Microbial Ecology: Introduction into Data Analysis

Nina Dombrowski

Institute for Biodiversity and Ecosystem Dynamics (IBED)
University of Amsterdam (UVA)

n.dombrowski@uva.nl

Table of contents

Introduction

Welcome to the *Microbial Ecology Command-Line Crash Course*!

In this short tutorial, you will learn how to use the command line and a High-Performance Computing (HPC) environment in order to move from raw 16S sequencing reads to a basic count table of microbial taxa.

This course is designed for students in microbial ecology with little or no prior experience using the command line. During this tutorial, you will learn how to:

1. **Install a Terminal**
    - How to access a Unix/Linux command line
    - How to run basic commands
2. **Document your Code**
    - How to use markdown, Quarto, and comments in scripts
    - How to track your work and make your analyses reproducible
3. **Navigate the Command Line**
    - Navigating the filesystem (`cd`, `ls`, `pwd`)
    - Working with files (`mkdir`, `cp`, `mv`, `rm`, `cat`, `head`, `less`)
    - Searching and filtering data (`grep`, `wc`, `cut`)
    - Viewing and understanding FASTQ files
4. **Work with an HPC**
    - Connecting to the cluster (SSH)
    - Understanding file systems and job schedulers (e.g., SLURM)
    - Running and interpreting command-line tools (e.g., `fastqc`, `seqkit`)
    - Setting up and activating conda environments
    - Running workflows with Snakemake

1 Setting up a terminal

1.1 Terminology

The **command-line interface (CLI)** is an alternative to a graphical user interface (GUI), with which you are likely more familiar. Both allow you to interact with your computer's operating system but in a slightly different way:

- In a **GUI**, you click buttons, open folders, and use menus
- In the **CLI**, you type text commands and see text output

The CLI is also commonly called the **shell**, **terminal**, **console**, or **prompt**. These terms are related but not identical:

- The **terminal** (or **console**) is the window or program that lets you type commands. It is the interface you open, for example, *Terminal* on macOS, *WSL* on Windows, or an HPC login session.
- The **shell** is the program that actually interprets the commands you type inside the terminal and tells the operating system what to do. The shell understands commands like `cd` that is used to change directories.
- **Prompt**: the text displayed by the shell that indicates it is ready to accept a command. The prompt often shows useful information, like your username, machine name, and current directory. Example of a prompt: `user@machine:~$`

There are several types of shells — for example, **bash** or **zsh** (that use slightly different languages to issue commands). This tutorial was written on a computer that uses **bash**, which stands for *Bourne Again Shell*.

1.2 Installation guides

1.2.1 Linux

If you're using Linux, you already have everything you need — no installation required. All Linux systems come with a terminal and a shell, and the default shell is usually Bash.

You can open a terminal from your applications menu or by searching for Gnome Terminal, KDE Konsole, or xterm, depending on your desktop environment.

To confirm which shell you're using, type:

```
echo $SHELL
```

If the output does not end in `/bash`, you can start Bash manually by typing:

```
bash
```

You should then see a new prompt (often ending in `$`) indicating that Bash is running.

1.2.2 Mac

All Mac computers also come with a built-in terminal and shell. To open the terminal:

- In Finder, go to Go → Utilities, then open Terminal.
- Or use Spotlight Search (+ Space), type Terminal, and press Return.

The default shell depends on your macOS version:

- macOS Mojave (10.14) or earlier → Bash
- macOS Catalina (10.15) or later → Zsh

Check which shell you're currently using:

```
echo $SHELL
```

If the output does not end in `/bash`, you can start Bash manually by typing:

```
bash
```

Then check again with echo `$SHELL`. If you have trouble switching, don't worry — nearly all commands in this tutorial will also work in Zsh.

1.2.3 Windows

Unlike macOS or Linux, Windows doesn't include a Unix-style terminal by default, so you'll need to install one of the following:

1. MobaXterm (recommended)
    - Provides a terminal with Linux-like commands, built-in
    - Installation guide: MobaXterm setup instructions
    - Easiest option for beginners and lightweight to install.
2. Windows Subsystem for Linux (WSL2)
    - Gives you a full Linux environment directly on Windows.
    - Recommended if you're comfortable installing software or already have some command-line experience.
    - Uses Ubuntu by default, which includes Bash and all standard Linux tools.
    - Installation guide:Microsoft WSL install instructions

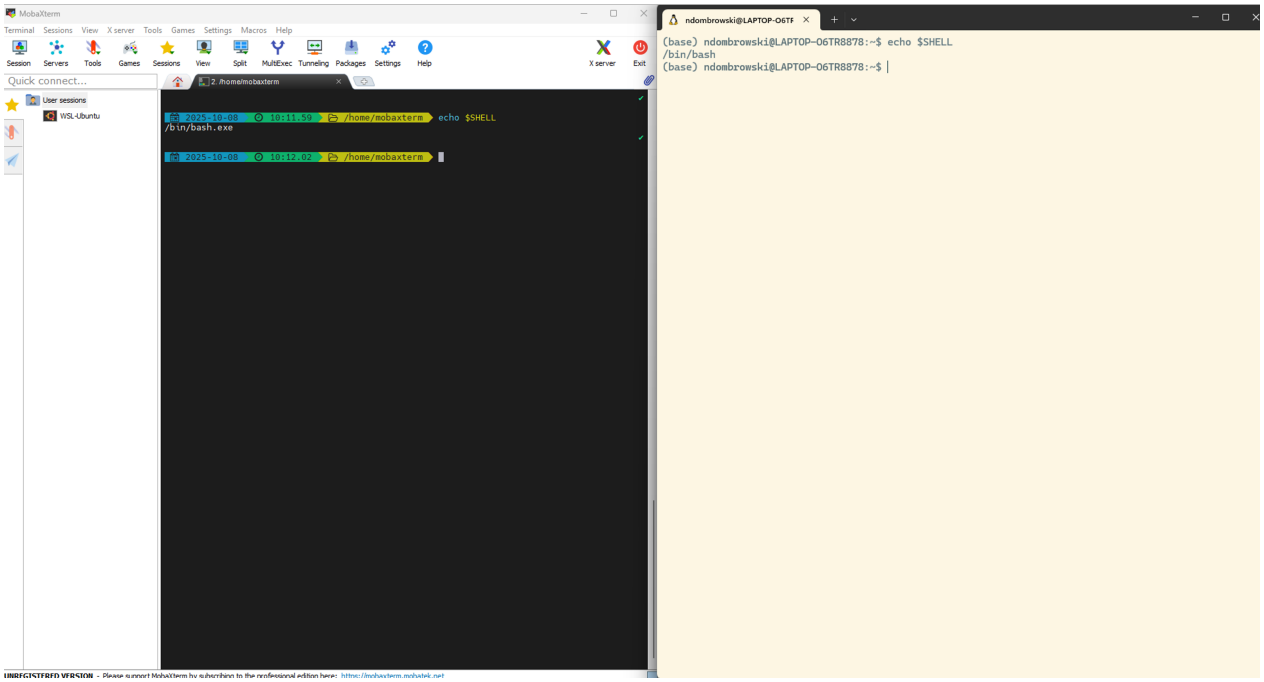
Once installed, open your terminal (MobaXterm or Ubuntu via WSL2) and verify that Bash is available:

```
echo $SHELL
```

1.3 Sanity check

After setup, open your terminal. You should see something like this:

Mobaxterm terminal **WSL terminal**



If you see a prompt ending in `$` (for example `user@machine:~$`), your shell is ready — you're all set to follow along with the tutorial!

2 Documenting your code

Documenting your code is crucial for both your future self and anyone else who might work with your code. Good documentation helps others (and your future self) understand the purpose, functionality, and usage of your scripts, much like a detailed lab notebook.

For more in-depth guidance, see A Guide to Reproducible Code in Ecology and Evolution. While examples are mainly in R, the principles are general and apply across programming languages.

2.1 Choosing your editor

2.1.1 Plain text editor

Avoid visual editors like Word, as they are not designed for code and can inadvertently change syntax (e.g., replacing backticks \`` `with apostrophes`' ').

Start simple with a **plain text editor**, such as:

- **TextEdit** (Mac)
- **Notepad** (Windows)

These allow you to write and save code safely, though they lack advanced features like syntax highlighting or integrated code execution.

2.1.2 Rmarkdown in RStudio

RMarkdown combines plain text, code, and documentation in one document. You can write your analysis and explanatory text together, then "knit" the document to HTML, PDF, or Word.

To create an RMarkdown file in RStudio:

1. Go to **File → New File → R Markdown**
2. Choose a **title**, **author**, and **output format**
3. Write your code and text
4. Click **Knit** to render the document

More info: RMarkdown tutorial

2.1.3 Quarto in Rstudio

Quarto is a next-generation alternative to RMarkdown. It supports R, Python, and other languages, and offers more output formats and customization options.

To create a Quarto document:

1. Go to **File → New File → Quarto Document**
2. Choose a **title**, **author**, and **output format**
3. Click **Render** to generate your document

More info: Quarto documentation

2.2 Markdown for Documentation

Markdown is a lightweight language for formatting text. You can easily add headers, lists, links, code, images, and tables.

Headers:

Use `#` to add a header and separate different sections of your documentation. The more `#` symbols you use after each other, the smaller the header will be. When writing a header make sure to always put a space between the `#` and the header name.

```
# Main Header
## Subheader
```

Lists:

Use `-` or `*` for unordered lists and numbers for ordered lists.

Ordered lists are created by using numbers followed by periods. The numbers do not have to be in numerical order, but the list should start with the number one.

```
1. First item
2. Second item
3. Third item
4. Fourth item
```

```
1. First item
2. Second item
3. Third item
    1. Indented item
    2. Indented item
4. Fourth item
```

Unordered lists are created using dashes (`-`), asterisks (`*`), or plus signs (`+`) in front of line items. Indent one or more items to create a nested list.

```
- First item
- Second item
- Third item
- Fourth item
```

```
 - First item
- Second item
- Third item
    - Indented item
    - Indented item
- Fourth item
```

You can also combine ordered with unordered lists:

```
1. First item
2. Second item
3. Third item
    - Indented item
    - Indented item
4. Fourth item
```

Code Blocks:

Enclose code snippets in triple backticks followed by the computational language, i.e. bash or python, used.

````
```bash
grep "control" downloads/Experiment1.txt
```
````

Links:

You can easily add links to external resources or within your documentation as follows:

```
[Link Text](https://www.example.com)
```

Emphasis:

You can use * or _ to write italic and ** or __ for bold text.

```
*italic*
**bold**
```

Pictures

You can also add images to your documentation as follows:

```
![Alt Text](path/to/your/image.jpg)
```

Here, replace `Alt Text` with a descriptive alternative text for your image, and `path/to/your/image.jpg` with the actual path or URL of your image.

Tables

Tables can be useful for organizing information. Here's a simple table:

```
| Header 1 | Header 2 |
| ---------| ---------|
| Content 1| Content 2|
| Content 3| Content 4|
```

3 Navigating the command line

3.1 `pwd`: Find out where we are

Once your terminal is open, let's get oriented by typing your first command:

```
pwd
```

The command `pwd` stands for **print working directory**. It tells you where you currently are in the file system — that is, which folder (directory) your shell is "looking at" right now. You should see something like:

```
#| eval: true
#| echo: false
pwd
```

Tip: finding the desktop on different user systems

Your home directory will be something like /Users/YourUserName but the path might be slightly different depending on your operating system. Below you find some hints to orient yourself better for different terminal interfaces/operating systems:
For MAC users:

- The home directory should be `/Users/YourUserName`
- To access the current folder in Finder you can try using `open .`
- Your desktop should be here `/Users/YourUserName/Desktop`

For Mobaxterm users:

- Your home directory is `/home/mobaxterm`
- By default this home directory is in a temporary folder that gets deleted every time you exit Mobaxterm, To give this folder a persistent home, do the following:
  - Settings –> Configuration –> General
  - In General set Persistent home directory to a folder of your choice
- To access the current folder in the file explorer you can type `explorer.exe .`
- The path to the desktop should something like this `/mnt/c/Users/YourUserName/OneDrive/Desktop` or `/mnt/c/Users/YourUserName/Desktop`

For WSL2 users:

- The home directory is `/home/YourUserName`
- To access the current directory in the file explorer you can type `explorer.exe .`
- The path to the desktop would be something like this `/mnt/c/Users/YourUserName/OneDrive/Desktop` or `/mnt/c/Users/YourUserName/Desktop`

Accessing the Uva OneDrive folder:
To access the OneDrive Uva folder, you need to add quotes between the absolute file path since bash does otherwise not know what to do with the space in the file name, i.e. `cd "/mnt/c/Users/UserName/OneDrive - Uva"`. This is also a good example for why you normally do NOT want to add spaces in your filepath.

3.2 `ls`: List the contents of a directory

Now that we know where we are, let's find out what files and folders exist in our home directory. For this you can use the `ls` command, which allows us to list all the contents of the directory you are currently in:

```
ls
```

In my case this returns something like this (and this will look different for your current directory):

```
Custom_scripts  LICENSE  README.md  _quarto.yml  docs  img  index.qmd  source  styles.css
```

The colors will look different depending on the CLI you use but in the example above you see a list of files (in bold text) and folders (green-highlighted text) that can be found in my current directory.

Since this output can easily become over-whelming if we deal with a lot of files and folders, lets look a bit closer into how we can optimize our commands by looking at the general structure of a command.