Genomes Workflow - LBCM

01 - General guidelines and workflow organization

Thursday 17th 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, 2024). 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).

2.2.1 Installing Mint

2.3 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).

2.3.1 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!

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

2.3.2 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.

2.3.3 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.

2.3.4 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

4 Conda

5

6 Tools & Software

6.1 Required Software

Installation notes

- **Primary tool:** Tool name and version
- **Dependencies:** Required libraries/packages
- **Optional:** Additional helpful tools

6.2 Installation Guide

```
# Installation commands
conda install -c bioconda tool_name
# or
```

```
sudo apt-get install package_name
```

Listing 1: Software installation

7 Workflow & Methods

7.1 Step-by-Step Protocol

Key parameters

1. **Data preparation:** Input requirements and formatting
2. **Quality control:** Initial data assessment
3. **Main analysis:** Core computational steps
4. **Result interpretation:** Output analysis and validation

Example 7.1. Practical example with real genomic data.

8 Practical Examples

8.1 Example 1: Basic Analysis

Input/output files

- **Input:** Sample data description
- **Command:** Based on approach from **example2024**
- **Output:** Expected results and file formats

```
# Example command with typical genomic data
tool_name -i input_file.fasta -o output_file.txt --parameter
    value
```

Listing 2: Basic command example

8.2 Example 2: Advanced Usage

Complex parameters

```
# Multi-step analysis pipeline
step1_tool input.fasta | step2_tool --param1 value1 >
    intermediate.txt
step3_tool intermediate.txt --param2 value2 -o final_result.txt
```

Listing 3: Advanced analysis pipeline

9 Results & Interpretation

9.1 Output Files

Common output formats and their interpretation:

File formats

- **Format 1:** Description and typical contents
- **Format 2:** When and how to use this output
- **Quality metrics:** How to assess result quality

Remark 9.1. Important note about result interpretation following **author2024**.

10 Scripts & Code

10.1 Helper Scripts

```python
#!/usr/bin/env python3
"""
Helper script for genomic data processing
Usage: python script.py input.fasta output.txt
"""

def process_sequences(input_file, output_file):
    """Process genomic sequences"""
    with open(input_file, 'r') as f:
        sequences = f.read()

    # Processing logic here
    processed = sequences.upper()

    with open(output_file, 'w') as f:
        f.write(processed)

if __name__ == "__main__":
    import sys
    process_sequences(sys.argv[1], sys.argv[2])
```

Listing 4: Data processing script

10.2 Quality Control

```bash
#!/bin/bash
# Quality control pipeline for genomic data

# Check file format
file_format_check.py $INPUT_FILE

# Basic statistics
sequence_stats.py $INPUT_FILE > stats.txt

# Quality assessment
quality_assessment_tool $INPUT_FILE --output qc_report.html
```

Listing 5: QC pipeline

11 Troubleshooting & Best Practices

11.1 Common Issues

Error solutions

- **Memory errors:** Reduce dataset size or increase available RAM
- **Format issues:** Check input file formatting and encoding
- **Parameter tuning:** Guidelines for optimization

11.2 Best Practices

- **Data backup:** Always keep original data copies
- **Version control:** Track analysis versions and parameters
- **Documentation:** Record all analysis steps and decisions
- **Reproducibility:** Use consistent environments and seeds

12 References

- Key papers: **example2024**; **author2024**; **smith2024**
- Software documentation: [Tool official docs]
- Related modules: [Other workflow modules]

13 Exercises & Next Steps

- **[TODO]: Practice with provided sample data**
- **[TODO]: Try different parameter settings**
- **Apply to your own genomic dataset**
- **[TODO]: Explore advanced features**

14 Research Notes

Additional observations and module-specific notes...

Key insight: Connection between this tool and genome annotation pipeline

[IDEA]: Extension: Integration with other bioinformatics tools in the workflow

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. 2024). *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).