ICME 13 - Rome Edition

Crash course on Nanopore sequencing for microbial ecology

Manuela Coci Luigi Gallucci Davide Corso

Last update: 2024-02-20

Contents

1 Introduction **5**
1.1 ICME 13 - Rome 5

2 Introduction to the command line **7**
2.1 Set-up a terminal 7
2.2 Working with the command line 7
2.3 Playing around with basic UNIX commands 8
2.4 Shell scripts 12

Chapter 1

Introduction

MicrobEco is a scientific organization with a common view: disseminate knowledge, strengthen collaboration, create and provide opportunities to learn about, discuss and challenge frontier issues in microbial ecology. We connect scientists and we educate future generations to know and to do not fear microbes.

1.1 ICME 13 - Rome

The 13th edition of the International Course in Microbial Ecology -ICME13 focuses on the Nanopore sequencing techniques and data analysis for microbial ecologists. It is a practical course addressed to maximum 25 PhD students and early career researchers. The course includes the following activities:

- Practical experience in sequencing using Nanopore technology (MinIon)
- Bioinformatic analysis of Nanopore sequencing data.

During the course, students will generate and analyze metagenomic data from DNA samples previously extracted and tested for the training activities. The course is hand-on training with a clear practical setting, preceded by theoretical activities. Online bioinformatic and ad hoc customized pipelines will be presented. Participants will have to introduce themselves with a 3 minutes presentation, including their main research topic.

Chapter 2

Introduction to the command line

2.1 Set-up a terminal

MacOS/Linux: Launch terminal on your machine.

Windows users options: Windows Subsystem for Linux (WSL) –> It creates an Ubuntu terminal environment where you can code just like from a linux Ubuntu terminal. This is useful for the course as well as for practice working in bash. from ubuntu website and from the windows website

SSH client –> Windows: [MobaXterm](https://mobaxterm.mobatek.net/download-home-edition.html . This is a very basic ssh client, meaning, it will allow you to connect to the server and it will serve as a terminal for the course.

If you are already using Visual Studio, it needs one ssh extension plugin to serve as a ssh.

Git for windows –> I am not sure this can be used as a ssh but, in regards to this course, it is also useful to practice coding on the terminal.

Very last-minute resource –> launch this terminal emulator in a new window.

2.2 Working with the command line

Most of the activities of the bioinformatic section of this workshop will be done using the Unix command line (Unix shell).
It is therefore highly recommended to have at least a basic grasp of how to get around in the Unix shell.

We will now dedicate one hour or so to follow some basic to learn (or refresh) the basics of the Unix shell.

[!question] What is the UNIX SHELL? What is Bash?

> [!todo] The shell is a program that enables us to send commands to the computer and receive output. It is also referred to as the terminal or command line. Some computers include a default Unix Shell program.
>
> [!todo] The most popular Unix shell is Bash, Bash is a shell and a command language.

For a **Mac** computer running macOS Mojave or earlier releases, the default Unix Shell is Bash.

For a **Mac** computer running macOS Catalina or later releases, the default Unix Shell is Zsh. Your default shell is available via the Terminal program within your Utilities folder.

The default Unix Shell for **Linux** operating systems is usually Bash.

2.3 Playing around with basic UNIX commands

2.3.1 Some notes!

These commands:

```
mkdir unix_shell
cd unix_shell
```

…are commands you need to type in the shell. Each line is a command. Commands have to be typed in a single line, one at a time. After each command, hit "Enter" to execute it.

Things starting with a hashtag:

```
# This is a comment and is ignored by the shell
```

…are comments embedded in the code to give instructions to the user. Anything in a line starting with a `#` is ignored by the shell.

Different commands might expect different syntaxes and different types of arguments. Some times the order matters, some times it doesn't! The best way to check how to run a command is by taking a look at its manual with the command `man` or to the –help for a shorter version of it:

```
man mkdir

# You can scroll down by hitting the space bar
# To quit, hit "q"
```

```
mkdir -h

# did it work?
```

2.3.2 Creating and navigating directories

First let's see where we are:

```
pwd  # print working directory
```

Are there any files here? Let's list the contents of the folder:

```
ls

# or

ll
```

Let's now create a new folder called `unix_shell`. In addition to the command (`mkdir`), we are now passing a term (also known as an argument) which, in this case, is the name of the folder we want to create:

```
mkdir unix_shell
```

Has anything changed? How to list the contents of the folder again?

HINT (CLICK TO EXPAND)

> ls

And now let's enter the `unix_shell` folder:

```
cd unix_shell
```

Did it work? Where are we now?

HINT

> pwd

2.3.3 Creating a new file

Let's create a new file called `myfile.txt` by launching the text editor `nano`:

```
nano myfile.txt
```

Now inside the nano screen:

1. Write some text
2. Exit the "writing mode" with ctrl+x nano

3. To save the file, type **y** and hit "Enter"
4. Confirm the name of the file and hit "Enter"

List the contents of the folder. Can you see the file we have just created?

2.3.4 Copying, renaming, moving and deleting files

First let's create a new folder called `myfolder`. Do you remember how to do this?

HINT

> mkdir myfolder

And now let's make a copy of `myfile.txt`. Here, the command `cp` expects two arguments, and the order of these arguments matter. The first is the name of the file we want to copy, and the second is the name of the new file:

```
cp myfile.txt newfile.txt
```

List the contents of the folder. Do you see the new file there?

Now let's say we want to copy a file and put it inside a folder. In this case, we give the name of the folder as the second argument to `cp`:

```
cp myfile.txt myfolder

# while typing myfold.. try using the TAB to predict the name of the folder!

cp myfile.txt myfolder/  # it will recognise it is a directory and add the / at the en
```

List the contents of `myfolder`. Is `myfile.txt` there?

```
ls myfolder
```

We can also copy the file to another folder and give it a different name, like this:

```
cp myfile.txt myfolder/copy_of_myfile.txt
```

List the contents of `myfolder` again. Do you see two files there?

Instead of copying, we can move files around with the command `mv`:

```
mv newfile.txt myfolder
```

Let's list the contents of the folders. Where did `newfile.txt` go?

We can also use the command `mv` to rename files:

```
mv myfile.txt myfile_renamed.txt
```

List the contents of the folder again. What happened to `myfile.txt`?

Now, let's say we want to move things from inside `myfolder` to the current directory. Can you see what the dot (`.`) is doing in the command below? Let's try:

```
mv myfolder/newfile.txt .
```

Let's list the contents of the folders. The file `newfile.txt` was inside `myfolder` before, where is it now?

The same operation can be done in a different way. In the commands below, can you see what the two dots (`.`) are doing? Let's try:

```
# First we go inside the folder
cd myfolder

# Then we move the file one level up
mv myfile.txt ..

# And then we go back one level
cd ..
```

Let's list the contents of the folders. The file `myfile.txt` was inside `myfolder` before, where is it now?

To remove files :

```
rm newfile.txt
```

Let's list the contents of the folder. What happened to `newfile.txt`?

And now let's delete `myfolder`:

```
rm myfolder
```

It didn't work did it? An error message came up, what does it mean?

```
rm: cannot remove ‘myfolder’: Is a directory
```

To delete a folder we have to modify the command further by adding the flag (`-r`). Flags are used to pass additional options to the commands:

```
rm -r myfolder
```

Let's list the contents of the folder. What happened to `myfolder`?

> [!warning] **In Bash, If you remove the wrong file/directory, it is gone forever!! (no recycle bin!) aka BE CAREFUL!!**

2.3.5 Copying this/any GitHub repository

Remember: You can check where you are with the command `pwd`.

To have access to some scripts and some of the mock data of this lesson, let's copy this GitHub repository to your home folder using `git clone`:

```
git clone https://github.com/TomasaSbaffi/ICME12_Verbania-2023
```

You should now have a folder called **ICME12_Verbania-2023** in your directory (where you were when you launched the command!).

To get the latest updates, pull the changes from GitHub using `git pull`:

```
cd ICME12_Verbania-2023
git pull
```

2.4 Shell scripts

You don't need to run a command at a time, you can pre- think, organize series of actions (a program) that you can then execute within Bash:

- Write a shell script that runs a command or series of commands for a fixed set of files.
- Run a shell script from the command line.

We want to see the first 10 lines of the forward fastq file of sample 1:

```
zcat S1_R1.fastq.gz | head -n 10
```

where: - **zcat** lets you see what is in a zipped file while not unzipping it - | is the PIPE, it lets you connect actions: the output of a command is the input of the next command

OR we can write a little program to a file and execute it as many times as we want:

- let's open a new nano/vim file fastq_head.sh and write in it:

```
#!/bin/bash
zcat S1_R1.fastq.gz | head -n 10 > S1_fwd_head.txt

# we also want to write the output to a .txt file
# #!/bin/bash   --> it is not necessary as in most cases Bash is your home shell
```

Exit nano and see that in the directory there is a new file, let's execute it.

```
sh fastq_head.sh
```

Something more useful! I need to know how many reads are in the same file and output a .txt file with the number of those, knowing that:

- grep -e "XXXX" is a function that finds and prints strings from a file
- wc is a command that *can* help count lines
- fastq files have a "@" in the reads headers.

HINT

zcat S1_R1.fastq.gz | grep -e "@" |wc -l

-l is the flag that let's you count lines