Genome Analysis labs

Student Manual

Introduction

The technological advancement around the usage of sequence data has surpassed all expectations in the last decades. The development of new sequencing strategies, hand in hand with the continuous release of improved bioinformatics software, is promising and unstoppable. The amount and breadth of both DNA and RNA data accrue in public databases. However, obtaining high-quality biological information from sequences is not straightforward, and, without proper care, the fast growth of sequence data can cause quantity to obliterate quality. For this reason, a deep understanding of bioinformatics methods, coupled with an educated interpretation of their results, is necessary to unleash the biological information concentrated within sequence data. This strategy opens doors to an incredible number of applications, including biomedical advances, environmental conservation or bioengineering, and uncovers new fundamental insights on biological diversity, evolutionary history, physiology, etc. In the proper hands, sequencing reads become knowledge goldmines.

One of the aims of these labs is to get familiar with bioinformatics tools and methods that are commonly used when analyzing sequencing data. We will do so by working with real-life genomic and expression data, and applying state-of-the-art methods. You will start off by selecting a recent scientific article, and will proceed by re-analyzing their data in a similar way as the authors did on the original study, and re-evaluating their biological conclusions.

The first task at hand will be to plan the analyses that you need to perform. It is very important that you understand the data that you will be working with, and to have clear aims to pursue. You will **adapt the analyses to the available tools and your own interests**. Therefore, those of you starting from the same data can follow different paths and produce different results. Think creatively, and work carefully: feel free to design your own workflow and produce your own scripts towards your particular interests, and introduce the right check-ups to ensure that each step works as expected. Moreover, note that papers often lack important details needed to reproduce their analyses, and some software may require specific concepts or parameters that are not straight-forward at a first glance. You will need to face these problems and try to find your way around it. In those cases remember that Google is your friend :-) .

As mentioned above, you are going to use the real data from the articles you chose. All the data can be found in the Sequence Read Archive (https://www.ncbi.nlm.nih.gov/sra/). See the ‘Getting familiar with your data’ section for more information. The tasks you will perform will often be computationally demanding, so you will need to run them in computer clusters. Therefore, you will use UPPMAX for most of the analyses. See the ‘Working on UPPMAX’ section for more information. Finally, an important task during any scientific project is to document all the analyses you have done. To do so, you will create a GitHub repository where you will record codes and used methods. See the ‘Working on GitHub’ section for more information.

Project Planning

Good project planning and data organization are integral parts of any research project. A research plan can help you understand your project better and make you aware of issues and bottlenecks that can occur in the process. This will help you avoid getting lost in the data and be more efficient. Creating a project plan will be the first thing you will do in these labs. Your project plan should address at least the following points:

- What is the aim of your project? What question(s) do you want to answer with your research?
- What type of analyses will you perform in order to answer these questions? And in which order? Which softwares will you use? Are there any time bottlenecks? If so, can you identify any analyses that will require longer times?
- What is the time frame for your project? Can you define some time checkpoints for when you should have finished certain analyses? When do you need to have finished running all the softwares so you can start to analyze the data?
- What types of data will you be handling? How much space do you need in order to store the data? (You might not know that in advance, but pay attention to this as you work and manage your available space!)
- How will you organize your data? (see “Project Organization”)

To make your project plan you will need to have **read and understood the paper** that you have chosen before the first lab session. It is extremely important to frame the questions you want to answer at early stages. Since you are going to re-analyze data, there will be certain things you will depend on and that cannot be changed, such as the data available or the experimental design. Make sure that you know what data you can use and **identify important information** such as the type of data (RNA-seq, Whole Genome Sequencing, short-reads/long-reads, etc), the biological source (tissue, sample, etc), and how it was generated (type of libraries, special adapters, potential issues, etc). These details will determine the choice of certain analyses or software over others. Once you have identified your aim and are familiar with your data, you can start planning the approach you will follow. A very intuitive way of doing this is by drawing a diagram in which the input/output data for each step is connected by the analyses to perform and the tools to use, such as the one in **Figure 1**.

At this point it is also important to try to spot **steps that can take a long time**, so you can optimize your work and avoid waiting long times. You may also need to account for **unexpected results or errors** that will require your strategy to be revised, so make sure to introduce a certain degree of flexibility in your project plan. You will work on this plan during the first lab session and the TAs will help you assess and polish it.

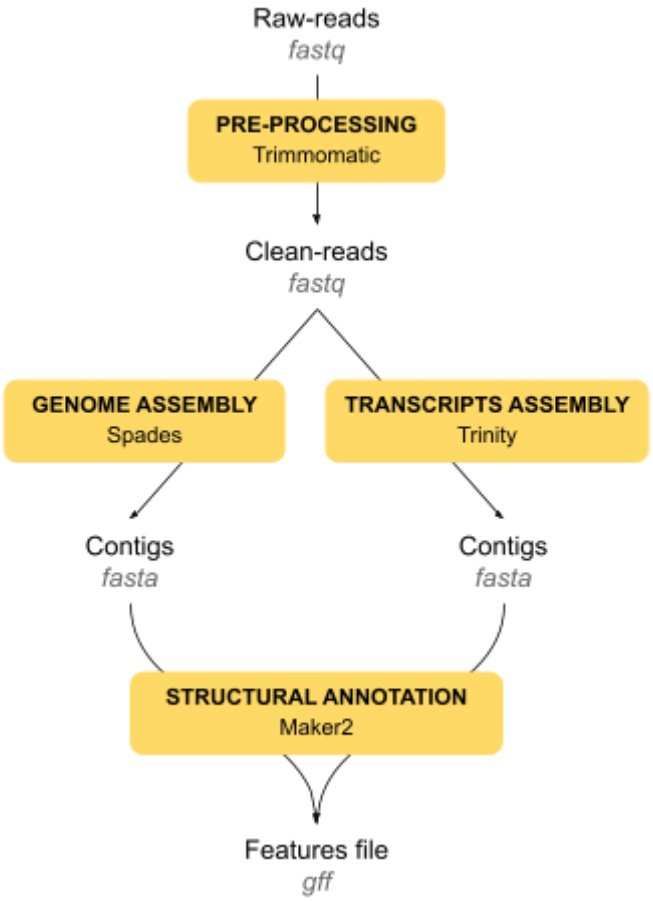

Figure 1. Diagram representing input/output workflow

Project Organization

Since you will work with large amounts of data it is extremely important to **keep a good data structure from the very beginning**. A good project organization will make your work easier as you generate more and more data and will help your "future you" and others understand your project. There are many things that we could discuss here, but we will focus on the importance of keeping clear metadata, and include a few tips about how to organize your working directory.

Keeping the metadata

Metadata refers to information about data, such as how or where it was collected. In your case this will be the information associated with the sequencing data and how it was generated. The process of collecting metadata starts in the lab, before the sequence data are produced. This information is as important as the sequences themselves, since without it the data are useless.

Since you will work with public data, you can obtain the metadata associated with the sequencing reads files from public databases and from the original article. There are many ways to store the metadata, but an easy one is to use a spreadsheet (like Excel, LibreOffice or Google Sheets). Since these **tables will be read by computers you need to keep them structured to avoid parsing errors**. The data to include will depend on each experiment. These are few guidelines to keep in mind when creating a spreadsheet table with the metadata for your sequence files:

- Write one sample per row and one variable per column. For example, all information about the SRR00001 should be on the same row where columns are variables such as: SRA_identifier, tissue, stage, type_of_sequencing, etc.
- Do not merge cells!
- Avoid spaces! Use '-' or '_' instead.
- Be consistent
- Use informative names
- Save the data in plain text files as .tsv (tab-separated values) or .csv (comma-separated values)

Getting familiar with your data

The data you will be working with comes from the NCBI Sequence Read Archive (SRA), a repository of next-generation sequencing data. In order to avoid exceeding our space allocations, we have downloaded, trimmed and, in some cases, subsampled these read files for you. These files are named according to their SRA accession, which can also be found on the paper. To learn more about your data you can follow the following steps:

1. Go to the http://www.ncbi.nlm.nih.gov/sra page and search for the SRA accession of your samples. If you click on the entry of interest you can already start gathering some information. Take some time to examine the page.
2. Under 'Study', click on "All Runs". This page will show all datasets related to the experiment. Inspect the information given on this page.
3. Under "Download" click on "Runinfo Table". This will download a file called SraRunTable.txt that contains information about the whole dataset (not the actual data).

Organizing your working directory

There is not one unique way of organizing a working directory that works for every person and every project. However, there are few guidelines that can be good to keep in mind:

- **Data and code should be separated**. Ideally you would like to keep your code in a repository with version control. Small data files, such as final results, figures and text

can also be included in your repository, but never big data files (there are other repositories that you could use if you wanted to make that data publicly available).

- Keep data files with unique and **informative names**. The read files with which you will be working are named as their SRA accessions. It is useful to give these files unique and meaningful names so we can recognise what they are right away. However, it is advisable to always keep the raw data with the original names. In those cases, we can use symbolic links that create shortcuts to the original files but that have different names. That can be done with a command such as

```
$ ln -s /path/to/original/file /path/to/new/file
```

- It is handy to generate folders or file names that start with a number, since by default they will be shown in alphanumerical order. It is easier to know in which order they were created when they are numerically organised.

- Data files, especially big data files, should be **compressed**. For example, files in FASTQ (sequences with quality scores) or SAM (mapped sequences) format are usually huge, and we normally have a limited amount of storage resources. For example, your home directory on UPPMAX (where you will be saving your analyses) can store up to 32 Gb. Having compressed files is not a problem for most tools and, for those tools that cannot handle it, we can uncompress files on the fly and use the result as standard input to the software. Here are some examples of how to do that:

```
# Uncompressing and piping
$ zcat sampleA.fastq.gz | wc -l

# Uncompressing and input as a file
$ SoftwareX <(zcat sampleA.fastq.gz)
```

An example of an appropriate working directory structure could be the following (note: '.gz' files are compressed; file names followed by "->" are symbolic links):

```
genome_analyses/
├── analyses
│   ├── 01_preprocessing
│   │   ├── trimming_software
│   │   │   ├── conditionA.trim.fastq.gz -> ../../../data/trimmed_data/SRR00001.trimmed.fastq.gz
│   │   │   └── conditionB.trim.fastq.gz -> ../../../data/trimmed_data/SRR00002.trimmed.fastq.gz
│   │   ├── fastqc_raw
│   │   │   └── fastqc_report.txt
│   │   └── fastqc_trim
│   │       └── fastqc_report.txt
│   ├── 02_genome_assembly
│   │   ├── assembly_softwareA_settingsX
│   │   └── assembly_softwareB_settingsX
│   └── 03_structural_annotation
│       └── annotation_software_settingsX
├── code
│   ├── 0_gather_data.sh
│   ├── 1_preprocessing.sh
│   ├── 2_genome_assembly.sh
│   └── 3_structural_annotation.sh
└── data
    ├── metadata
    │   └── sample_information.csv
    ├── raw_data
    │   ├── SRR000002.fastq.gz
    │   └── SRR00001.fastq.gz
    └── trimmed_data
        ├── SRR00001.trimmed.fastq.gz
        └── SRR00002.trimmed.fastq.gz
```

Working with computers

During these computer labs you will work either on the local computers or on the UPPMAX cluster. In both cases, you will work on a Linux environment.

If you have any doubts about how to do that, please check this introduction to Linux and these linux exercises. And, as we said before: Google is your friend! You can find answers online about how to do almost anything in Linux.

Moreover, if you want information about specific programs, you can obtain much of that from the program itself. For **every new command you run**, try searching for manuals (program 'man') or help options (options '-h' or '-help'), e.g.:

```
$ man ssh
$ blastp -help
```

To start the local computer in Linux mode:

Switch on the computer (or reboot if it has been started in a Windows session). When the blue 'hp' screen appears at the beginning of the process, press F12 and the computer will switch to Linux.

Working on the local computers

Most of the analyses should be run on UPPMAX, since it offers much higher computational power, memory and storage space.

However, some softwares are better run on local computers. This is especially true for graphics intensive programs such as Integrated Genome Viewer (IGV), Artemis, Artemis Comparison Tool (ACT) and R studio.

Whenever you want to run a software on a local machine using as input a file which is on UPPMAX, you must first transfer that file to the local machine. You will learn how to do that in the next section.

Sometimes you might need to access the university network from your own computer. If your computer runs Linux or macOS, you can ssh from the terminal with:

```
$ ssh -AX username@solander.ibg.uu.se
```

Depending on your distribution, you may first have to install ssh-client. On Windows you can use PuTTY (see instructions here) to connect to `solander.ibg.uu.se`. Be aware that not all software is accessible when working remotely on solander.

Working on UPPMAX

UPPMAX is Uppsala University's computer cluster for large-scale computation and data storage. We will do most of our work and store most of our files on UPPMAX. To do that, we use ssh (Secure SHell), a program that allows us to connect remotely to other computers and gives us access to a command-line on them.

To connect to UPPMAX from linux, open a terminal and type:

```
$ ssh -AX username@rackham.uppmax.uu.se
```

Replace '`username`' with your UPPMAX user name. The option '**`-X`'** activates X-forwarding on the connection, allowing for graphical data to be transmitted if a program requests it, i.e. programs can use a graphical user interface.

After introducing your password you should see something like what is shown in **Figure 2**:

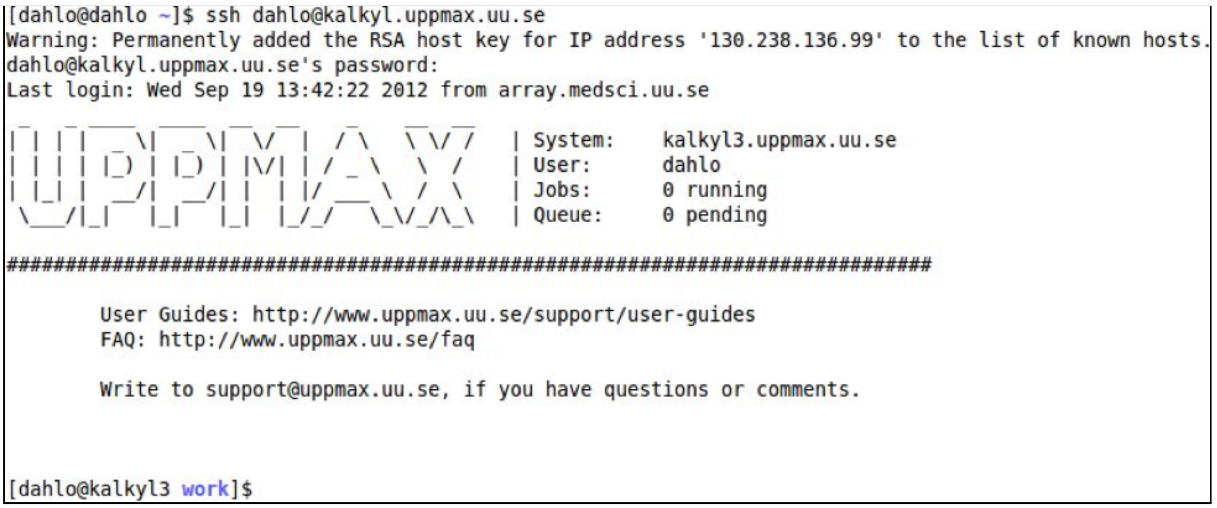

Figure 2. Welcome message from Uppmax. This is your home directory.

This welcome banner means that you have connected to a login node in UPPMAX. When you log in you enter one of the login nodes (a node is basically a PC-unit with its own processor cores, memory, storage etc within a computer cluster like UPPMAX).

Note that **login nodes should NOT be used for computation**, and UPPMAX staff will send angry emails if that happens. In order to run jobs, you have two options:

- Start an interactive session in a working node (see 'Connecting to a working node'), which will allow you to run programs directly from the command line.
- Submit batch jobs (see 'Submitting batch jobs')

During the labs, we will use UPPMAX reservations that will allow us to work with priority using interactive sessions. However, some processes will require longer running times, and will need to be submitted as batch jobs.

Importantly, the software packages are made available to the users in UPPMAX as a module system, requiring that you load specific modules in order to run software (see 'Using UPPMAX modules').

Connecting to a working node

To ask for a slot in a computation node (we will only require two cores per person), type:

```
$ salloc -A uppmax2022-2-5 -M snowy -p core -n 2 -t 00:00:00
--reservation=<code>
```

And include the amount of time asked for under the option '**-t**', and the correct reservation code that will be given to you in the lab. The '-M snowy' flag is needed because we will use computation nodes on the cluster called snowy. (More details on using snowy are here **https://www.uppmax.uu.se/support/user-guides/snowy-user-guide/**). Once you are granted access, the commands you type will be done on that node and not the login node.

During the labs, you will be provided with a reservation code that will be different each day. These reservation codes will give you priority on Uppmax so you don't need to wait to get a working node. However, these reservations will be available just during lab hours. If you want to work on Uppmax after lab hours you will need to queue until the job allocation is granted, and this waiting time will depend on the demand of the cluster at that time. To ask for a job allocation after hours remove the --reservation=<code> option from the previous command:

```
$ salloc -A uppmax2022-2-5 -M snowy -p core -n 2 -t 00:00:00
```

NOTE: When the time you requested is over, you will be automatically disconnected from the node and any programs that are running will be interrupted, possibly causing loss of data!

Submitting batch jobs

Sometimes analyses take a long time to run, and the time we have available in the computer labs is limited.

In such cases, sending a batch job can be very handy, as the jobs will run even when you are not connected to UPPMAX (i.e: you can run time consuming analyses overnight). To do that, you will write a script containing submission information and the commands to be run. You will need to create a text file with the following format:

```
#!/bin/bash -l

#SBATCH -A uppmax2022-2-5
#SBATCH -M snowy
```

```
#SBATCH -p core
#SBATCH -n 2
#SBATCH -t 00:00:00
#SBATCH -J job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user your_email

# Load modules
module load bioinfo-tools
module load ....

# Your commands
<Command_1...>
<Command_2...>
```

Make sure you change all relevant information on the header, such as:
`-n` Number of cores (do not use more than 2 unless specified in the software section!). **Important!** By using this flag you will **reserve** the number of cores, but you still need to specify the number of threads (cores) to use **on the command** of the software to run (e.g. `blastp -num_threads 2`). Otherwise you will still be using a single core, even though you have reserved more.
`-t` How much time you wish to allocate to this job (dd-hh:mm:ss)
`-J` A name for your job
`--mail-user` The system will send messages to this e-mail address specifying when jobs have started and finished. Useful for monitoring your jobs

NOTE: When the amount of hours you assigned to the job is over, the job will be cancelled.

Once you have written your script you need to submit it to the cluster using the following command:

```
$ sbatch sbatch_script.sh
```

To see the status of your submitted jobs you can run the following (replace “username” with your own username):

```
$ squeue -M snowy -u username
$ jobinfo -M snowy -u username
```

Using UPPMAX modules

To run a program on UPPMAX, first we need to upload its module, otherwise the program will not be found by the computer. To do that, type:

```
$ module load <module_name>
```

To see which modules are available in UPPMAX, type:

```
$ module avail
```

NOTE: Most bioinformatics software in UPPMAX are under an umbrella module called 'bioinfo-tools', which needs to be loaded before other bioinformatics modules can be loaded or even seen through '`module avail`'.

If we do not remember the complete name of a module or its spelling, we can use '`module spider`'. '`module spider`' can be used as a searching engine for other modules. It has the advantage of being case-insensitive.

```
$ module spider <search_term>
```

For example if we want to run samtools we need to type:

```
$ module load bioinfo-tools
$ module load samtools
```

Or simply:

```
$ module load bioinfo-tools samtools
```

After that you can call samtools simply by typing:

```
$ samtools
```

NOTE: All modules are unloaded when you disconnect from UPPMAX, so you will have to load the modules again every time you log in. If you load a module in a terminal window, it will not affect the modules you have loaded in another terminal window, even if both terminals are connected to UPPMAX. Each terminal is independent of the others.

To view which module you have loaded at the moment, type:

```
$ module list
```

To unload a module, use:

```
$ module unload <module name>
```

And to unload all loaded modules, type:

```
$ module purge
```

Transferring files between UPPMAX and the local computers

There are several programs that you can use to transfer files between UPPMAX and the local computers, such as ‘scp’ or ‘rsync’. The general way to do it (rsync example) is the following.

To transfer a local file to a remote computer:

```
$ rsync -P file user@machine:/destination/path/
```

To transfer a file in a remote computer to our local machine:

```
$ rsync -P user@machine:file /destination/path/
```

If we want to transfer from UPPMAX to our local computer we will do something like:

```
$ rsync -P username@rackham.UPPMAX.uu.se:/path/to/your/file .
```

Note that you always have to run this on the terminal on your local computer (not on UPPMAX)

Where should I run my analyses?

All analyses should preferably be run in your home folder at UPPMAX (`/home/your_user_name/`). Inside your home you can create folders and organize your files as you wish. The home folder is limited to 32Gb, and for some analysis, this might be insufficient. In this case you can also create a personal folder within the project directory (under `/proj/genomeanalysis2022/nobackup/work/`). Please note that files in this directory are readable (and writable) to everyone in the course. Also note that if you want to keep anything after the end of the course, you need to save it either to your /home or to your private computer.

The “raw data” you will use in your projects is stored in the project folder on UPPMAX (`/proj/genomeanalysis2022/Genome_Analysis/Name_of_your_paper/`)

These files are very big, so **do not move/copy them** to your home or they will use up a very large part of the total disk space you have. Instead, create soft links to them using “ln -s”, e.g.:

```
$ cd /home/your_user_name/
$ ln -s
/proj/genomeanalysis2022/Genome_Analysis/Name_of_your_paper/raw_data/fil
e ./raw_data/
```

Working with GitHub

GitHub (https://github.com/) is a code-hosting platform for version control and collaboration. It allows you to keep and manage multiple parallel versions of a document, and provides you with a platform to invite people to collaborate and to make your work available to others, if you want to share it.

In our project, you will use **GitHub to document and keep track of your analyses and code.** Good documentation allows you to troubleshoot problems, repeat analyses when needed, and allows others to see and understand what you have done. It is through the GitHub documentation that the course teachers can see what you are doing, and how they can help you figure out what might be going on whenever you have a problem with the project.

For these reasons, **it is important that you create a GitHub page which is comprehensive, regularly updated and well organized**. This will be part of the evaluation of your project (see the 'Evaluation' section).

What should you include in your GitHub repository?

You can keep any project-related documentation here, but most importantly you are expected to use GitHub to save your code and to keep an updated wiki on your project. You should save all code you use, be it software commands, a simple bash one-liner, a longer Perl/Python script, an UPPMAX batch script, etc.

Moreover, your GitHub wiki is where you can write explanations about what you are doing, why you are doing it, and what you plan to do. It should include:

- Which paper you are working with
- Your project plan
- Your goals/hypotheses
- A section for each analysis you run (quality control, assembly, RNA mapping, etc) with explanations on methods you are using, the results you got, a short discussion about them, etc
- Any general thoughts, discussions, speculation, etc that you may have about the project and your overall results
- Also suggested: A daily log of what you did on each day of work

IMPORTANT NOTE: **Both code and wiki must be written in a clear and organized way,** with enough explanations for another person to properly understand and reproduce what you are doing. Consider that your documentation should be clearly understandable to:

- Yourself, if you need to check what you did yesterday or 3 weeks ago

- Your classmates, if they want to read and discuss your project
- Your course teachers, so that they have an idea of your progress and can help you with questions and problems you may run into. Moreover, this documentation will serve as a full report of your work and your interpretation of your results, and will be evaluated as such.

Getting started with GitHub

Here we make a summary of how to use some of the basic features of GitHub, but we strongly recommend you to have a look at the links below when starting with GitHub for the first time or when trying to find answers to questions related to the system.

This site provides answers to a large number of GitHub questions, and can be a good starting point when trying to learn how to do something on the platform. Other tutorials can be found here, and here.

But before going into details about creating and managing your repository, we'll briefly introduce the environment you'll be working on.

As shown in **Figure 3**, GitHub is a multilevel interactive environment. Everything will be stored in your ***remote repository***, that will exist in the GitHub cloud. However, you will perform all your analyses from the ***working copy*** located in your computer (i.e. your project folder), and you will eventually transfer the data you want (e.g. your code) to the remote repository. Between you and the cloud there are two extra intermediate levels: the ***local repository*** (that you have to actively create) and the ***index*** (also known as the ***staging area***, that acts as a cache between you and the local repository). All these levels are connected to each other through different functions. We will explain to you the most important that you will use for this project, but you can explore on your own all the possibilities that this virtual environment offers.

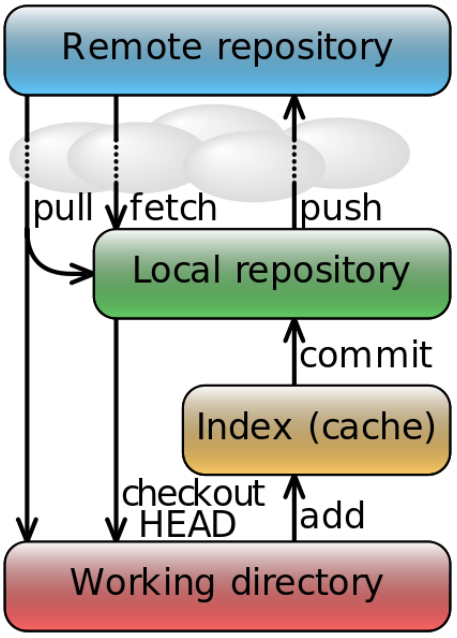

Figure 3. Diagram representing the GitHub environment

Creating a repository

The first thing to create a project (from now on, referred to as a repository) is to go to https://github.com/ and create an account (if you do not have one already). Although in this lab we will be working with public repositories, note that you can apply to the Student Developer Pack to get unlimited private repositories with your university email (more info here).

To create a new repository, click on the plus "+" symbol on the upper right corner and click on "new repository". Make it a public repository so that your classmates and the teachers can also see what you publish. Check the box "Initialize this repository with a README". This will create a readme file (you can change its name later, if you want), which will be the first file in your repository.

Before going any further, take a minute to register your repository in the following form.

Now you have created your *remote repository*, you need to create the *local repository* by "copying" the remote one to your computer. You can do this from the folder from which you will work during the labs, by using the **git clone** command (you will do this just the first time to set-up your local repository):

```
$ cd genome_analysis/
$ git clone https://github.com/git_username/repository_name.git
```

Now you are ready to work on your local repository.

Local vs remote repository

Your remote and local repositories should always be connected, and you can have as many copies of the local repository (e.g. in Uppmax, your laptop, etc) as you want. However, to avoid problems with the different versions, you will have to **git pull** everytime that you work from a different computer to synchronize your last working session. In this way, the changes tracked on your remote repository will be transferred to the current local repository:

```
$ git pull origin master
```

The name *origin* is a local nickname for your remote repository. It is the standard nickname so we suggest you use the same one. *Master* is the main branch of your repository. We won't go into details here but feel free to read more about it.

In the same way, when you finish your working session you want to sync your local repository to the remote one. This time, you use the **git push** command:

```
$ git push origin master
```

Pull and push guarantee that your remote repository is always up-to-date even if you work from different computers.

Building up your repository

You can now start getting familiar with your working environment and create the different folders that you will use (i.e. /data, /documents, /analyses, etc) based on your project plan and organization. For instance, it could be useful to have a starting file where you explain the structure that your repository will have.

```
$ touch folders_structure.txt
```

Every time that you create/modify something in your working environment, git will notice it. You can keep track of these changes by using the **git status** command:

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        folders_structure.txt

nothing added to commit but untracked files present (use "git add" to track)
```

However, as you see from git's answer after checking the status, although git knows that something has changed, it won't update those changes until you command it.

Updating your repository

Although git kind of knows what's going on, you need to actively transfer the data to the remote repository to keep track of every change. To do so, you need three steps.

First, you use the command **git add** to transfer your data to the staging area (the "limbo" area between your working copy and the local repository.) You have to use this option even when you modify an existing file (not only when you create it).

```
$ git add folders_structure.txt
```

The second step sends the data and changes from the staging area to your local repository, and you do it with the **git commit** command. A commit is a comment that describes what changes you have made since the last update (the last commit). This is an essential function to structure the state of your project and to be able to go back to any point. Therefore, it is strongly recommended that the text you add to the commit is clear and describes the step that you are updating.

```
$ git commit -m “Creating file containing the folder structure of my
project”
```

Finally, you need to send the data to the remote repository. You can do this at the end of each session with the **git push** command that we have already introduced. It will send everything contained in the local repository since your last push.

Papers - short summary

In this section you will find brief information about the articles on which your projects will be based.

Paper I

RNA-seq and Tn-seq reveal fitness determinants of vancomycin-resistant *Enterococcus faecium* during growth in human serum

Xinglin Zhang, Vincent de Maat , Ana M. Guzmán Prieto , Tomasz K. Prajsnar , Jumamurat R. Bayjanov, Mark de Been , Malbert R. C. Rogers , Marc J. M. Bonten , Stéphane Mesnage , Rob J. L. Willems and Willem van Schaik

Enterococcus faecium is a commensal bacterium in the human gut that is associated with opportunistic bloodstream infections in immunocompromised hospitalized patients. Moreover, it has recently acquired resistance to multiple antibiotics, which represents a big public health concern. However, the growth and survival mechanisms of this opportunistic pathogen in the bloodstream have not been characterized. In this study you will identify what genes allow *E. faecium* to grow in human blood by different profiling techniques based on RNA-Seq and Tn-Seq.

Illumina PacBio Nanopore Genome Assembly RNA-Seq Tn-Seq Differential Expression

The main analyses included in this study are:

- Genome assembly on *E. faecium*
- Differential gene expression of *E. faecium* on human serum against rich medium using RNA-Seq data.
- Identification of genes that contribute to survival and growth in human serum using Tn-Seq data.

Paper II

Multi-omics reveal the lifestyle of the acidophilic, mineral-oxidizing model species *Leptospirillum ferriphilum*^{T}

Stephan Christel, Malte Herold, Sören Bellenberg, Mohamed El Hajjami, Antoine Buetti-Dinh, Igor V. Pivkin, Wolfgang Sand, Paul Wilmes, Ansgar Poetsch, Mark Dopson

Bacteria can provide a plethora of applications for environmental and industrial practices, due to their abilities to catalyze useful reactions and thrive in a wide variety of conditions. *Leptospirillum ferriphilum* can oxidize iron under a low pH and high metal concentrations, which make it a key player in biomining, with large environmental impact. This study is

aimed at obtaining a complete genome for this organism and characterising adaptations and strategies that are relevant to its lifestyle.

PacBio Illumina Genome assembly RNA-Seq Differential expression

The main analyses included in this study are:

- De novo assembly of long reads obtained using PacBio sequencing.
- Transcriptomics and differential expression analyses of *L. ferriphilum* grown in continuous cultures with ferrous iron and in bioleaching cultures containing chalcopyrite, an abundant ferric mineral found in biomining environments.

Paper III

Metabolic roles of uncultivated bacterioplankton lineages in the Northern Gulf of Mexico "dead zone"

J. Cameron Thrash, Kiley W. Seitz, Brett J. Baker, Ben Temperton, Lauren E. Gillies, Nancy N. Rabalais, Bernard Henrissat, Olivia U. Mason

NOTE: Since this dataset is too big to run analyses in a realistic time you will work with a subset.

Marine regions with low dissolved oxygen concentrations are often called dead zones. Previous researchers have shown that different bacterial groups inhabit these regions, but very little is known about their metabolic roles in the ecosystem. This study employs a metagenomic approach to assemble genomes of bacteria found in different regions of the Gulf of Mexico dead zone and uses metatranscriptomic data to assess their metabolic activity.

Illumina Metagenome assembly Metatranscriptomics Phylogenetics Binning

The main analyses included in this paper are:

- Metagenomic assembly
- Binning of assembly
- Phylogenetic placement of bins
- RNA expression analysis of assembled genomes

Paper IV

The draft genome of tropical fruit durian (*Durio zibethinus*)

Bin Tean Teh, Kevin Lim, Chern Han Yong, Cedric Chuan Young Ng, Sushma Ramesh Rao, Vikneswari Rajasegaran, Weng Khong Lim, Choon Kiat Ong, Ki Chan, Vincent Kin Yuen Cheng, Poh Sheng Soh, Sanjay Swarup, Steven G Rozen, Niranjan Nagarajan & Patrick Tan

Durian (*Durio zibethinus*) is a Southeast Asian tropical plant famous for the sulfury and onion-like odor of its fruits. This species is of major economic value, especially in China. Despite the importance of durian as a tropical fruit crop, there are few genetic studies about this species. This study aims to describe a draft whole-genome assembly of the *D. zibethinus* Musang King cultivar and characterize important genes and pathways involved in different processes in durian.

PacBio Illumina Genome Assembly RNA-Seq RNA assembly Differential Expression

The main analyses included in this paper are:

- Draft genome assembly of the *Durio zibethinus* Musang King cultivar using PacBio reads corrected with Illumina reads.
- Transcriptomics and differential expression of different plant organs from two different *Durio zibethinus* cultivars.

Evaluation

The grade for this lab will be sent individually to each student together with general feedback. In this course, the goal is both to learn bioinformatics methods and how to apply them to obtain biological information.

To pass this part of the course (**grade 3**) you will have to fulfil the following points:

- **A set of basic methods** (see for each project below), and document them in your GitHub wiki. There will be certain deadlines along the course (see below), to make sure that everyone works at a good pace and will fulfil the minimum methods requirements.
- **GitHub wiki**: this will serve as your 'report'. Your text should be understandable to others (not only to yourself!) and contain the main steps needed to reproduce your work. In the wiki, you need to include:
    - the project plan (including specifics about data management),
    - the methods that you have been using for your project, and
    - the biological interpretations and conclusions of your results.

  See "What should you include in your GitHub repository?" for details.
- **GitHub code**: you need to save all commands and scripts that you run to GitHub. It is important to keep this part updated because you might need to check previous steps in order to solve problems that you can have during the analyses. It also allows us to help you troubleshoot issues in your analyses.
- **10 min presentation + 5 discussion**. A good presentation should have a balanced structure covering aims and conclusions on your project. Your slides should be uploaded to Studium before the presentations. To ensure that you get proper feedback, make sure you time your presentation so that it doesn't exceed the given time. After 10 minutes, presentations can be stopped even if the presenter hasn't finished.

On Wednesday 18th May before the presentations should be uploaded to Studium. We will carefully check the state of your Git Wiki and your uploaded slides. This will be the basis (grade 3) of your final lab evaluation.

If you aim to get a higher grade (4-5), you need to fulfil additional criteria. You can get a 4 if you have **correctly answered the Lab manual questions** in the lab manual related to your paper and we deem that you have completed one of the additional requirements listed below:

- **GitHub wiki and presentation**: your wiki and final presentation are well-structured and display a clear understanding of the methods used and the meanings of the results.

- **Extra analyses**: you have performed at least one extra analysis and extensively discussed the result.

For a 5, you need to fulfill all of the above, and multiple extra analyses.

Of course, these will also need to be updated on your GitHub account by Wednesday 18th May in order to be included in the final evaluation.

Paper I - Zhang *et al.* 2017

Basic analyses:

- Genome assembly with PacBio reads.
- Assembly evaluation.
- Structural and functional annotation.
- Synteny comparison with a closely related genome.
- Reads preprocessing: trimming + quality check (before and after)
- RNA-Seq reads alignment against the assembled genome.
- Differential expression analysis between rich medium and heat-inactivated serum conditions.

Extra analyses:

- Genome assembly with Illumina and Nanopore reads.
- Assembly evaluation (extra methods).
- Plasmid identification.
- SNPs calling.
- Evaluate antibiotic resistance potential.
- Identify essential genes for growth in human serum based on the Tn-Seq data analysis.

Paper II - Christel *et al.* 2017

Basic analyses:

- Genome assembly of PacBio reads.
- Assembly quality assessment.
- Structural and functional annotation.
- Synteny comparison with a closely related genome
- Reads preprocessing: trimming + quality check (before and after)
- Mapping and counting RNA-seq reads, and analysing differential expression.

Extra analyses:

- Assembly refinement and comparison with other software
- Annotation refinement (verification and improvement).
- Analysis of metabolic and other functional capabilities
- Comparative genomics: comparison of genes in common with 1 or more species
- Identification of other sequences within the genome (promoters, repeats, mobile elements, …)
- Deeper analysis of the differential expression results: e.g. thorough evaluation of systems and genes that are differentially expressed, comparison with the results in the published paper

Paper III - Thrash *et al.* 2017

Basic analyses:

- Reads preprocessing: trimming + quality check (before and after)
- Metagenome assembly.
- Binning.
- Quality check of assembly and bins.
- Basic phylogenetic placement of bins (Taxonomic ID).
- Functional annotation
- Analysis of activity (expression level) of different bins.

Extra analyses:

- Abundance of different organisms/bins.
- Refine taxonomic ID of assembled genomes.
- Metabolic pathway reconstructions for chosen bins
- Analysis of expression data of chosen gene groups (i.e: respiratory genes, genes involved in carbohydrate metabolism, etc).
- Comparisons across bins (pathways, expression certain genes groups, etc).
- Comparative genomics of bins
- Ortholog gene clustering of bins

Paper IV - Tean Teh *et al*. 2017

Basic analyses:

- Reads preprocessing: trimming + quality check (before and after)
- Genome assembly of PacBio reads.
- Correct the assembly with Illumina reads.
- Assembly quality assessment.

- Structural and functional annotation.
- Differential expression analyses.

Extra analyses:

- Assembly with different parameters.
- Assembly evaluation with more than one method.
- Deeper analyses of differential expression analyses: e.g. different comparisons.
- Transcriptome assembly.

Practical Information About Software

The choice of software to use will depend on the kind of data you have (type of organism, DNA or RNA, type of reads…), and it's something very important to consider beforehand. Here you have a list of software that we have tested for all the suggested papers. We have included as well essential tips for certain software that you will need to keep in mind when you run your own analyses. Of course you are free to use other tools to perform your analyses, but we cannot guarantee that they will work nicely for your data. It is still possible that you get errors when running the analyses with the recommended software. If that happens, keep calm, **read what the error says** and google it if you don't know what to do. Probably it has happened to other people and the solution might be out there. It is a good learning method for you to experience how real research feels and try to figure out your own way.

This section also contains some questions about each analysis. It is very important that you take some time to think about them since it will give you a better understanding of your own data, what you are doing and will also help you to interpret your results. Even though not all the questions might apply to the specific analyses you are doing it is a good exercise to think about them.

Reads quality control

It is not recommended to use the raw data directly in your analyses because the quality of some of them might not be optimal which can lead to errors or artifacts during the further analyses. So the very first thing you should always do when working with reads is to assess the quality of your data. If the quality of your data is not good enough you might need to remove those parts of the reads with low quality or adapters still present on your reads. After that step you should always check the quality of your data again to make sure that all the issues are solved.

- **FastQC** aims to provide a simple way to check the quality of short reads coming from high throughput sequencing pipelines. It performs a modular set of analyses to generate a report that you can use to have a quick impression of whether your data have any problems of which you should be aware before doing any further analysis.

To think about:

- What is the structure of a FASTQ file?
- How is the quality of the data stored in the FASTQ files? How are paired reads identified?
- How is the quality of your data?
- What can generate the issues you observe in your data? Can these cause any problems during subsequent analyses?

Reads preprocessing

An essential step when working with short-reads is removing the low quality base-calls and present adapters from the reads. Sometimes this is done directly by the assembler, but other times this is a manual step that cannot be skipped. Since this analysis can be time-consuming and occupy much of the allocated storage space, we are providing you with already trimmed reads. However, you might need to do it anyway for some specific sample.

- **Trimmomatic** is a fast, multithreaded command line tool that can be used to trim and crop Illumina (FASTQ) data as well as to remove adapters. These adapters can pose a real problem depending on the library preparation and downstream application.
  Notes:
    - *Trimmomatic will perform the different filtering steps in the same order as specified on the command line. In this case the order of the settings is important!*

To think about:
- How many reads have been discarded after trimming?
- How can this affect your future analyses and results?
- How is the quality of your data after trimming?
- What do the LEADING, TRAILING and SLIDINGWINDOW options do?

Genome and Metagenome Assembly

There are different assemblers that can be used depending on the characteristics of your sequencing reads (short or long reads) and the kind of data (DNA, RNA, metagenome…). Here is a list of the main assembly software that can be useful for you:

- **DNA assembly**
    - **Canu** is a *de novo* whole-genome shotgun (WGS) assembler especially designed for long-read sequencing technologies.
      Notes:
        - *Paper II: The software may detect that the quality of the PacBio reads is not as high as it would be optimal, and stop the assembly. To avoid that, you can use the option '*stopOnReadQuality=false*'.*
    - **SOAPdenovo** is a short-read assembly method that can produce large genomes (human-size).

- **Spades** is an assembly toolkit capable of providing hybrid assemblies (combining short and long reads, i.e. Illumina + PacBio). It can work with paired-end reads, mate-pairs and unpaired reads.

  Notes:
  - *Spades requires A LOT of memory and LONG running times. In order to reduce the running time set just one kmer size (by default Spades will use several) and never use the option --careful in this lab. You can also skip the error correction part, but in those cases spades will use even more memory.*

- **Megahit** is an assembler especially designed for large and complex metagenomics NGS reads.

  Notes:
  - *Use "--kmin-1pass" to reduce memory usage or it will crash in UPPMAX*

- **Pilon** is a software tool that can improve draft assemblies, and find variation among strains. It uses short reads alignment against long reads assemblies, to identify inconsistencies between the input genome and the mapped reads.
- **RepeatMasker** is a program that screens DNA sequences and detects repeats and low complexity regions. In order to predict genes accurately in a novel genome, the genome should be masked for repeats. There are two types of masking on DNA sequences, softmasking (i.e. putting repeat regions into lower case letters and all other regions into upper case letters) and hardmasking (i.e. replacing letters in repetitive regions by the letter N for unknown nucleotide)

RNA assembly

- **Trinity** is an assembler useful for the *de novo* reconstruction of transcriptomes from RNA-seq data.
  Note:
  - *On Uppmax use the version 2.4.0.*

To think about:
- What information can you get from the plots and reports given by the assembler (if you get any)?
- What intermediate steps generate informative output about the assembly?
- How many contigs do you expect? How many do you obtain?
- What is the difference between a 'contig' and a 'unitig'?
- What is the difference between a 'contig' and a 'scaffold'?
- What are the k-mers? What k-mer(s) should you use? What are the problems and benefits of choosing a small kmer? And a big k-mer?
- Some assemblers can include a read-correction step before doing the assembly. What is this step doing?
- How different do different assemblers perform for the same data?

- Can you see any other letter apart from AGTC in your assembly? If so, what are those?

Assembly evaluation

Any assembly of our reads will most likely not be optimal. There are different strategies to test how good an assembly is, and it depends on you (or the available references to compare with) to decide which one to use.

- **QUAST** evaluates how good a newly generated genome assembly is. It can work both with and without a reference genome to compare with, and accepts multiple assemblies.

- **MUMmerplot** uses the output from mummer, nucmer, promer (whole-genome alignment) , and generates an alignment dot-plot comparing two aligned assemblies of the similarities between them.
  Note:
  - *If the data contain several contigs, use the parameters -R and -Q. They make the plot interpretation easier by delimiting the different contigs.*
  - *If your assembly contains too many contigs remove the smallest ones or concatenate your assembly to get a readable plot.*

To think about:

- What do measures like N50, N90, etc. mean? How can they help you evaluate the quality of your assembly? Which measure is the best to summarize the quality of the assembly (N50, number of ORFs, completeness, total size, longest contig ...)
- How does your assembly compare with the reference assembly? What can have caused the differences?
- Why do you think your assembly is better/worse than the public one?
- When running metaQuast for a metagenome, it may happen that very few contigs map back to the reference genomes.
    - Is this expected?
    - Does that mean your assembly is bad? Why?

Binning

After assembling reads coming from metagenomes we end up with a mix of contigs that come from the different organisms present in the sample. Binning is the process of classifying those contigs into different organisms to reconstruct their genomes.

- **Metabat** is an automated metagenome binning software which integrates empirical probabilistic distances of genome abundance and tetranucleotide frequency.

To think about:

- Metabat uses information about the contig coverage and tetranucleotide frequency to classify contigs into bins. What are they? Why are they suitable features to use?
- Check how many contigs you have in your metagenome assembly. And look at how many contigs are in your bins.
- Do the numbers add up? Is this expected? What does it mean?

Binning evaluation

After binning a metagenome assembly, it is important to evaluate the recovered bins both to see if the binning software did a good job and to start understanding what is inside each of them. This involves getting basic statistics for each bin such as size and GC content, verifying if each bin represents a complete genome or only part of it, checking if you have only one genome per bin or if it still a mix of different organisms, and so on.

- **CheckM** provides a set of tools for assessing the quality of genomes that is very useful when working with metagenomes. It gives you several basic statistics of the genomes/bins, calculates the estimated completeness and contamination, and gives you a rough taxonomic ID.
  Notes:

  - *To be able to run CheckM you will first have to create a copy of the database that CheckM uses, run module help CheckM/1.1.3 on UPPMAX for how to do this.*

  - *If running the "lineage_wf" analysis, use 4 cores and the flag "--reduced_tree", otherwise the analysis will crash from lack of memory. We recommend that you use a batch script to run this analysis.*

  - *CheckM will complain if the input files have a dot "." in their names (as Metabat output files do!). So make sure to rename Metabat output bin names before running checkM on them.*

To think about:
- How does CheckM estimate completeness of your genomes? And the contamination values? Do you think this reflects the actual completeness and contamination of your genomes?
- What is the difference between the estimates of "contamination" and "heterogeneity"?
- How precise are the taxonomic ID's you initially got?
- Is it useful to have a rough ID of your assembled genome? Why?

Annotation

Once we have assembled our genome/metagenome, the next step is to predict what genetic elements it encodes. A good complement to our annotation is attaching biological information to our predicted genetic elements, such as the biochemical function they perform, or the biological process they are involved in. The best strategy is to use software pipelines that combine several programs in an automated way. Although they are a great starting point, these automatic approaches usually make mistakes and might require extensive manual curation.

- **Prokka** is a software pipeline that combines different tools for the annotation of prokaryotic genomes. It combines both structural and functional annotation by predicting and identifying genetic elements encoded in the genome.
  - *Be aware if you are annotating Archaea, since the default for Prokka is bacterial annotation. Check the "kingdom" option.*

- **BRAKER** is a structural annotation pipeline that combines GeneMark-ET and AUGUSTUS, that uses genomic and RNA-Seq data to automatically generate full gene structure annotations in novel genomes.

  Some arrangement needs be done at the working directory on UPPMAX:

  - *AUGUSTUS needs writing access to the configuration file:*

```
chmod a+w -R /home/your_user_name/working_directory/augustus_config/species/
```

  - *GeneMark key needs to be copied to the home directory:*

```
cp -vf /sw/bioinfo/GeneMark/4.33-es/snowy/gm_key $HOME/.gm_key
```

  Some tips to successful gene prediction:

  - *BRAKER needs the following modules loaded to be able run:*

```
# Load modules
module load bioinfo-tools
module load braker/2.1.1_Perl5.24.1
module load augustus/3.2.3_Perl5.24.1
module load bamtools/2.5.1
module load blast/2.9.0+
module load GenomeThreader/1.7.0
module load samtools/1.8
module load GeneMark/4.33-es_Perl5.24.1
```

  - *BRAKER needs the following flags of the paths to the softwares*

```
-- AUGUSTUS_CONFIG_PATH=/home/your_user_name/augustus_config
-- AUGUSTUS_BIN_PATH=/sw/bioinfo/augustus/3.4.0/snowy/bin
-- AUGUSTUS_SCRIPTS_PATH=/sw/bioinfo/augustus/3.4.0/snowy/scripts
-- GENEMARK_PATH=/sw/bioinfo/GeneMark/4.33-es/snowy
```

  - *Species needs to be updated with `--useexisting` flag once the new species file is created.*
  - *If the genome is masked, use the `--softmasking` flag of `braker.pl`*
  - *BRAKER suggested to be run with 8 cores*

- **EggNOGmapper** is a tool for fast functional annotation of already predicted sequences (works both for genes and proteins) using precomputed eggNOG-based orthology entries. It has an online web server.
  Note:
  - *There is a bug in the online version and does not retrieve the best hit when searching with Diamond. Choose HMM instead of Diamond to get this information.*

To think about:

- What types of features are detected by the software? Which ones are more reliable a priori?
- How many features of each kind are detected in your contigs? Do you detect the same number of features as the authors? How do they differ?
- Why is it more difficult to do the functional annotation in eukaryotic genomes?
- How many genes are annotated as 'hypothetical protein'? Why is that so? How would you tackle that problem?
- How can you evaluate the quality of the obtained functional annotation?
- How comparable are the results obtained from two different structural annotation softwares?

Homology search

Sequence homology shows us how similar two or more organisms are by comparing them at the gene or protein level.

- **Blastn** is a local alignment algorithm included in BLAST that allows you to compare a nucleotide query against a database, or two nucleotide sequences against each other. It can be used from a single gene to the entire genome.

To think about:

- How relevant is the output format that you choose?
- How do the resulting hits vary when you change the minimum e-value?
- How is the alignment score calculated?
- How important is the number of threads when you blast against a database, or against a particular sequence?

Phylogenetic placement

When working with environmental samples and metagenomes we often do not know exactly which organisms we are dealing with, and a single sample can contain a surprisingly large microbial diversity. If we want to study our organism under an evolutionary context, phylogenies are a very useful approach. Reconstructing a tree based on the core genome shared by our organism and closely related species can help us to classify it and give information about its evolutionary history.

- **Phylophlan** is a computational pipeline for reconstructing highly accurate and resolved phylogenetic trees based on whole-genome sequence information. It is especially indicated for metagenomic analyses.
  Notes:
  - *You have to install Phylophlan in your home in UPPMAX. Use the following commands to downloading and uncompressing it:*

```
$ wget https://bitbucket.org/nsegata/phylophlan/get/default.tar.gz
```

```
$ tar -xzvf default.tar.gz
```

  - *Phylophlan has to be run from the installation folder. Create a project folder in the "input" folder containing all protein sequences you want to analyze*
  - *Needs modules python, biopython, FastTree, muscle, usearch/5.2.32*
    **Note: You have to use this version of usearch!*

To think about:
- How precise are the taxonomic ID's you got?
  - How could you improve them?
  - What are the implications of erroneous IDs to the analyses?

Mapping

Aligning reads against a reference can be something you want to do either to determine the reads coverage, to improve or correct an assembly, or to account for differential expression of certain genes, among others. Something important you need to consider here is what kind of organism you are working with, because that will determine what software is more suitable for you.

- **BWA** is a package for mapping low-divergent sequences against a reference genome, as long as they do not contain an exon-intron structure. It will be extensively used across these labs to map reads against assembled genomes with different purposes.
- **STAR** Spliced Transcripts Alignment to a Reference (STAR) is a fast RNA-seq read mapper. It detects splice-junctions between exons and fusion reads.

To think about:

- What percentage of your reads map back to your contigs? Why do you think that is?
- What potential issues can cause mRNA reads not to map properly to genes in the chromosome? Do you expect this to differ between prokaryotic and eukaryotic projects?
- What percentage of reads map to genes?
- How many reads do not map to genes? What does that mean? How does that relate to the type of sequencing data you are mapping?
- What do you interpret from your read coverage differences across the genome?
- Do you see big differences between replicates?

Post-mapping analyses

This section includes tools that you might need to use to manipulate mapping files and/or extract specific information from them.

- **SAMtools** provides various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format.
  Notes:
  - *Alignment files in SAM format are extremely heavy, so generating BAM files is strongly recommended.*
  - *In the process of obtaining a position-sorted final alignment, it is easy to generate several intermediate and dispensable files of large sizes. To avoid this, make sure that you pipe all your SAMtools inputs and outputs in one command to obtain directly a sorted BAM output file when you align your reads against an assembly.*

- **BCFtools** is a set of utilities very useful to find SNPs and indels in our assembly by analyzing short reads mappings against a reference assembly.
  Note:
  - *Paper I: if you are to detect SNPs, you can use the Illumina reads generated in the same study, and some extra ones from this paper.*

To think about:

- What is the structure of a SAM file, and how does it relate to a BAM file?
- What is the structure of vcf and bcf files?
- How many SNPs and INDELs do you get?
- How is the quality of those variants?
- What is the difference between the variant quality, the mapping quality and the fastq quality?
- How are these variantes distributed along the genome?

Read counting

After mapping RNA reads to a reference, you might want to know how many reads map to different genomic features (i. e. genes) or to a specific contigs, for example. That's an essential step for looking at the expression of particular genes and doing a differential expression analysis.

- **HTSeq** is a Python package that will be used for counting reads that map on genomic features.
  Notes:
  - *BAM alignment files must be previously sorted by position*
  - *The features file (.gff) must not contain nucleotide sequences. You might need to manually remove them to use this tool.*

To think about:

- What is the distribution of the counts per gene? Are most genes expressed? How many counts would indicate that a gene is expressed?
- In the metagenomics project, the data doesn't offer enough statistical power for a differential expression analysis. Why not? What can you still tell from the data only from the read counts?

Expression analyses

To compare gene expression between samples, first, we need to count the amount of reads that maps against each gene, and then we determine the differences between them. Once you have the count table with the number of reads mapped to each gene (see "Read counting" section), you can run the differential expression analysis on it:

- **DESeq2** is an R package included in Bioconductor to estimate variance-mean dependence in the counted reads against gene features under different conditions, and test for differential expression based.
  Note:

  - *If you're having problems with version incompatibility, to install DESeq2 for R 3.5 use:*

    If (!requireNamespace("BiocManager", quietly = TRUE))

    install.packages("BiocManager")

    BiocManager::install("DESeq2", version = "3.8")

    NOTE: Installation can take a while. Do it in advance, while you wait for other analyses to run.

  - *In the metagenomics project, the data doesn't offer enough statistical power for a differential expression analysis, so you'll do only look at the expression data*

To think about:
- If your expression results differ from those in the published article, why could it be?
- How do the different samples and replicates cluster together?
- What effect and implications has the p-value selection in the expression results?
- What is the q-value and how does it differ from the p-value? Which one should you use to determine if the result is statistically significant?
- Do you need a normalization step? What would you normalize against? Does DESeq do it?
- What would you do to increase the statistical power of your expression analysis?
- In the metagenomics project, the data doesn't offer enough statistical power for a differential expression analysis. Why not? What can you still tell from the data?

Visualization of the genomes

Finally, sometimes you will need to visually analyse and inspect genome features (annotation, genome comparisons, mapping reads, etc). Here are a couple of tools that will be useful for that purpose.

- **IGV** is a visualization tool for interactive exploration of large, integrated genomic datasets useful to analyse different data types (short and long reads and genomic annotation).
  Notes:

- *If you want to work with IGV locally (recommended) you need to download it from their website (you need to register first).*
- *When you look at the different tracks, keep an eye on the scale since this could be misleading. The scale is shown in the top-left corner of each track between square brackets (e.g [0-2394]). You can change the scale of a track to improve the visualization of it by right clicking on the track and ticking on "Autoscale".*
- *You can also select several tracks (for example all mRNA data) and set the same scale for all of them, by right clicking on the selected tracks and "**Group Autoscale**". In this way you could have different scales for different types of data (e.g. one for the mRNA data and one for the ChiP-seq data).*

- **Artemis Comparison Tool (ACT)** is another visualization tool especially designed for displaying pairwise comparisons between two or more DNA sequences (obtained with BLAST).

Appendix I: Software list

Analysis	Type	Software	Installed	Tutorial
Reads Quality Control	Illumina	FastQC	UPPMAX	https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
	PacBio	Inside Canu	UPPMAX	See canu
Reads preprocessing	Illumina	Trimmomatic	UPPMAX	http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/TrimmomaticManual_V0.32.pdf
DNA assembly	Illumina	SOAPdenovo	UPPMAX	https://vcru.wisc.edu/simonlab/bioinformatics/programs/soap/SOAPdenovo2MANUAL.txt
	Illumina	Spades	UPPMAX	http://cab.spbu.ru/files/release3.14.0/manual.html
	PacBio	Canu	UPPMAX	http://canu.readthedocs.io/en/latest/quick-start.html
	Metagenomics	Megahit	UPPMAX	https://github.com/voutcn/megahit
	Assembly Improvement	Pilon	UPPMAX	https://github.com/broadinstitute/pilon/wiki
	Assembly Masking	RepeatMasker	Online UPPMAX	https://www.repeatmasker.org/cgi-bin/WEBRepeatMasker
Binning	Metagenomics	Metabat	UPPMAX	https://bitbucket.org/berkeleylab/metabat/wiki/Home
RNA assembly	Illumina RNA	Trinity	UPPMAX	https://github.com/trinityrnaseq/trinityrnaseq/wiki/Running-Trinity
Assembly evaluation	Fasta sequences	CheckM	UPPMAX	https://github.com/Ecogenomics/CheckM/wiki/Quick-Start
		Quast	Locally, UPPMAX	http://quast.sourceforge.net/docs/manual.html
		MUMmerplot	UPPMAX	http://mummer.sourceforge.net/manual/#mummerplot
		BCFtools	UPPMAX	https://samtools.github.io

				/bcftools/bcftools.html https://bioinf.comav.upv.es/courses/sequence_analysis/assembly.html
Annotation	Bacteria	Prokka	Locally, UPPMAX	https://github.com/tseemann/prokka#invoking-prokka
	Eukaryotes	BRAKER	UPPMAX	https://github.com/Gaius-Augustus/BRAKER#running-brakerhttps://github.com/Gaius-Augustus/BRAKER#running-braker
	All	eggNOGmapper	Online	
Phylogenetic placement	Protein sequences	PhyloPhlan	UPPMAX	https://github.com/biobakery/phylophlan/wiki
Mapping	DNA & bacterial RNA	BWA	UPPMAX	http://bio-bwa.sourceforge.net/bwa.shtml
	Eukaryotic RNA	STAR	UPPMAX	https://github.com/alexdobin/STAR https://hbctraining.github.io/Intro-to-rnaseq-hpc-O2/lessons/03_alignment.html
Differential Expression	Count features	Htseq	UPPMAX	https://htseq.readthedocs.io/en/release_0.9.1/count.html#count
	Comparison	Deseq2 (R library)	Locally, UPPMAX	https://bioc.ism.ac.jp/packages/2.14/bioc/vignettes/DESeq2/inst/doc/beginner.pdf
Visualization	Genomes - Reads - Annotation	IGV	Locally, UPPMAX	http://software.broadinstitute.org/software/igv/userguide
	Genomes	Artemis	Locally	https://www.sanger.ac.uk/tool/artemis/
	Genomes - BLAST comparisons	ACT	Locally, UPPMAX	https://www.sanger.ac.uk/tool/artemis-comparison-tool-act/
Homology search	Nucleotide sequences	Blastn (BLAST)	Locally, UPPMAX	https://www.ncbi.nlm.nih.gov/books/NBK279690/

Appendix II: Estimated running time for analyses

Analysis	Software	Running time
Reads Quality Control	FastQC	Paper III: ~ 15 min
Reads preprocessing	Trimmomatic	Paper I: ~ 50 min per file (1 core) Paper II: ~ 15 min per file (2 cores)
DNA assembly	Spades	Paper I: (short reads + long reads) ~ 2 h (1 core)
	Canu	Paper I: ~ 4,5 h (1 core) Paper II: ~ 11,5 h (2 cores) Paper IV: ~ 17 h (4 cores)
	Megahit	Paper III: ~ 6 h (2 cores)
	Pilon	Paper IV: ~ 30 min (2 cores)
	RepeatMasker	Paper IV
Binning	Metabat	Paper III: < 30 min (2 cores)
RNA assembly	Trinity	Paper IV: ~ 5,5 h (4 cores)
Assembly evaluation	CheckM	Paper III: ~ 2 h (2 cores)
	Quast	Paper I: < 15 min (1 core) Paper II: < 15 min (1 core) Paper III: ~ 45 min (2 cores)
	MUMmerplot	Paper I: < 5 min (1 core) Paper II: < 5 min (1 core)
	BCFtools	Paper I: ~ 90 min (1 core)
Annotation	Prokka	Paper I: < 5 min (1 core) Paper II: < 5 min (2 cores) Paper III: ~ 1 h (2 cores)
	BRAKER	Paper IV
	eggNOGmapper	Paper II: ~ 1 h (HMM algorithm)
Phylogenetic placement	PhyloPhlan	Paper III: ~ 6 h (2 cores)
Aligner	BWA	Paper I (paired-end reads): ~ 30 min (1 core) Paper I (single reads): < 15 min (1 core) Paper II: ~ 5 h (2 cores) Paper III: ~ 4-6 h (2 cores) Paper IV: ~ 1 h (2 cores)
	STAR	Paper IV: 15 min (8 cores)
Differential Expression	Htseq	Paper I (paired-end reads) ~ 2-7 h (1 core) Paper I (single reads) < 10 min (1 core) Paper II ~ 8 h

	Deseq2 (R library)	Variable

Appendix III: Analyses checkpoints

To ensure that everyone will get to the end of the labs with the minimum analyses expected, you have to plan accordingly to meet the following deadlines, note that there are also two compulsory lab sessions:

Day	Hours	Eukaryotes	Prokaryotes	Metagenomics	Reservation code
24/3	2	*Seminar*	*Seminar*	*Seminar*	
29/3	4	Project planning	Project planning	Project planning	uppmax2022-2-5_1
30/3	4				uppmax2022-2-5_2
8/4	4	Compulsory computer Lab	Compulsory computer Lab	Compulsory computer Lab	uppmax2022-2-5_3
19/4	4	Genome Assembly	Genome Assembly + Genome annotation	Assembly	uppmax2022-2-5_4
21/4	4			Binning	uppmax2022-2-5_5
27/4	4		Comparative genomics	Bin quality assessment + annotation	uppmax2022-2-5_6
28/4	4	Annotation		RNA Mapping	uppmax2022-2-5_7
3/5	4				uppmax2022-2-5_8
11/5	4	RNA mapping	RNA mapping	Phylogeny	uppmax2022-2-5_10
13/5	4	Compulsory computer Lab	Compulsory computer Lab	Compulsory computer Lab	uppmax2022-2-5_11
16/5	4				uppmax2022-2-5_12
17/5	4				uppmax2022-2-5_13