Genomes Workflow - LBCM

01 - General guidelines and workflow organization

Tuesday 22nd July, 2025

Contents

1 Overview

1.1 Module Objectives

This module covers: Learning goals

- Basic Linux terminal concepts and usage.
- What is git and basic usage.
- Conda environment logic.
- Conda basic usage.
- Recommended bioinformatics project organization.

2 The Linux question

Taking in consideration the variety of tools available, their functionalities and general comprehension around systems and workflow organization, the choice of operational system constitutes a central point.

In multiple fronts, Linux get's the spotlight. One big advantage is the cost to install and implement the OS: Zero. With a GPL License, everyone has the right to change and redistribute it, following the condition that the code should stay available (Garrels, 2008). It's portability, security and the existence of a large and committed community all helps on the final OS choice for general Bioinformatics research projects.

In the current chapter, we intend to present basics concepts of the Linux ecosystem. Terminal operation will also be included, since it's intrinsic relation to the system core functionalities. Finally, a brief Linux Mint presentation and guidelines shall be pointed, as it's the initial distro of choice.

Remark

For further and in depth comprehension of base Linux related topics, it is advised to check Garrels, 2008.

2.1 So... What is it?

With the crescent wave on tech development, the fact that some systems were directly developed for specific hardware started to weight a ton on user instruction and final cost of the products. A team of developers then started working on what would come to be the **"UNIX"** project. This operational system brought to the table the ability of code to be recycled, the use of a higher programming language then assembly (C) and an unique overall simplicity (Garrels, 2008).

Definition

UNIX refers to a family of operational systems based on the final product of the original UNIX project. They aim to be simple, elegant and allow code recycling. To achieve their goals, modular logic is applied, alongside a hierarchical file system and innate command line interface.

Then, home PC's started getting traction, in a way that running UNIX on them became a possibility. In this context, the idea of a free system directly connected to the original **UNIX** would led to the birth of **Linux** as a freely available OS.

Definition

Linux is an open source OS based on UNIX. Is composed mainly by (**WhatLinux2025**):

Bootloader: Manages the boot process of the computer.

Kernel: The actual **Linux**. Manages the CPU, memory and peripheral devices.

Initialization system: Sub-system that controls daemons.

Daemons: Background services started up during or after boot.

Graphical server: Sub-system for graphical display.

Desktop environment: Interfaces for direct user interaction.

Applications: Any software that can be installed on the system and offer some service or act as a tool.

2.2 Choosing a Distro

A lifelong debates inside the Linux community revolves around which Linux distribution to choose. One of the biggest reasons for such is that there's no correct answer. Multiple factors interfere in the final choice, and it can be largely personal too.

Across the internet, a variety of guides and lists can be found on the subject. As it's not the central purpose of this project, we won't dive deeply in this matter. But, even though this creative freedom was taken, we will still, in a further chapter, approach one specific Linux distro, called Linux Mint. This decision was made to lay a foundation for some installation instructions, that can change based on the system of choice. Always when that's the case, it shall be pointed out.

The choice of Mint itself was based purely on the fact that it is a distro with large support, reach and with a more reasonable learning curve for Linux newcomers.

Tip

If you want to dive a little deeper on the Mint Distro, we recommend it's User Guide (Mint, 2024b). In it, you can find informations on the system's update manager, multiboot, upgrades from older versions and so on. The online official documentation Linux Mint webpage presents in multiple format all available guides (Mint, 2025).

2.3 Installing The Chosen One

An encouraged practice when making a software available is to always provide an in depth documentation on it. One can assume, then, that almost every, if not all, chosen Linux distribution has a guide to system installation and usage. Taking this in mind, the current project will describe in general lines the basic needs to install a distro, with some key concepts and necessary tools, avoiding redundancy with the material the system itself has.

2.3.1 Before anything...

When faced with the idea to change the current OS to a Linux distro, before major definitive modifications are made, some safety measures are encouraged. The central point is to make sure that everything runs smoothly, and avoid system breaking errors and data loss as much as possible. In this project, we joined some general measures, presented below.

Applications list: The majority of bioinformatics software is designed with Linux based systems in mind, or provides a compatible version. But, it's recommended to make a list of the main needs and goals of the operator with the system to facilitate the post-install and prevent incompatibility issues.

Tip

This list can also be used to help in choosing the distro that fits your needs the best.

Backup: Always backup crucial files.

Driver support: It's a good practice to check also if external drivers of daily and necessary use have functional drivers for Linux based systems.

System requirements: Check beforehand if your current hardware specifications match at least the minimum requirements of the chosen distro. Usually this information can be found in the main webpage of the OS.

Tool and Applications support: Although a lot of software nowadays is developed with Linux compatibility in mind, that's not always the case. Some services like the Microsoft Office Package and the Adobe Suite

don't have a secure and optimal distribution for Linux based systems, making their usage impossible, impractical or unreliable.

After considering the items listed above, you can better decide which path to take.

Tip

If you aren't yet secure of moving totally to Linux, you can set up a Dual Boot function with your current OS and your distro of choice. You can also use a Virtual Machine to emulate a Linux environment and use the OS from inside it!

2.3.2 Tools

In the majority of cases, not much is needed to proceed with a Linux OS installation. Usually, it sums up to:

Bootable device: A storage medium containing the core files for the installation of the chosen distro. Usually consists on the medium with the chosen system's ISO file "flashed" into.

Image writer: A dedicated software that is able to correctly flash a storage device with an system image, turning it into a bootable device.

ISO file: Is a file on the ISO format that contains the necessary structure, directions and files for the correct system installation.

Definition

Flashing: Consists on taking an ISO image and transferring all the partitions, file systems and files that were present on the device used to create the image.
ISO format: The ISO format can be seen as the equivalent of replicating the contents and structure of a CD, DVD or even a Blu-Ray disc.
Storage device: An external drive that can be used as a storage medium, e.g. an USB flash drive.

2.3.3 Validating the ISO

As we can infer, given the above definitions, the process basically consists on taking the target OS general structure and files, flashing it into the bootable device and using the latter to build the system on the machine. Thus, we can logically infer, the success of the operation relies heavily on the integrity of the ISO file, as it represents what will be installed.

To avoid the task of manually reviewing, we have more automated means to check integrity. For that, in the majority of cases, we can use SHA256 sums to complete this task. The usual workflow for such is:

1. Download the correct sha256sum.txt and sha256sum.txt.gpg files according to the chosen distro and version.

   Obs: Make sure to download the files themselves, and not just copy and paste their content on a new file.

2. Open a terminal window where the downloaded sha256sum.txt file is, or navigate to the location.

3. If on a Linux system, verify the **integrity** with:

```
sha256sum -b chosendistro.iso
# Or, if the iso is on another location
sha256sum -b /path/to/chosendistro.iso
```

Remark

On macOS, the command structure is similar, although the command itself is only **shasum**. But, if you are currently using Windows, the process follows a distinct path. If that's the case, we recommend checking the guide available on the Linux Mint Forum, that approaches the problem at hand.

4. If both match, the ISO image was successfully downloaded. If not, it's advisable to re-download the file.

Tip

Before downloading the ISO again, it's always worth the shot to check a second time if the correct hash file is being tested, that being, the hash for the same distro and same distro version chosen.

Remark

The shum files are usually provided with the ISO download or can be retrieved in the same page as the main download mirror.
If you wish to further comprehend what the SHA hash is and how it allows data verification, we recommend the quick post available at How to Geek.

Okay, the file is intact. But, how do we know that the hash file used to test it's integrity is really authentic, and wasn't corrupted or altered? Simple, checking it's authenticity. In the case of Linux Mint, it goes as follow (Mint, 2024a):

1. Import the Linux Mint signing key.

```
gpg --keyserver hkp://keys.openpgp.org:80 --recv-key \
27DEB15644C6B3CF3BD7D291300F846BA25BAE09
```

2. Check key importation:

```
gpg --list-key --with-fingerprint A25BAE09
```

3. Confirm that the output from the previous command presents the key informed on the first command used and the relation to Linux Mint.

4. Check the authenticity with:

```
gpg --verify sha256sum.txt.gpg sha256sum.txt
```

5. If the output points that the signature is good and was made with the key imported, the test was passed.

Tip

PATH: Make sure that all files informed on the commands are on the current work directory of the opened terminal. If not, you can either change the terminal WD, the files location or specify the full path to the files on the commands.

GPG warning: The GPG may send a warn that the signature isn't trusted by the computer, this is expected and normal.

2.3.4 Process Overview

A full installation guide can usually be found on your distro of choice homepage. It usually covers all the needed steps and present FAQ's to help with following them. Here you can see the documentation for installing Linux Mint. The process has similar features for all distros:

1. Choose and download the intended version.
2. Verify the ISO integrity.
3. Create the bootable device.
4. Insert the bootable media onto your machine and boot from it (Usually accessing the BIOS and changing boot order).
5. Enter the system.
6. Start the installation setup and follow the steps pointed.
7. Reboot the machine.
8. Remove the bootable device.

The new OS is almost completely ready to use!

Tip

Most Linux distributions allow the user to test the system before fully installing via the bootable device.

2.3.5 Aftermath

With the installation done, the core of the system is available for you to use. But, some actions are recommended.

Extra drivers: After the first package repository update, check and install possible needed drivers. This step ensures good functioning of connected devices and machine hardware. The Linux Mint Documentation offers a step-by-step guide for this task.

Language and Keyboard Layout: Usually set during installation, but sometimes the user may need to add support to other languages and layouts or change the previously selected. For Linux Mint this can be done on the Languages or Keyboard layout settings.

Multimedia codecs: Some media functionalities require specific codecs that may not come with the installed version, even if the option is given during installation. If that's the case, they must be manually installed, according to the user's need and current distro. The Linux Mint Documentation also approach this subject.

Update the system: Although the base applications, drivers, libraries and others are already installed, it's a good habit to update them after the first reboot and from time to time. Some distributions, like Mint and Ubuntu, offer a GUI application to do so, but all distributions offer a terminal package manager option. For Debian derived distros, like Mint, one can achieve this by running on the terminal the following:

```
# Refresh package list
sudo apt update
# Install all available updates
# for installed packages
sudo apt upgrade
```

Tip

It's a good practice to check which packages are available to upgrade, since the operation may cause versioning problems.

2.4 Command Line and Terminal operation

When using Linux based OS's, one specific "feature" get's the spotlight on the eyes of newcomers: the command line. Basically all tasks done via GUI's can be also executed via command line, usually mediated with a terminal interface.

Taking this in mind, understanding the relevance and basic navigation of the command line, allows an overall better and more efficient system operation. In this sense, the present section aims to establish broad concepts and present general guidelines on the subject.

Definition

The term command line, actually refers to the **shell**. It consists in a program whose purpose is to take keyboard commands and direct them to the OS for execution. In Linux systems, the core **shell** usually is **bash**. **Terminals** are interfaces that allow users with a GUI to interact with the **shell**(Shotts, 2024).

2.4.1 Basics

Nothing better to learn than getting punched by the subject. Let's experiment with basic terminal manipulation and usage. This section is an adaptation of the first chapter of **shotssLinuxCommandLine2024**, in that sense, for further comprehension on the subject, please refer to the main source.

First, open the terminal window (Ctrl+Alt+T on Linux Mint). What you are now seeing, I hope so, is something like:

```
[user@machine ~]$
```

Or, if using conda, probably something like:

```
(base)[user@machine ~]$
```

This is what we call the shell prompt, indicating that it's ready to receive orders. It informs you the current user name, the machine name and the current directory. Knowledge of the directory currently set is of extreme importance, and will be discussed on the next topic.

Now, type out on the terminal the following:

```
[user@machine ~]$ ls
```

The expected output, is a listing of all files and folders available **directly** on the current work directory (I said it was important). Done, you executed your first bash command! The majority of commands supports what we call **arguments**, that can be present in different forms, like **flags**, **parameters** and **positional**. Let's try it out:

```
[user@machine ~]$ ls -a
```

Definition

Argument: A value, whose format can vary, that informs something to the program that will be executed. In this way, he allows more interaction and control of the user with the command itself.

Flag: A flag consists in an argument that doesn't have a value. It usually appears in the format of a single dash followed by a letter (-l, -a, etc).

Parameter: A parameter is an argument with an user determined value. The base format of a parameter is two dashes followed by a space and the value (like in --help). In some cases, the space is substituted for a equality symbol (like in --output=/home).

Positional argument: Indicates to the command which directory or file it should seek and operate. Usually presents itself with a space from the rest of the arguments and the initial command followed by the path to needed directory or file.

Now, you can see a lot more files are being shown. That's because the **-a** flag indicates to the command that you want to see **all** possible files and directories under the current work directory. So, now your output will also include the hidden files, marked by the presence of a single dot before the file name.

Remark

In the majority of commands or terminal executed scripts, you can see a summary of possible flags and operational directions for command usage applying the --help parameter or the -h flag.

Tip

For better quality of life, you can use the up and down arrow keys to access the **terminal command history**. Try cycling through previous commands and re-running them.

This covers up our initial contact with the shell and terminal. As a way of saying goodbye, instead of closing the window in the traditional way, you can exit the shell by simply typing the below!

```
1   [user@machine ~]$ exit
```

2.4.2 The file system tree

Now, we are going to approach the file system tree of Linux based systems. The tree analogy says it all, the files are organized in what's called **hierarchical directory structure**(Shotts, 2024). In this sense, the system has a base for the tree, which we call the **root directory**. All others directories, that we can think as branches, and files, the leaves, will be located inside this root.

Remark

Unix-like systems approach storage devices in a particular way. They're treated as part of the single file system tree that begins on the root. That being, any storage device that is connected to the system, is being **mounted** in certain **point**(location) inside the main tree.

Below you can see some of the root base sub-directories (Garrels, 2008).

Definition

bin: Programs shared by the system, the system administrator and all of the users.

boot: Stores the startup files and the kernel.

etc: Stores the most important system configuration files.

home: Acts as a "root" for the directories of common system users.

lib: Library files needed by the system and it's users.

mnt: Standard mount point for all external file systems.

root: The home directory of the system administrator.

tmp: Used to store temporary files, being cleaned up on reboot.

usr: Files and directories related to all user-related programs.

Remark

Caution! The /root refers to the administrator home directory, while / refers to the root directory.

2.4.3 Navigating with the shell

Remember when was said that the current work directory is important? Now's the time to understand why. As was presented on the previous sub-section, Linux based systems operate on a single rooted file system. This implies that we can, from inside the terminal, access all existent sub-directories and files, but also, that a hierarchical order is present. In this sense, we can conclude that the terminal needs to know where he stands on this tree to operate in the intended way.

When opening a new terminal window, we saw that after the machine name, the **current working directory** is shown. This is where the terminal stands on the tree. The logic of this is easier comprehended by imagining as if we are at the CWD looking forward. We can see all connected branches and the current branch leaves and operate on them and theirs subsequent branches and leaves. We saw this with the ls command previously tested.

Let's try some basic terminal navigation. Open a new terminal window

(Ctrl+Alt+T on Linux Mint). Let's check the current work directory we are on:

```
[user@machine ~]$ pwd
```

Remark

The ~ directory is a short way to refer to the current user home directory. It's the equivalent of /home/user/. You can test this by opening a terminal window and typing the **pwd** command, that shows the name of the current WD.

If all worked as supposed to, the current WD was shown on the screen. Great! Now, let's create a test directory inside the current one and make it our new WD. After that, let's run again the pwd command.

```
mkdir test
cd test
pwd
```

First, we used **mkdir** with the argument 'test' to create a new sub-directory called test. Then, the **cd** command allowed us to change our WD to the one specified as an argument, in this case 'test'. The output of **pwd** shows that we did, indeed, change the current "point of view" of our shell session.

Still in this terminal, try to change again to the test folder. As you can see, the bash returns a message saying that he wasn't able to find such file or directory. This is because it really doesn't exists! At least not inside the WD. Remember, the Linux file system is hierarchical. In this sense, the command searches and operates based on the current work directory, seeing only the branches and leaves that are "inside" of it. This is extremely relevant, for when running scripts or other commands from a terminal, we need to make sure the correct WD is set, because other way, the shell call won't be able to find and operate on the intended files and directories.

Tip

The cd usage can be simplified in some cases. If we want to go back to the user home directory, we can use:

```
cd ~
```

Also, the parent folder can be referred as two dots (..). Using this we can:

```
cd .. # goes up one parent folder
cd ../.. # goes up two parent folders, and so on
```

Remark

Linux based systems have what is called a **PATH environment variable**. This consists on a system variable that can be changed if necessary that contains a list of system paths. Anytime a shell session begins and any command is executed, the system searches not only the WD and it's sub-directories and files, but also the paths included in this environment variable.

2.4.4 Commands

Aiming to provide some base commands, the user can see the cheat sheet below. In depth command usage, availability and functionality can be found on software or distribution documentation.

Cheat Sheet

Command: `ls`
Description: List files in current work directory.

Command: `pwd`
Description: List current work directory.

Command: `cd 'new-directory'`
Description: Changes current work directory.

Command: `mkdir 'new-directory'`
Description: Creates new directory inside current WD or on given path.

Command: `cp 'file' 'copy-location'`
Description: Creates a copy of certain files inside the given location.

Command: `mv 'current-location' 'new-location'`
Description: Moves a file inside the file system.

Command: `df`
Description: Show current amount of free space on the disk drives.

Command: `lsblk`
Description: Show current mounted disks and their mount points.

Command: `rm 'file-path'`
Description: Removes the file indicated.

3 Git

Git is a powerful tool for handling different aspects of projects related to their development, maintenance and distribution. Based on this, we consider that approaching some basic concepts and presenting useful commands should be helpful for newcomers in the bioinformatics field.

3.1 What is it?

Git is a Version Control System that conceptualizes the data that's stored as a series of snapshots of a mini file system. So, by using the command to commit the changes, git takes a snapshot of the current state of the files and stores a reference to it (Straub and Chacon, 2025). The capability of simultaneous work on the same file system in a controlled environment, with a large and lightweight support of branch creation, is what makes git an important tool to learn.

Definition

Branches are a core functionality of nearly every VCS. It consists on diverging from a main state of the current project and being able to work on it without directly affecting the origin (Straub and Chacon, 2025). So, if all go wrong, you can just go back in time. And, if all goes right, you merge. Neat.

The user initially clones an available repository, thus receiving all of it's files and file structure. Then, he can operate locally on them and commit eventual changes, that can be reviewed, if necessary, and merged with the original source.

3.2 The basics

Let's work on a hipotetical example, aiming to exercise git basic operation. Firstly, we must be sure git is installed on the current system. In a new terminal, run the following:

```
git --help
```

If further instructions on git usage are shown, then git is installed and can be used for repository manipulation on the current machine.

Tip

On Linux Mint, you can install git via:

```
sudo apt update # refresh the package listing
sudo apt install git #install the git package
```

Also, if it's the user's first time on git, we recommend checking git's first time setup docs.

Ok, now we are going to fetch the current repository and start to work within it.

```
git clone https://github.com/user/repository_name.git
cd repository_name
ls -a # Here you can see that all files
      # and repository structure are now
      # downloaded and available locally
```

Inside the new folder system gathered, you can start manipulating files and structure. But, first, it's usually recommended to create a new branch for such, aiming to maintain system integrity and organization. Further information on git branching can be seen on the git doc's.

```
git branch --list # shows currently available branches
git checkout -b new_branch_name # creates new branch and switches
                                # automatically to it
```

Tip

You can see where you are in comparison with the bigger picture with the command below. It shows your current branch, it's relation to the origin and your current work "status".

```
git status
```

Let's suppose you made some changes and want to sync them with the non-local repository. Firstly, you need to stage the changes. This means that you need to inform to git what files or paths were altered, with the add command. Then, you need to commit those changes, being a good practice to indicate with a short text what was worked on. Full documentation on git commit's can be found here.

```
# suppose you edit a file named test.txt
git add test.txt
# if more files were altered:
git add test.txt test2.txt
# to commit all changes simply
git commit -m "Commit message"
# if a path was altered, you can add and commit
# it at the same time, without affecting
# the previous changes
git commit -m "What was changed" General_test
```

Tip

You can commit directly all altered files with the -a flag. Keep in mind this doesn't affect new files that git doesn't knows about.

Let's suppose you forgot to add a file on the commit. You can then amend that commit, generating a commit that will replace the results of the first.

```
git add what_you_forgot
git commit --amend
```

Now, you still have the same files and structure on the main branch of the repository, but a whole new view on your work branch. Let's say your work on this new branch is done, and can now be incremented to the main structure. You need first to go back to the master branch, and then perform what we call a branch merge. After all, usually the created branch, now merged, is deleted, aiming to keep directory organization.

```
git checkout master # switch branch
git merge new_branch_name # performs the merge
git branch -d new_branch_name # deletes older branch
```

But, what about the files themselves that were altered? How do we send them to the remote repository, if the changes occurred locally? Simple. We push.

```
git push REMOTE-NAME BRANCH-NAME
```

In the case we were using above, we would:

```
git push origin main
```

Further information on push usage can be found on the github push documentation.

Remark

The opposite to pushing, is pulling. You can pull changes from given remote repository to the current branch. If the current is behind, it gets updated. If divergences are found, the user must specify how they should be approached. For further comprehension on command usage, please refer to the git documentation.

3.3 Useful commands

Cheat Sheet

Command: `add`
Description: Stage certain files for the current commit.

Command: `clone`
Description: Clones to local work space the given repository.

Command: `commit`
Description: Marks staged changes of remote repository to the current branch snapshot.

Command: `pull`
Description: Fetches changes of a remote repository to the current branch.

Command: `push`
Description: Sends local changes to the remote repository.

Command: `status`
Description: Print current local repository status, like current branch and summary of changes.

4 Conda

When working actively with bioinformatics pipelines, software version management and compatibility quickly becomes a major issue, requiring close attention in order to avoid errors and issues that could affect the correct system operation. With this in mind, Conda appears with the intent to minimize or

solve completely those issues. It is a command line tool aimed to package and environment management, running in all base OS's available (Linux, mac and Windows).

Given it's importance and ease to use, for newcomers in the field, we shall approach quickly conda's main topics and commands.

4.1 Installation

Although one can install directly conda from the source, two main minimal installers are currently available. Their focus is on allowing a quick and functional install of the core conda package. These are the Miniconda and Miniforge. The first is related directly to the Anaconda company, while the latter is community maintained. We advise newcomers to first experience the software via the miniforge application, given it's strong community and support. The next steps will be taken considering you followed our advice.

Detailed installation directions can be found on the miniforge github page. We need to replicate the files and structure of the software for it to correctly operate. Happily, this don't need to be done manually. The complete installation and initial setup is realized entirely via the available bash file, that we can obtain with terminal command call or downloading straight from the github source.

```
curl -L -O "https://github.com/conda-forge/miniforge/releases \
/latest/download/Miniforge3-$(uname)-$(uname -m).sh"
# or
wget "https://github.com/conda-forge/miniforge/releases \
/latest/download/Miniforge3-$(uname)-$(uname -m).sh"
# to install simply run the script
bash Miniforge3-$(uname)-$(uname -m).sh
```

Remark

The uname and uname-m portions refer to the current version available of miniforge. They can be altered to specific versions, though not recommended.

After the install, it shall ask you, the user, if auto start should be available. For ease of use, we recommend so, although it remains totally as a personal choice. This way, the initialization of the base conda environment is added to the .bashrc file and to the PATH environment, allowing to be called from any terminal window. To check if all worked as intended, you can run the following command:

```
conda --help
```

If a summary list of the available options shows up, everything is working as intended. You can also see that, on new terminal windows or by resourcing .bashrc the tag (base) shows up before the usual terminal header. This tag indicates the active conda environment.

Definition

An **environment** can be seen as a sub system inside your main one. It store certain packages and applications on specific versions and can be activated or deactivated. In that way, one can better manage dependencies and avoid major inconveniences on package installation and maintenance.

4.2 General usage

The basic operation revolves around environment usage. You can check what environments are currently available with:

```
conda info --envs
```

Then, you can start any of those with:

```
conda activate env_name
```

But let's suppose you're off from a fresh start. The first step is to create an environment. Usually, it's recommended to create a virtual environment for each "bigger purpose". In another terms, it's advised to operate with an environment that revolves around a well defined goal and where the internal components aren't in direct conflict. To create a full genome ambient, one could use:

```
conda create -n genome_workflow
# if you already know base packages that shall be
# needed, you can install them with the creation
conda create -n genome_workflow pandas biopyhton
# to activate the created env, as above we use
conda activate genome_workflow
```

You can see that, after activating it, the (base) before your name on the bash terminal is changed to the current active virtual environment. Done! Now, when calling commands and running pipelines on this terminal window, the specifics of the environment shall be used as rule, and not the base system's default options. But, more important, environment specific software and versions shall be contained in it, being returned to "default", other versions or non-existence, as soon as you exit the venv. To install a package, simply run:

```
conda install package_name
# you can also specify the package source
conda install conda-forge::package_name
```

Tip

You can see all currently installed packages and their versions with:

```
conda list # running within the env
conda list -n env_name # running from base terminal
```

When done, you can exit the venv by simply using:

```
conda deactivate genome_workflow
```

And done, that's the basic conda usage and logic!

Tip

Some packages can be found on specific conda channels. The available channels are found inside the .condarc user file. You can modify such directly via terminal or with the recommended doc's commands:

```
conda config --add channels bioconda
conda config --add channels conda-forge
conda config --set channel_priority strict
```

Keep in mind depending on the installation method, one or more of those can already be present.

4.2.1 An important note on environments

Given the centrality of environments for conda usage, we will approach some interesting functionalities related to environment manipulation and maintenance. For an in-depth look in the subject, we recommend accessing the conda enviroments docs.

Apart from creating a "blank" environment, as pointed in the previous topic, you can use a pre-made file that sets the env with a "model". For this, you will need a environment.yml file, that can be manually created, fetched from some repository or exported from another environment.

```
conda env create -f environment.yml
```

Tip

Environment local export can be made with:

```
conda export
# or specifying the file name
conda export --file name
```

Further instructions and options can be seen on conda doc's.

This file can also be updated to better reflect the newest releases.

```
conda env update --file environment.yml --prune
```

Let's suppose you want Another option is to clone an environment that already exists on your machine, if, let's say, you will start a similar project where the same base shall suffice. For that, use:

```
conda create --name cloneenv --clone originalenv
```

Finally, we have also the option to produce an environment specs list, that explicitly details installed packages, theirs versions and distributions. Then, this list can be used to replicate it's properties on another env of choice.

```
conda list --explicit # only print the specs
# We can use the > terminal operator to put the
```

```
# output of the command inside certain file
conda list --explicit > spec-file.txt
# This file can then be exported to other machines
# or used in the same one to create new identical environments
conda create -- name myenv --file spec-file.txt
# or aggregate said packages on existing ones
conda install --name myenv --file spec-file.txt
```

4.3 Useful commands

Cheat Sheet

Command: `activate`
Description: Activate given environment.

Command: `deactivate`
Description: Returns to base environment.

Command: `create`
Description: Allows environment creation.

Command: `info`
Description: Print basic current conda install informations.

Command: `install`
Description: Allows package install inside an environment.

Command: `list`
Description: Prints installed packages information of the current active environment.
Useful parameters and flags:
explicit - Gives full detail on installed packages specs.
e - Output machine-readable requirement strings of packages.
no-pip - Exclude pip-only installed packages.

Command: `update`
Description: Fetch updates for given packages. Can be used to update conda itself with:

```
conda update conda
```

5 Module Summary

In this module we provided a quick overview of important tools and software of the field in a macro view. First, we understood basic logic and operation of Linux based system, crucial for command line programs usage. Then, the git logic for repository work and cloning was presented, vital for organizing big projects and granting better maintainability. Finally, conda was shown, providing the user with a powerful tool for package integrity and ease of use workflow setups.

5.1 What's next?

With the general topics covered, we shall start approaching the pipeline workflow itself. First, we are going to study data pre-processing.

References

Garrels, Machtelt (June 2008). *Introduction to Linux*. 1.27. Vol. 1. USA: Online. ISBN: 1-59682-112-4. (Visited on 07/01/2025).

Mint, Linux (Dec. 2024a). *Installation Guide*. (Visited on 07/01/2025).

— (Dec. 2024b). *User Guide: Linux Mint*. (Visited on 07/01/2025).

— (Jan. 2025). *Linux Mint Documentation*. https://www.linuxmint.com/documentation.php. (Visited on 07/01/2025).

Shotts, William (Nov. 2024). *The Linux Command Line*. 6th ed. Vol. 1. Online. (Visited on 07/01/2025).

Straub, Ben and Scott Chacon (Apr. 2025). *Pro Git*. 2nd ed. Vol. 1. Online. (Visited on 07/01/2025).