extends these alignments to full-length reads, taking into account alternative splicing events. The alignment results are used to construct a comprehensive picture of the transcriptome, including gene expression levels, alternative splice forms, and gene fusion events.
STAR also supports the use of a genome-guided mode, in which transcriptome information is used to guide the alignment of RNA-Seq reads to the reference genome, leading to improved accuracy and sensitivity.

TASK :

1. **Go the github page of STAR (https://github.com/alexdobin/STAR) and download the current manual.**

   Answer:

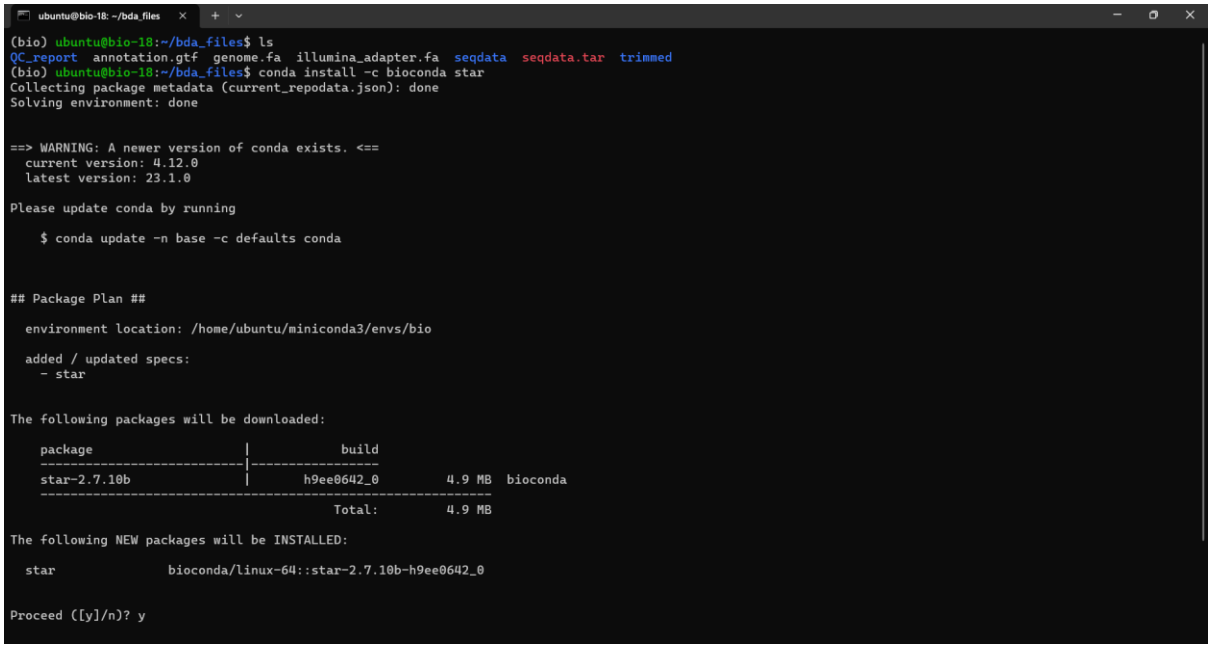
   STAR/STARmanual.pdf at master · alexdobin/STAR · GitHub

2. **Install STAR via conda to your "bio" environment.**

   Answer:

   In order to install the STAR into the **bio** environment, following command is used:

   `$ conda install -c bioconda star`



3. **Start reading the manual at section 1.2 and solve the following tasks by using the STAR manual:**

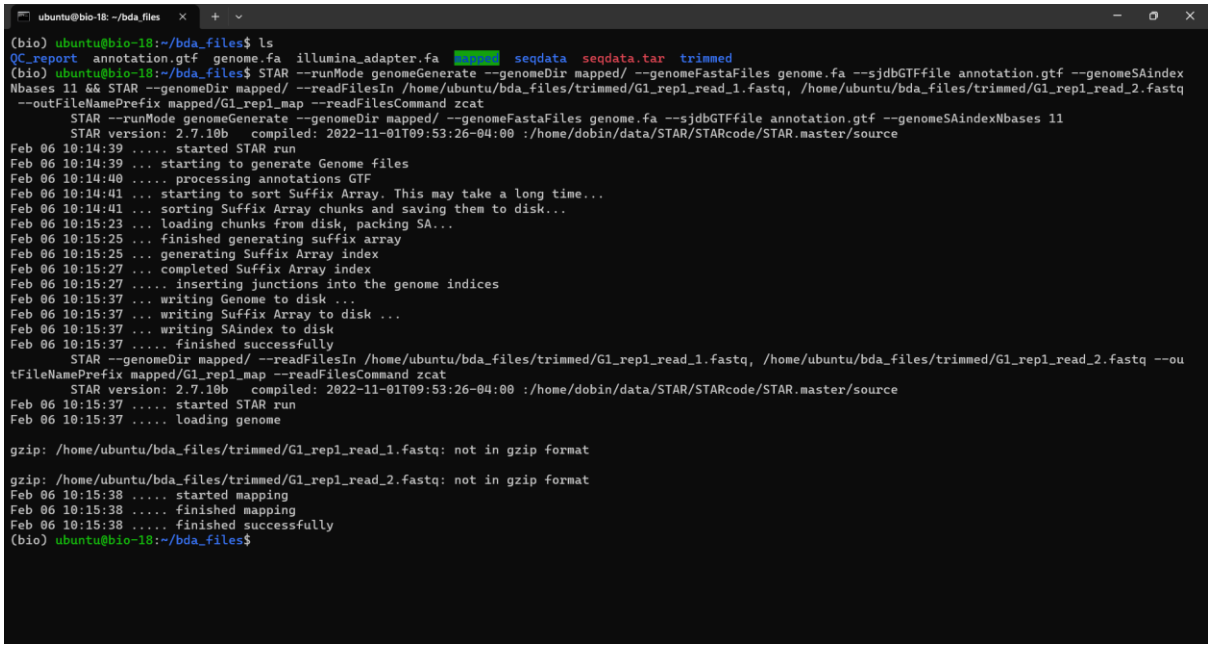
   i. **Generate an index for STAR using your genome *fasta* file and the corresponding annotation file. Use the following option in addition:**

   `--genomeSAindexNbases 11`

   Answer:

   After installation, a new directory to store the alignment file is made and the trimmed reads generated at the last step are aligned with the reference genome using the 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:** This option specifies 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/:** This option specifies the directory where the genome index will be stored.

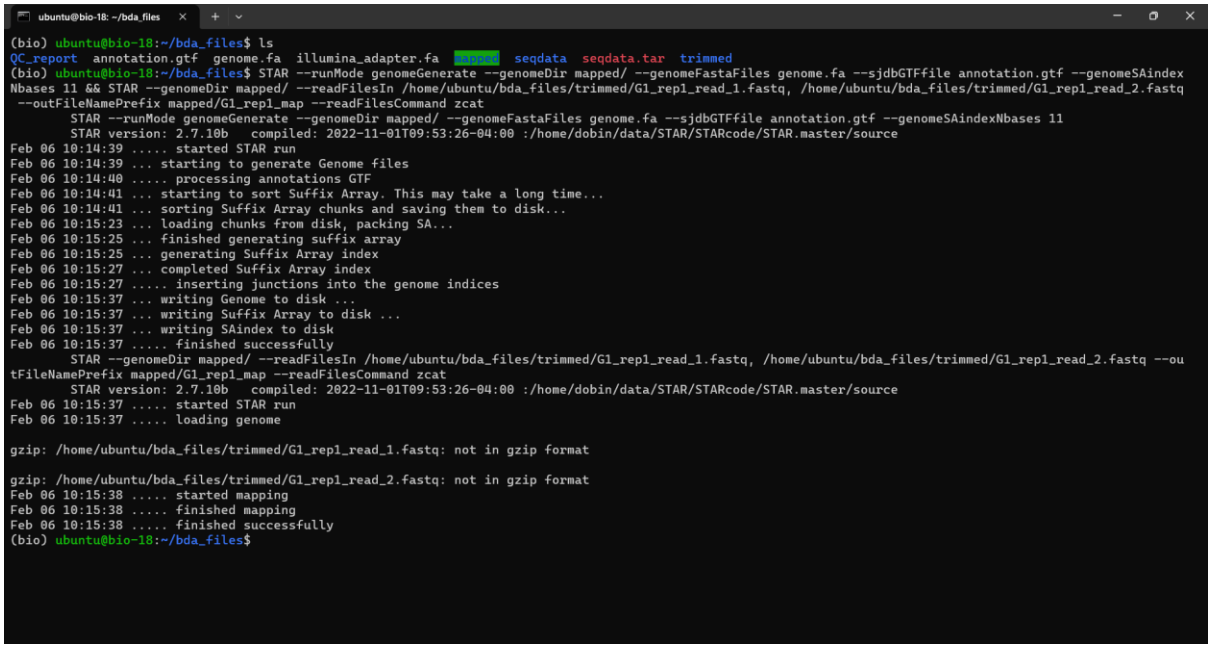
- **--genomeFastaFiles /home/ubuntu/bda_files/genome.fa:** This option specifies the FASTA file of the reference genome.
- **--sjdbGTFfile /home/ubuntu/ bda_files /annotation.gtf:** This option specifies the GTF (Gene Transfer Format) file of the annotation.
- **--genomeSAindexNbases 11:** This option specifies the number of bases for constructing the suffix array index of the genome.
- **--genomeDir /home/ubuntu/bda_files/mapped/:** This option specifies the directory where the previously generated genome index is stored.
- **--readFilesIn:** This option specifies the paired-end RNA-seq read files that will be aligned to the reference genome.
- **--outFileNamePrefix: /home/ubuntu/bda_files/mapped/G1_rep1_map:** This option specifies the prefix for the output file names.

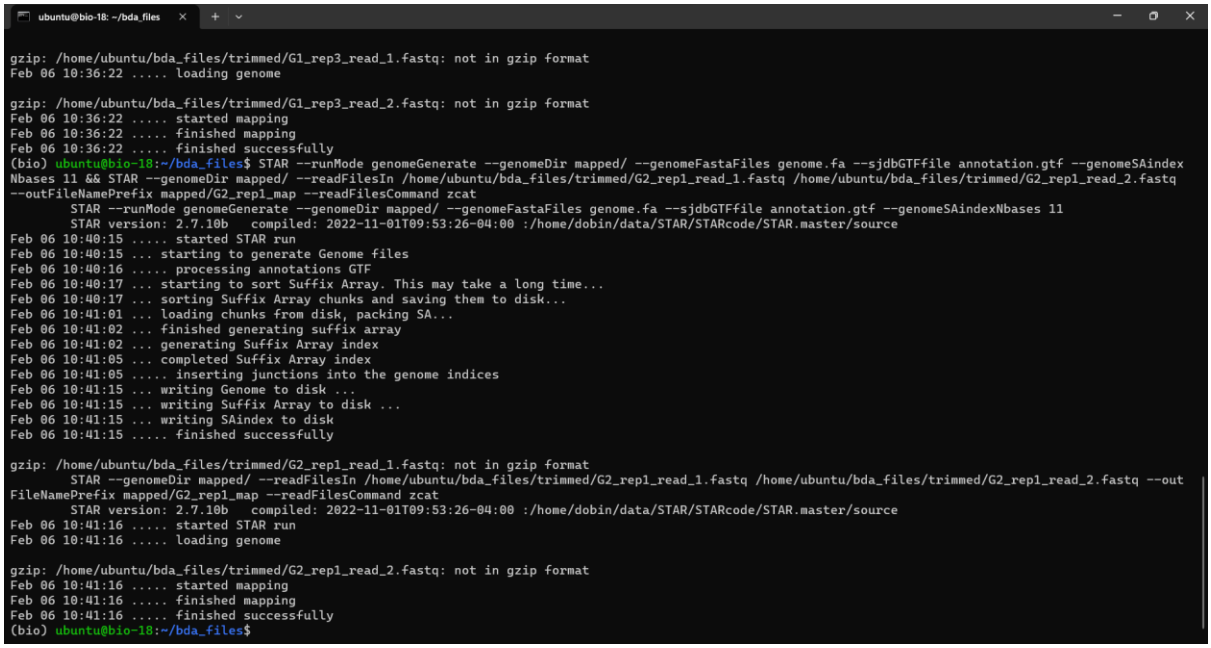
ii. **Use STAR to map the paired-end reads for each sample. Use the following options in addition:**

`--outFileNamePrefix /ouput/path/prefix`

`--readFilesCommand zcat`

Answer:





STAR OUTPUT FILES :

STAR produces multiple output files. All output files have a standard name, but the prefix can be changed via the

`--outFileNamePrefix /output/path/prefix` option.

A detailed description of all output files can be found at section 4 of the STAR manual. The **Log.final.out** as well as the **Aligned.out.sam** output file are shown here in more detail.

Example of a **Log.final.out** file:

The final log file contains summary statistics about the mapping job and is typically used for quality control.

For instance:

- to check the percentage of uniquely mapped reads.
- to check the percentage of unmapped reads.

SAM File Format :

Sequence Alignment/Map format, or SAM for short, is the usual output format used by NGS-read alignment algorithms. An alignment section and an optional header section make up a SAM file. The alignment section, which has 11 necessary columns and a TAB-delimited style, offers comprehensive alignment information.

Mandatory fields of the alignment section:

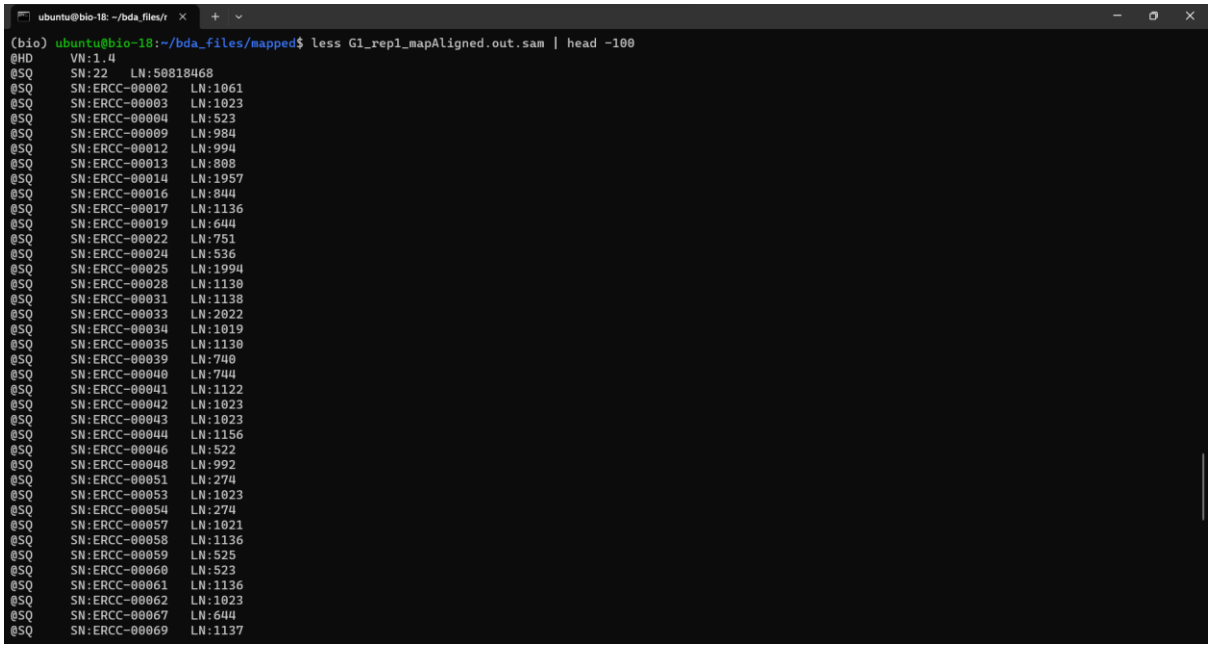
1	QNAME	String	Query template name
2	FLAG	Int	Bitwise flag
3	RNAME	String	Name of the reference sequence
4	POS	Int	1-based leftmost mapping position
5	MAPQ	Int	Mapping quality
6	CIGAR	String	Detailed alignment information
7	RNEXT	String	Name of the mate read
8	PNEXT	Int	Position of the mate
9	TLEN	Int	Length
10	SEQ	String	Query sequence
11	QUAL	String	ASCII Phred scores

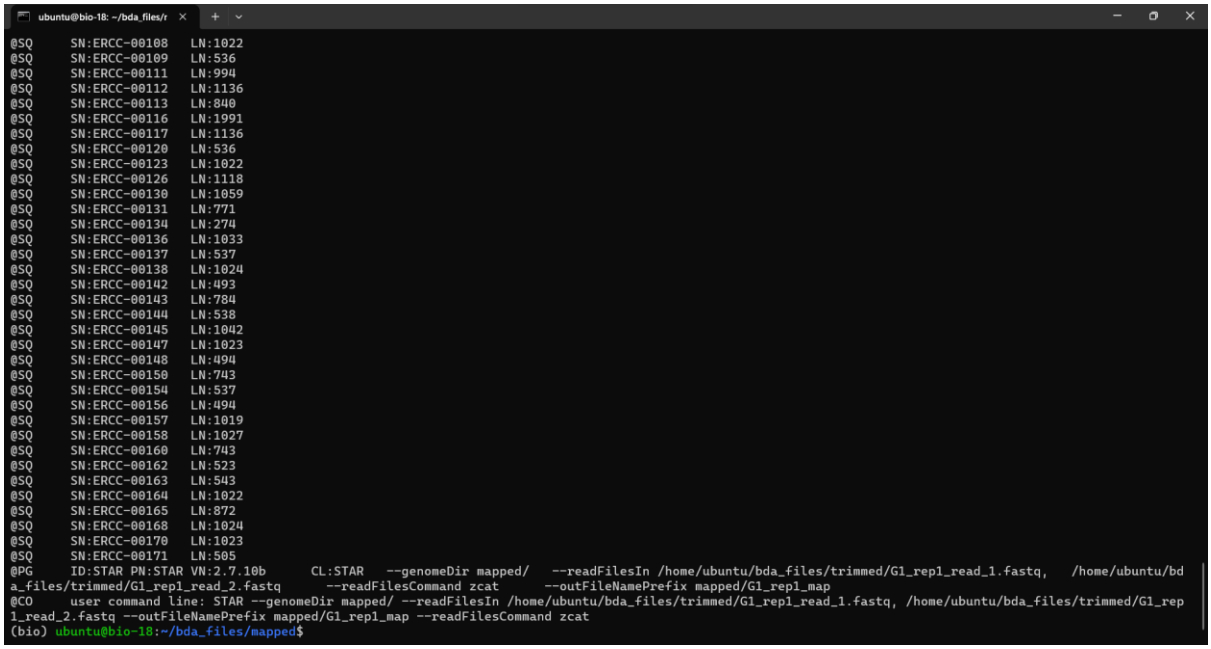
Optional 12th column with optional fields in TAG:TYPE:VALUE format

SAM format specification:	https://samtools.github.io/hts-specs/SAMv1.pdf
SAM optional field specification:	https://github.com/samtools/hts-specs/blob/master/SAMtags.pdf
Translating bitwise FLAGs:	http://broadinstitute.github.io/picard/explain-flags.html

The following command shows the format of a SAM file by running one of the SAM file generated in the previous step.

```
$ less G1_rep1_mapAligned.out.sam | head -100
```


ubuntu@bio-18: ~/bda_files/r
(bio) ubuntu@bio-18:~/bda_files/mapped$ less G1_rep1_mapAligned.out.sam | head -100
@HD VN:1.4
@SQ SN:22 LN:50818468
@SQ SN:ERCC-00002 LN:1061
@SQ SN:ERCC-00003 LN:1023
@SQ SN:ERCC-00004 LN:523
@SQ SN:ERCC-00009 LN:984
@SQ SN:ERCC-00012 LN:994
@SQ SN:ERCC-00013 LN:808
@SQ SN:ERCC-00014 LN:1957
@SQ SN:ERCC-00016 LN:844
@SQ SN:ERCC-00017 LN:1136
@SQ SN:ERCC-00019 LN:644
@SQ SN:ERCC-00022 LN:751
@SQ SN:ERCC-00024 LN:536
@SQ SN:ERCC-00025 LN:1994
@SQ SN:ERCC-00028 LN:1130
@SQ SN:ERCC-00031 LN:1138
@SQ SN:ERCC-00033 LN:2022
@SQ SN:ERCC-00034 LN:1019
@SQ SN:ERCC-00035 LN:1130
@SQ SN:ERCC-00039 LN:740
@SQ SN:ERCC-00040 LN:744
@SQ SN:ERCC-00041 LN:1122
@SQ SN:ERCC-00042 LN:1023
@SQ SN:ERCC-00043 LN:1023
@SQ SN:ERCC-00044 LN:1156
@SQ SN:ERCC-00046 LN:522
@SQ SN:ERCC-00048 LN:992
@SQ SN:ERCC-00051 LN:274
@SQ SN:ERCC-00053 LN:1023
@SQ SN:ERCC-00054 LN:274
@SQ SN:ERCC-00057 LN:1021
@SQ SN:ERCC-00058 LN:1136
@SQ SN:ERCC-00059 LN:525
@SQ SN:ERCC-00060 LN:523
@SQ SN:ERCC-00061 LN:1136
@SQ SN:ERCC-00062 LN:1023
@SQ SN:ERCC-00067 LN:644
@SQ SN:ERCC-00069 LN:1137


ubuntu@bio-18: ~/bda_files/r
@SQ SN:ERCC-00108 LN:1022
@SQ SN:ERCC-00109 LN:536
@SQ SN:ERCC-00111 LN:994
@SQ SN:ERCC-00112 LN:1136
@SQ SN:ERCC-00113 LN:840
@SQ SN:ERCC-00116 LN:1991
@SQ SN:ERCC-00117 LN:1136
@SQ SN:ERCC-00120 LN:536
@SQ SN:ERCC-00123 LN:1022
@SQ SN:ERCC-00126 LN:1118
@SQ SN:ERCC-00130 LN:1059
@SQ SN:ERCC-00131 LN:771
@SQ SN:ERCC-00134 LN:274
@SQ SN:ERCC-00136 LN:1033
@SQ SN:ERCC-00137 LN:537
@SQ SN:ERCC-00138 LN:1024
@SQ SN:ERCC-00142 LN:493
@SQ SN:ERCC-00143 LN:784
@SQ SN:ERCC-00144 LN:538
@SQ SN:ERCC-00145 LN:1042
@SQ SN:ERCC-00147 LN:1023
@SQ SN:ERCC-00148 LN:494
@SQ SN:ERCC-00150 LN:743
@SQ SN:ERCC-00154 LN:537
@SQ SN:ERCC-00156 LN:494
@SQ SN:ERCC-00157 LN:1019
@SQ SN:ERCC-00158 LN:1027
@SQ SN:ERCC-00160 LN:743
@SQ SN:ERCC-00162 LN:523
@SQ SN:ERCC-00163 LN:543
@SQ SN:ERCC-00164 LN:1022
@SQ SN:ERCC-00165 LN:872
@SQ SN:ERCC-00168 LN:1024
@SQ SN:ERCC-00170 LN:1023
@SQ SN:ERCC-00171 LN:505
@PG ID:STAR PN:STAR VN:2.7.10b CL:STAR --genomeDir mapped/ --readFilesIn /home/ubuntu/bda_files/trimmed/G1_rep1_read_1.fastq, /home/ubuntu/bda_files/trimmed/G1_rep1_read_2.fastq --readFilesCommand zcat --outFileNamePrefix mapped/G1_rep1_map
@CO user command line: STAR --genomeDir mapped/ --readFilesIn /home/ubuntu/bda_files/trimmed/G1_rep1_read_1.fastq, /home/ubuntu/bda_files/trimmed/G1_rep1_read_2.fastq --outFileNamePrefix mapped/G1_rep1_map --readFilesCommand zcat
(bio) ubuntu@bio-18:~/bda_files/mapped$

RNA-SEQ ANALYSIS : FEATURE COUNTING

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.

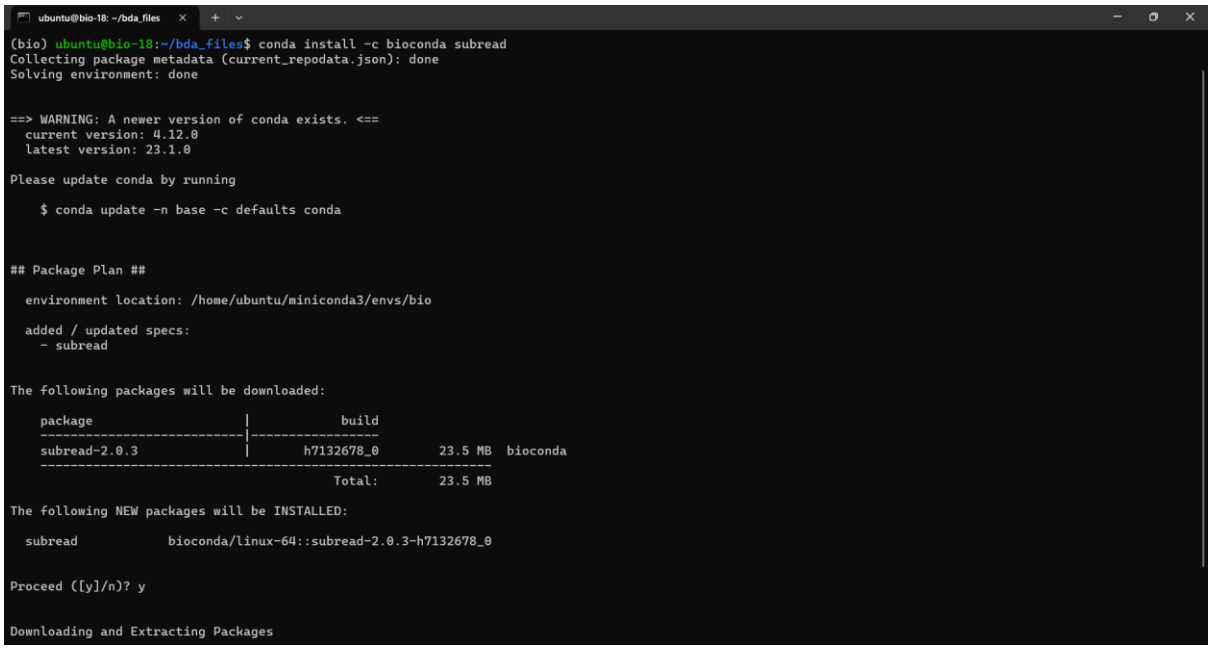
TASKS :

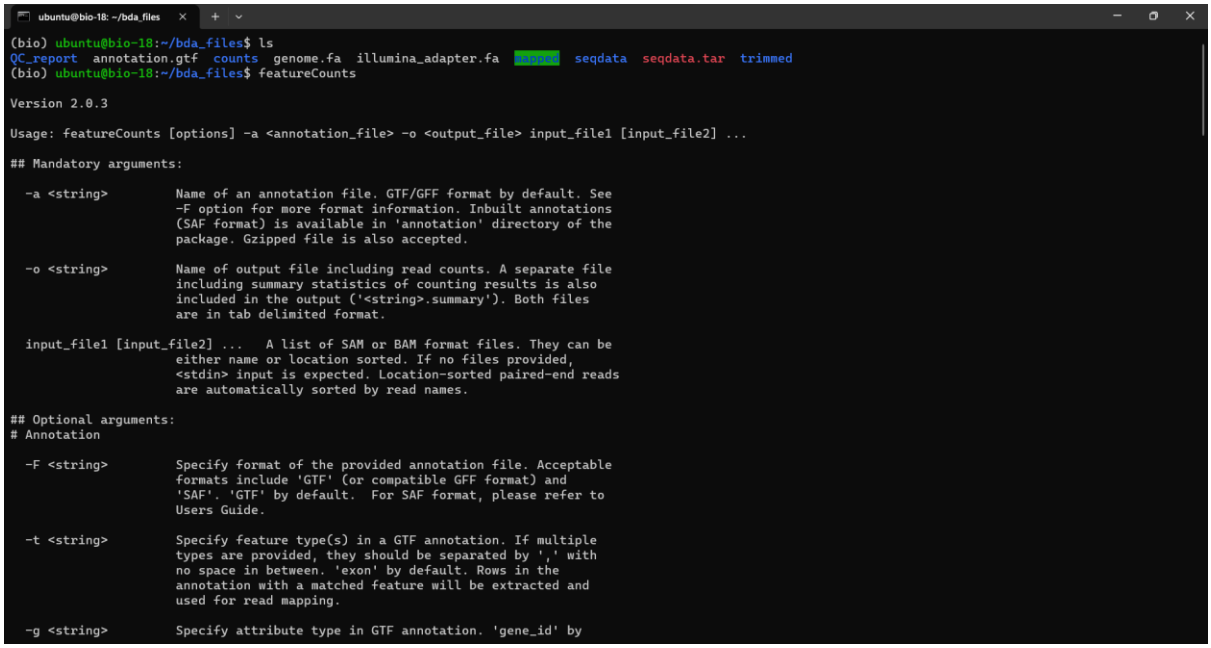
1. **Find out which conda package includes the *featureCounts* program and install the package via conda into your "bio" environment. (Hint: To test if your installation was successful you can type in *featureCounts* into your command line. You should see the help page of the featureCounts program if you found the right package and installed it successfully).**

   Answer:

   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
   ```





2. **Apply *featureCounts* to your STAR alignment output files to count reads per gene. Find out which options you need to address the following:**
   i. ***featureCounts* shall use the gene name of the GTF annotations as identifier. Find out which option is needed to use the gene names used in a GTF annotation file.**

      Answer:

      The **-a** options specify the annotation file for the genomic features to be counted.

ii. **Since we have paired-end data, find the option that is needed to specify that fragments (both read pairs) rather than reads are counted.**

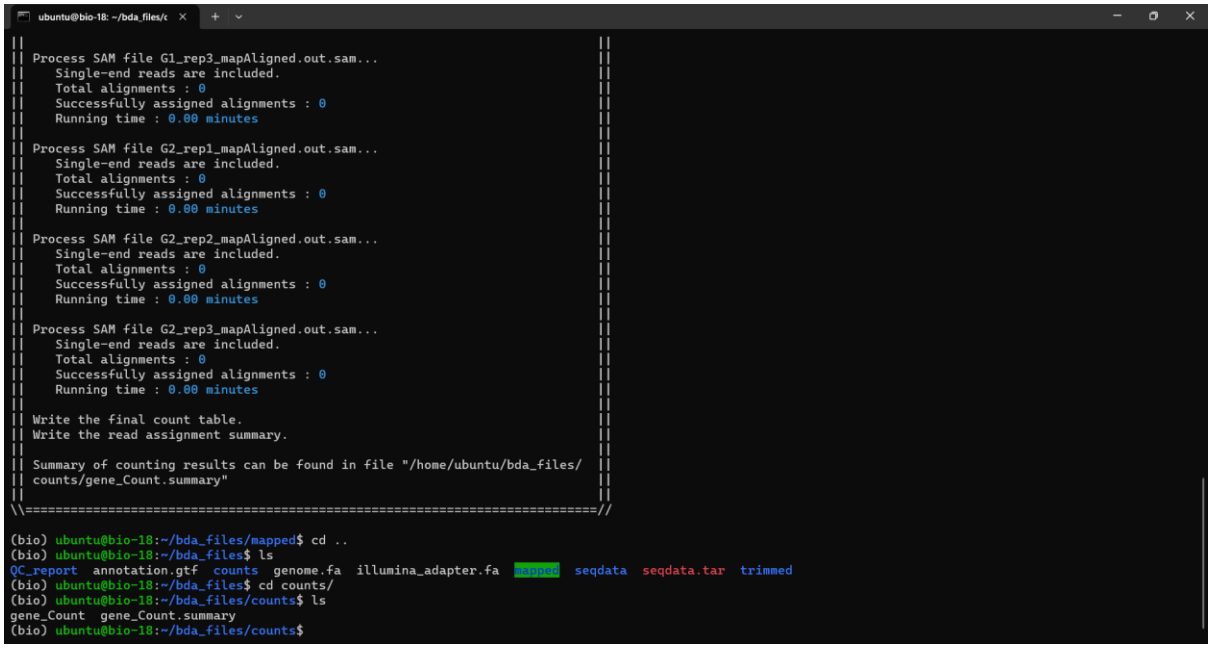
Answer:

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, following command is used:

Parameters specific to paired end reads

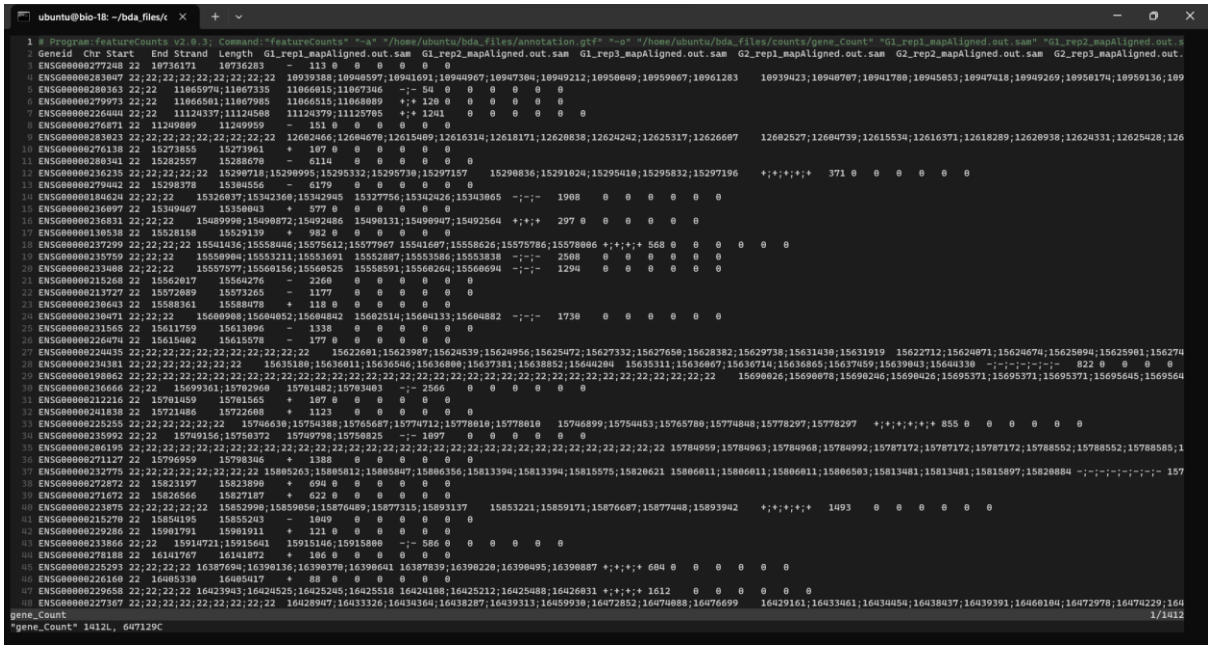
-p : Specify that input data contain paired-end reads. To perform fragment counting (i.e. counting read pairs), the '`--countReadPairs`' parameter should also be specified in addition to this parameter.



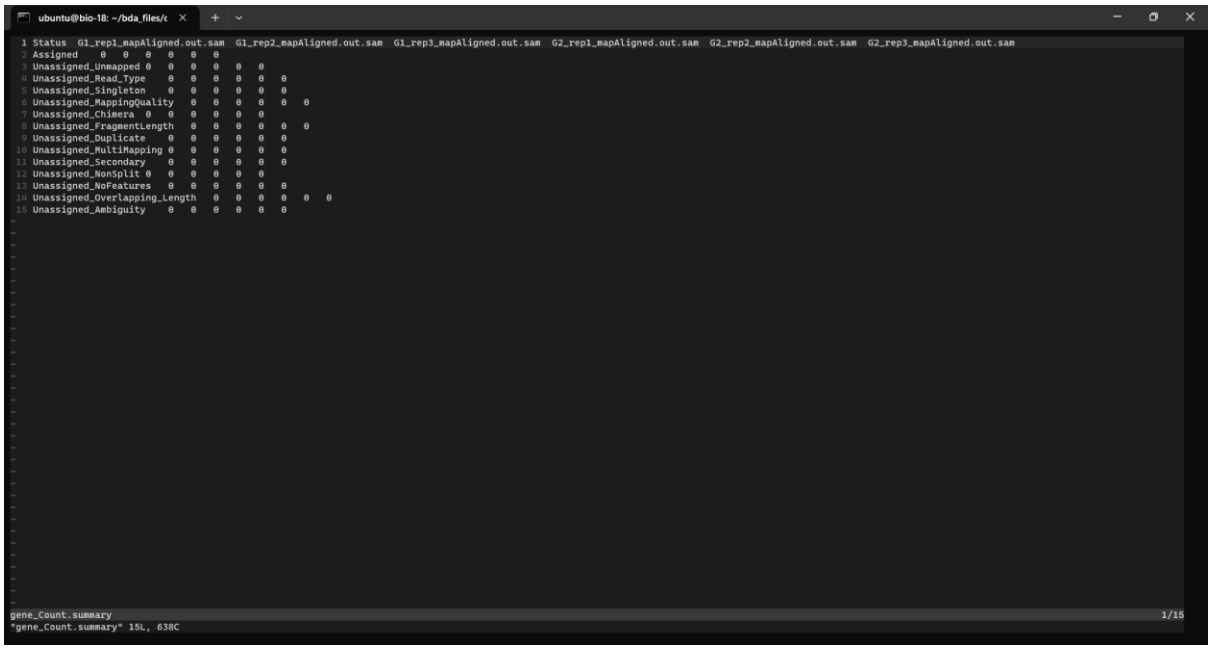
The **-o** option on the other hand specify the directory in which the output files need to be generated.

OUTPUT FILES :

File : `gene_counts`



File : `gene_counts.txt.summary`



THE R PROGRAMMING LANGUAGE :

R is a freely available language and environment for statistical computing and graphics - [https://cran.r-project.org].

- R is free and open source.
- R is a popular language for data analysis and statistics.
- R is an interpreted language that makes working with data more comfortable.
- Together with Python, R is the de-facto standard for data analysis. From my personal experience a useful rule of thumb is that the more a company/group focuses on statistics the more they are in favor of R.
- Experience in both languages is helpful in the _eld of data analysis. Often, a package/method needed for an analysis is only available in one of the languages (i.e. DESeq2).
- Likewise to Python, R is used in notebooks together with markdown to make reproducible data analysis possible.
- Thousands of packages are available to R that extend the language with specific analysis options and visualization features.

TASK :

1. Familiarize with R:
    i. Read the data from the featureCounts output.txt _le.
    ii. Extract the gene count data and put it into a separate data frame.
    iii. Compute the mean for each gene
    (Hint: have a look at the function rowMeans()).
    iv. Compute the standard deviation for each gene.
    v. Only keep the means and corresponding standard deviations where both the mean and the sd are > 0.
    vi. Generate a scatter plot of the computed means and standard deviations.

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;109407
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)
```

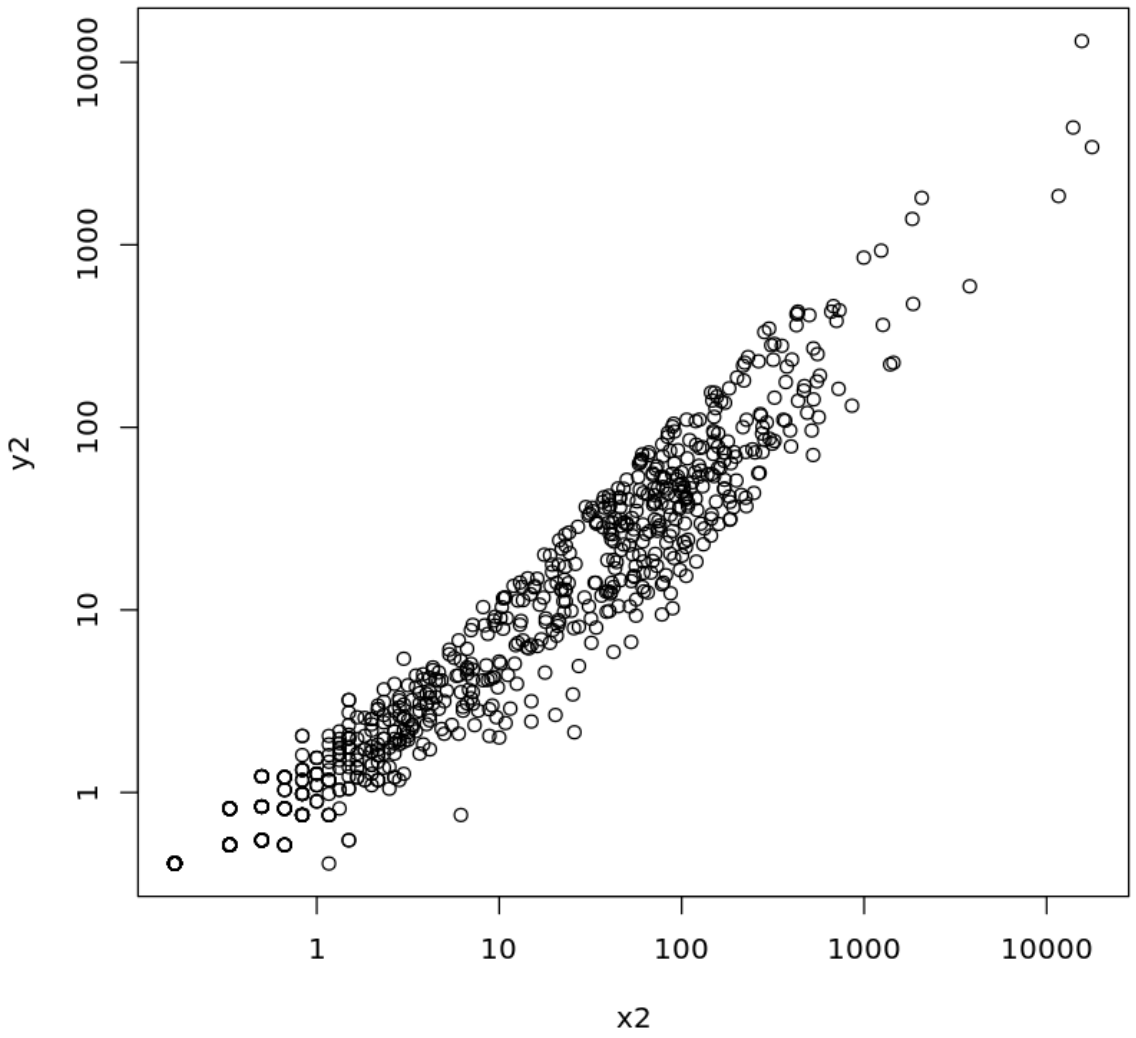
0 · 17737.5

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")
```

OUTPUT PLOT :



WORKFLOW MANAGER : GENERAL OVERVIEW ABOUT WORKFLOW MANAGERS

Workflow managers are tools that help automate and manage complex scientific workflows, which often involve multiple interdependent tasks. The aim of these tools is to make it easier for researchers to manage the many tasks involved in a project, and to ensure that the process of executing the tasks is repeatable and reproducible.

Some of the key features of workflow managers include:

- Definition of tasks: Workflow managers allow users to define tasks, which are the basic units of work in a workflow.
- Dependency management: Workflow managers help manage the dependencies between tasks, ensuring that tasks are executed in the correct order.
- Execution: Workflow managers provide mechanisms for executing workflows, either locally or on remote systems.
- Monitoring and logging: Workflow managers provide mechanisms for monitoring the progress of workflows, and for logging the results of each task.
- Parallel execution: Workflow managers allow tasks to be executed in parallel, which can significantly reduce the time required to complete a workflow.

Some popular workflow managers include Snakemake, CWL, Pegasus, and Nextflow. The choice of workflow manager will depend on the needs of the user, including the size and complexity of the workflow, the resources required to run it, and the desired level of control over the execution process.

In this course we will work with Snakemake.

WORKFLOW MANAGER : SNAKEMAKE

Workflow Management For Bioinformatics :

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.
**https://snakemake.readthedocs.io/
**https://snakemake.github.io

Install Snakemake via Conda :

As always, we need the software before we can start. Install Snakemake into your "bio" environment using conda:

```
$ conda install snakemake
```

How To Run Snakefiles with Snakemake :

You should find the following directory on your Linux VMs: `/home/ubuntu/smake_example/workflow`. The workflow directory contains a couple of example workflows to try out Snakemake. The snm0 _le only contains the shown rule.

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 output files 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`

Define Workflows in Terms of Rules :

- Workflows are defined in terms of rules that define how to create **output** files from **input** files. Dependencies between the rules are determined automatically.
- Hard coded values should be replaced by **wildcards**, since hard coded values don't allow any general rule application. By generalizing the rules the files are no longer hard coded in the workflow and the rule can be applied on any given file.
- The **expand()** function can be used to collect lots of input files.

SNAKEMAKE WORKFLOW FOR RNA-SEQ ANALYSIS

TASK :

Throughout the course you have gained hands-on experience on a couple of programs that can be used to build an exemplarily RNA-Seq workflow.

The commands/steps that you used were the following:

1. Quality control of raw sequencing data using ***fastqc***.
2. Performing adapter trimming using ***flexbar***.
3. Aligning reads to the reference using **STAR**.
   i. Build an index for STAR.
   ii. Performing read alignment.
4. Get read counts for genomic features by using ***featureCounts*** from the subread package.

Build a Snakemake workflow based on the commands that you have used throughout the course to build an exemplarily RNA-Seq workflow.
The final outcome of your Snakemake workflow shall be the ***featureCounts*** output.

EXPLANATION :

This is a Snakemake workflow that performs quality control, adapter trimming, mapping, and counting of gene expression data.

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

- The first rule (all) declares that the input for the workflow is the "genes_counts.txt" file, which is expanded for all 6 samples.
- The second rule (fastqc) performs quality control on the fastq files (read1 and read2) of each sample and generates the fastqc html reports.
- The third rule (adapter_trimming) uses flexbar to trim adapters from the reads and generates the trimmed fastq files.
- The fourth rule (mapping) maps the trimmed reads to a reference genome using STAR aligner and generates the mapped SAM files. The param was used to add prefix to the output files generated by STAR.
- The fifth rule (feature_counts) performs gene counting using the featureCounts tool on the mapped SAM files and generates the gene expression counts file.

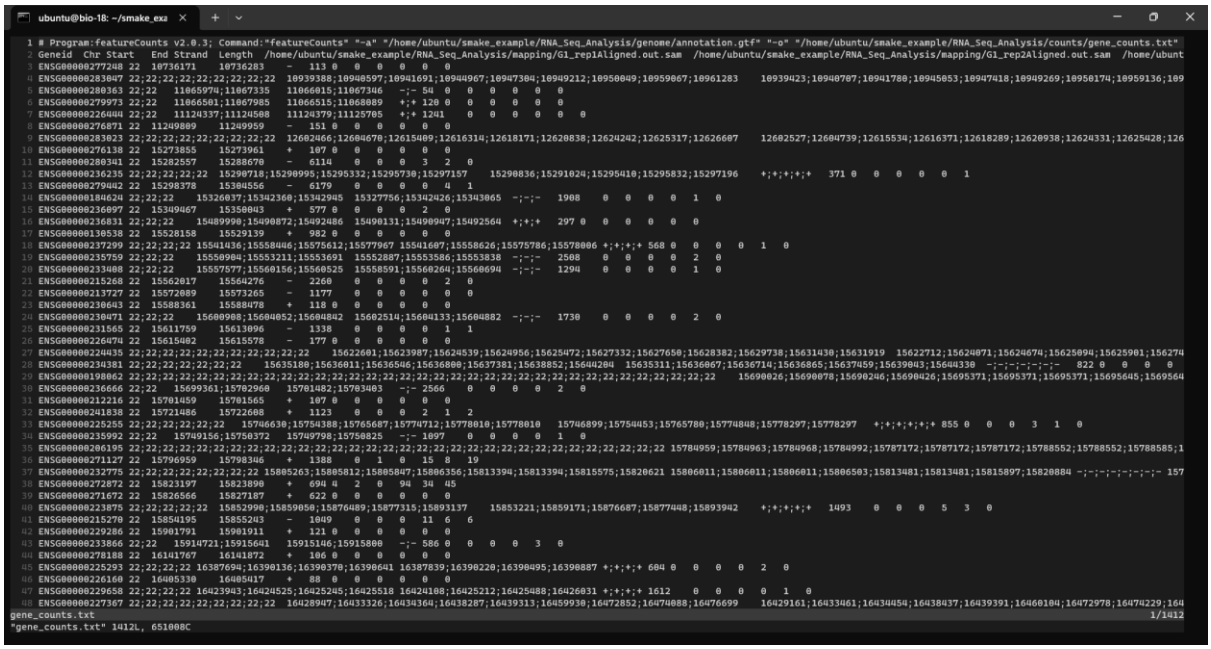
WORKFLOW CODE :

```
ubuntu@bio-18: ~/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}"

rule mapping:
    input:
        input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_1.fastq",
        input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_2.fastq"
    output:
        "/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/{sample}Aligned.out.sam"
    shell:
        "STAR --runMode genomeGenerate --genomeDir /home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/ --genomeFastaFiles /home/ubuntu/smake_example/RNA_Seq_Analysis/genome/genome.fa --sjdbGTFfile /home/

rule feature_counts:
    input:
        input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep1Aligned.out.sam",
        input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep2Aligned.out.sam",
        input3="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep3Aligned.out.sam",
        input4="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep1Aligned.out.sam",
        input5="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep2Aligned.out.sam",
        input6="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep3Aligned.out.sam"
    output:
        "/home/ubuntu/smake_example/RNA_Seq_Analysis/counts/gene_counts.txt"
    shell:
seqA                                                                                                   1/49
```

```
ubuntu@bio-18: ~/smake_exa


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}"

rule mapping:
    input:
        input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_1.fastq",
        input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/trimmed/{sample}_read_2.fastq"
    output:
        "/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/{sample}Aligned.out.sam"
    shell:
        "STAR --runMode genomeGenerate --genomeDir /home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/ --genomeFastaFiles /home/ubuntu/smake_example/RNA_Seq_Analysis/genome/genome.fa --sjdbGTFfile /home/

rule feature_counts:
    input:
        input1="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep1Aligned.out.sam",
        input2="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep2Aligned.out.sam",
        input3="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G1_rep3Aligned.out.sam",
        input4="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep1Aligned.out.sam",
        input5="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep2Aligned.out.sam",
        input6="/home/ubuntu/smake_example/RNA_Seq_Analysis/mapping/G2_rep3Aligned.out.sam"
    output:
        "/home/ubuntu/smake_example/RNA_Seq_Analysis/counts/gene_counts.txt"
    shell:
        "featureCounts -a /home/ubuntu/smake_example/RNA_Seq_Analysis/genome/annotation.gtf -o {output} -p --countReadPairs {input.input1} {input.input2} {input.input3} {input.input4} {input.input5} {inpu
seqA                                                                                                   49/49
```

OUTPUT :

File : `gene_counts.txt`



File : `gene_counts.txt.summary`