

Bioinformatics · Practical notes

Terminal Basics for Bioinformatics

From Linux and WSL2 to Reproducible Exercises
with Bash, Conda and Python

This note is for students with little experience of terminals or programming. It explains, step by step, where a command runs, which files it affects and how to check its result. Students using native Linux and those using WSL2 on Windows follow the same Bash and Python exercises. Resource limits, software environments and file locations are explained through the decisions they require. A small synthetic sequence table connects basic file inspection to a repeatable analysis.

Code and environments:
main_scripts/terminal_basics/

Tuna Birgün
github.com/tunabirgun

31 October 2026

Contents

1 Basic concepts **2**
2 Installing Linux and WSL2 **3**
3 Allocating RAM and processor resources to WSL2 **4**
4 The file system and basic terminal use **6**
5 Combining commands and writing Bash scripts **8**
6 The Conda ecosystem and project environments **11**
7 An introduction to Python **14**
8 Performance and I/O bottlenecks **16**
9 An integrated exercise and reproducibility **18**
References **22**

1 Basic concepts

Using an analysis tool requires more than knowing its command name. You also need to know which input it reads, where it writes its output and how it reports failure. The terminal makes these relationships visible. Reading three records from a sequence table, calculating their lengths and writing the result to a new file are separate operations. Defining them explicitly is a first step towards repeating the same analysis on another day.

1.1 What is behind the window?

An **operating system** manages hardware and the resources used by programs. At its centre, the **kernel** coordinates tasks such as sharing processor time, managing memory and accessing files. Strictly speaking, Linux is the name of a kernel. A **distribution**, such as Ubuntu, supplies that kernel together with command-line tools, a package manager and other software. In this note, Linux may also refer to this working environment as a whole, depending on the context.

A **terminal** is the interface that displays the text you enter and the programs' responses. A **shell** is the program that interprets the command line. It can read a command, expand variables and start another program. A running instance of a program is a **process**; each process has an identifier, uses resources and has a working directory. Starting the same Python script twice creates two separate processes.

A **command** is an instruction asking the shell to perform an operation. Its name may refer to a shell builtin or a program on disk. For example, `cd` changes the working directory within the shell, whereas `python` usually starts a separate program. The shell searches for program names in the directories listed in the `PATH` environment variable. The same word, `python`, can therefore launch a different interpreter depending on the software environment you have activated.[1]

1.2 Bash, PowerShell and Windows Terminal

Bash is a shell and scripting language widely used on Linux distributions. **PowerShell** is a different shell and language, often used to administer Windows but also available on other operating systems. PowerShell can pass objects between commands; the Bash pipelines in this note operate on programs' text or byte output. Similar command names do not imply identical options.[2]

Windows Terminal is the application that opens these shells in tabs. One tab may contain PowerShell and another Ubuntu's Bash shell. A similar-looking window does not mean that both tabs use the same language or file system.[3]

Practical note

Run boxes labelled **WINDOWS / POWERSHELL** in a PowerShell tab on Windows, and boxes labelled **LINUX / BASH** in native Linux or the Ubuntu tab inside WSL2. Boxes labelled **PYTHON** contain Python code; do not paste them directly into Bash. Output and configuration boxes are not commands to execute. Windows steps requiring administrator privileges are identified separately. Do not add the `$` or `PS>` prompts shown on your screen to the commands.

1.3 Reading a command

In the following command, `ls` is the program name, `-l` requests a detailed listing and `/tmp` is the directory to examine. An **argument** is an item of information passed to a program; options are one kind of argument.

LINUX / BASH

```
ls -l /tmp
```

The filenames in the output will depend on your computer. What you should expect here is a detailed listing of `/tmp`. A command can produce no output without failing. We will assess results using files, explanatory error messages and, later, exit codes.

Reflection. Why can you not simply continue using Linux's `cd /home` after closing the Ubuntu tab and opening a PowerShell tab in Windows Terminal? More than the appearance has changed: the shell interpreting the command and the file system initially available to it are different. You need to enter the Linux session explicitly.

2 Installing Linux and WSL2

With a native installation, the Linux kernel runs on the computer's hardware. In WSL2, it runs inside a lightweight virtual machine managed by Windows. Ubuntu's user-space tools are available through both routes; in WSL2, Windows and Linux share the same physical RAM, processor and disk. WSL1 has a different architecture; this note targets WSL2.[4]

2.1 Installing native Linux

If your computer already runs a supported Ubuntu release, you can proceed to verification without reinstalling it. For a new installation, follow Ubuntu's official desktop guide: download and verify the ISO for the appropriate architecture, prepare an installation USB drive, boot from it, test the hardware in the live session, choose a disk layout and create a user account.[5]

Writing an installation image changes the contents of the USB drive. Disk partitioning can also damage existing data, so make a backup and verify the target disk before installation. If you want to retain Windows, do not choose the option that erases the disk. Students unfamiliar with dual booting, BitLocker or disk partitions should install with an instructor's help. No single disk layout is imposed here. Start with a supported LTS release and record the version chosen for the course packages.

2.2 WSL2 and Ubuntu on Windows

Microsoft's simplified installation procedure is available on Windows 11 or Windows 10 version 2004/build 19041 and later. This historical minimum does not mean that an old Windows release is still supported today; use a current, supported installation. Hardware virtualization must be enabled in UEFI/BIOS. Ubuntu's WSL guide also explains the virtualization and Virtual Machine Platform requirements.[6, 7]

Open PowerShell from the Start menu **as administrator**. Check the current state first; an existing Ubuntu installation does not need to be removed or reinstalled. For a new installation, the official command enables the necessary Windows components and installs the distribution.

WINDOWS / POWERSHELL

```
wsl --list --verbose
wsl --list --online
wsl --install -d Ubuntu
```

If WSL is not yet installed, the first command may report that installation is required. Restart Windows after the last command if prompted. If you selected a differently named or versioned distribution from the online list, use that exact name in subsequent commands. The name `Ubuntu` may point to a different default release in the future; we will read the actual version from within Linux.

Open Ubuntu from the Start menu. On first launch, create a Linux username and password. This account is separate from your Windows account. It is normal for no characters or asterisks to appear while you enter the password. Run everyday analyses as this ordinary user; `sudo` grants temporary privileges only for necessary system operations.[8]

2.3 Verifying the installation

Run the following Windows commands in a normal PowerShell session. `--version` reports the WSL software version, whereas `--list --verbose` shows whether each distribution uses WSL1 or WSL2. These are different meanings of "version".[9]

WINDOWS / POWERSHELL

```
wsl --version
wsl --status
wsl --list --verbose
wsl -d Ubuntu
```

The distribution's `VERSION` column should contain `2`. A `Stopped` state does not mean that the installation is broken; the distribution may simply not be running yet. The last command opens a Linux session. Run the next commands **inside Ubuntu/Bash**; native Linux users can perform the same checks.

LINUX / BASH

```
whoami
pwd
cat /etc/os-release
uname -r
uname -m
bash --version
```

`whoami` reports the user, `pwd` the working directory, `/etc/os-release` the distribution and `uname -r` the kernel. `uname -m` reports an architecture such as `x86_64` or `aarch64`. This must match the Linux installer we download later.

If the distribution uses WSL1, back up your work and follow Microsoft's conversion instructions; conversion may take time. Changing the default for newly installed distributions does not convert an existing one. The commands illustrating this distinction run in PowerShell.[9]

WINDOWS / POWERSHELL

```
wsl --set-default-version 2
wsl --set-version Ubuntu 2
```

Check your understanding. Does `uname -r` tell you the Ubuntu version? No; it reports the kernel version. Read `/etc/os-release` for the distribution version and use `wsl --version` for the Windows-side WSL package.

3 Allocating RAM and processor resources to WSL2

Before installing packages, consider WSL2's resource limits. The computer's physical RAM and the memory that Linux is allowed to use may differ. Windows, the browser and other open applications also need memory. Giving all RAM to Linux does not increase the total available; it can increase pressure on Windows.

3.1 Distinguish capacity from current use

In the Performance tab of Windows Task Manager, inspect total RAM, current use, available memory and the number of logical processors. Total capacity is fixed, while availability changes with the applications you run. The **memory cap** selected for WSL2 is not a fixed block that is fully consumed as soon as it is assigned to analysis.

Microsoft's current documentation gives half of Windows RAM as the default memory limit, and all Windows logical processors as the default processor count. These are defaults, not recommendations for your workload.[10] For example, on a computer with 16 GB of RAM, an 8 GB WSL limit should be evaluated under actual use with the browser and other course software open. The same setting is not suitable for a machine with only 8 GB in total. If a large analysis alone exceeds the limit, reducing the input, running fewer concurrent jobs or using a suitable server may be more appropriate than adding swap.

3.2 WSL Settings and .wslconfig

Microsoft recommends the **WSL Settings** application in the Start menu on supported versions. You can adjust memory, processor and swap settings through this interface; the available fields may vary by version. For text-based configuration, use `.wslconfig` in your Windows user profile. The Linux file `/etc/wsl.conf` serves a different purpose. Settings in `.wslconfig` affect the virtual-machine resources shared by WSL2 distributions.[10]

The following PowerShell command opens the file in Notepad. If it already exists, read it and save a copy first; edit the relevant values without removing other settings. The filename must not become `.wslconfig.txt`.

WINDOWS / POWERSHELL

```
notepad "$env:USERPROFILE\.wslconfig"
```

This configuration illustrates the syntax for a **hypothetical teaching computer with 16 GB of RAM and at least 8 logical processors**. It is neither a measured optimum nor a recommendation for every student.

EXPECTED OUTPUT / FILE CONTENTS

```
[wsl2]
memory=8GB
processors=4
swap=2GB
```

`memory` sets the Linux virtual machine's RAM cap. Increasing it can help a job that does not fit in memory, but leaves less headroom for Windows. `processors` limits the available logical processors. Seeing four processors does not mean that every program automatically uses all four; the analysis tool's thread setting must be chosen separately. More threads do not always shorten runtime and may increase memory use.

Swap is disk space used to hold some memory pages. It is not equivalent to physical RAM. Repeatedly moving active data between RAM and disk can cause long delays. Nonzero swap use alone does not prove sustained slowdown; examine swap activity over time as well. An overly low memory cap or insufficient total memory can cause processes to be terminated for lack of memory.

3.3 Applying and verifying the settings

Save the file, then safely close running jobs and open files in your Linux sessions. The next command stops **all running WSL distributions**; do not use it in the middle of an analysis. Reopen Ubuntu afterwards.[9]

WINDOWS / POWERSHELL

```
wsl --shutdown
wsl -d Ubuntu
```

Check the resources visible in the new Linux/Bash session.

LINUX / BASH

```
free -h
nproc
swapon --show
```

`free -h` summarizes memory and swap in readable units. Its `total`, `free` and `available` fields differ; `available` estimates memory that could be used, including reclaimable cache. If the example's processor limit has taken effect, `nproc` should report 4, although additional process restrictions can change this value. Units and memory reserved by the kernel mean that exactly `8.0Gi` need not appear. If swap is disabled, `swapon --show` may produce no output.

If the expected change is absent, check that the file is in the Windows profile, has the correct extension and `[wsl2]` heading, and that the distribution actually uses WSL2. Then verify that you shut it down and reopened it. Native Linux does not use `.wslconfig`, but the same memory-monitoring commands remain useful.

Exercise. Why is running an analysis with 16 threads a poor starting point when WSL has been assigned 4 logical processors? Extra threads do not create processors; they can compete for processor time and use more memory. Start with a small input and limited concurrency, then measure.

4 The file system and basic terminal use

When a command cannot read a file, the first problem is often not its format: we may be working in the wrong directory. Before starting an analysis, we therefore need to establish where we are and which file we mean. From this section onwards, all commands run in Bash on native Linux or inside WSL2 unless stated otherwise.

4.1 Root, home and paths

The Linux file system is a tree starting at the root directory, `/`. A user's home directory is usually under `/home/kullanici`. In Bash, `~` represents the home directory; `.` is the current directory and `..` its parent. An **absolute path** begins at the root; a **relative path** is interpreted from the working directory. For example, `/home/ogrenci/proje/veri.tsv` is absolute, whereas `proje/veri.tsv` is relative.

The exercise below uses a separate area from your real data. If `terminal-pratik` already exists, choose a new practice-directory name instead of overwriting old files. The examples retain the Turkish edition's names so that both editions use identical commands: `veri` means data, `sonuc` means result and `betikler` means scripts.

LINUX / BASH

```
cd ~
mkdir terminal-pratik
cd terminal-pratik
mkdir veri sonuc betikler
pwd
ls
```

`mkdir` creates a directory. If the directory already exists, the first `mkdir` fails; pause and inspect its contents. The `ls` output should include `betikler`, `sonuc` and `veri`, although their order can vary with locale settings. In the terminal, Tab completes paths and the up arrow recalls command history. `Ctrl+C` usually interrupts the foreground job; it is not a shortcut for saving a file.

Linux filenames are case-sensitive: `Veri.tsv` and `veri.tsv` are different names. Spaces are allowed, but require quoting. Short, meaningful names without spaces reduce typing mistakes in these exercises; do not rename a real sample identifier merely for convenience.

4.2 Where should files go in WSL?

WSL2's Linux file system usually uses ext4 inside a virtual disk. This is where `~/terminal-pratik` resides. By contrast, `/mnt/c/Users/ogrenci` accesses files on Windows drive C. Being able to see both from Bash does not imply identical access paths or permission behaviour. Microsoft recommends keeping projects processed with Linux tools in the Linux file system.[11]

In WSL, the following command opens the current Linux directory in Windows File Explorer. It illustrates calling a Windows program from a Linux session; native Linux users do not need this command.

LINUX / BASH

```
explorer.exe .
```

You can also use `\\wsl.localhost\Ubuntu\home` in Windows File Explorer. Adjust the distribution name if yours differs. Do not manually edit Linux files through the distribution's Windows application-data directory; use the supported sharing path. When copying input from Windows to the Linux project directory, preserve the source and verify the copy's contents. Quoting a path handles spaces; it does not prove that the file exists there.

4.3 Creating and inspecting a small table

Let us write the labels and lengths of three synthetic records. `printf` produces formatted text; `\t` is a tab and `\n` a newline. `>` redirects output to a file and truncates any existing file with that name. Run this only in the new practice directory, before the target exists.

LINUX / BASH

```
printf 'id\tlength\nseq_a\t8\nseq_b\t12\nseq_c\t8\n' > veri/uzunluk.tsv
cat veri/uzunluk.tsv
head -n 2 veri/uzunluk.tsv
wc -l veri/uzunluk.tsv
```

`cat` displays the entire contents, while `head -n 2` shows the first two lines. Inspecting a few lines first is more useful for a large file. Here, `wc -l` reports 4: three data rows and one header. A line count is not automatically a sample count. In files lacking a final newline, `wc -l` may also count the last visible record differently from what you expect.[12]

EXPECTED OUTPUT / FILE CONTENTS

```
id      length
seq_a   8
seq_b   12
seq_c   8
```

The spacing in the box aligns columns visually; the real file uses tabs as delimiters. Browse it with `less veri/uzunluk.tsv` and press `q` to quit. Learn each program's own exit shortcut rather than relying on `Ctrl+C`.

4.4 Searching, selecting fields and finding help

`grep` searches for patterns within lines; `find` searches a directory tree using properties such as names and file types. The first search below uses a literal string; in the second, quotes prevent the shell from expanding the asterisk prematurely.

LINUX / BASH

```
grep -F 'seq_b' veri/uzunluk.tsv
find veri -type f -name '*.tsv'
cut -f 2 veri/uzunluk.tsv
```

`grep -F` treats the pattern as plain text rather than a regular expression. It finds the `seq_b` line here, but does not by itself test for an exact identifier match in the first column. `cut -f 2` extracts the second tab-separated field, including the header. Such simple splitting rules may be unsuitable for general CSV files containing quoted commas; we will use Python's `csv` module.[13, 14]

Help text is the first place to check what an option does in the installed version. `man` opens an installed system manual; `help` is for Bash builtins.

LINUX / BASH

```
ls --help
man ls
help cd
type cd
command -v python
```

Press `q` to leave `man`. If a Python environment has not yet been installed, the last command may produce no output. Knowing the path selected for a program helps identify a version coming from the wrong environment.

4.5 Permissions and safe file operations

In `ls -l` output, `r`, `w` and `x` indicate read, write and execute permissions for the owner, group and other users. On a directory, `x` allows traversal and access to names within it. `chmod u+x betik.sh` adds execute permission only for the owner; giving every file `777` permissions is not an appropriate fix for a permission error. Permissions on Windows drives accessed through WSL also depend on Windows permissions and mount settings.[12, 15]

List the source and destination before copying or moving a file. In this example, `-i` asks before overwriting an existing target. The deletion applies only to the copy created in this step.

LINUX / BASH

```
ls -l veri/uzunluk.tsv
cp -i veri/uzunluk.tsv sonuc/uzunluk-kopya.tsv
cmp veri/uzunluk.tsv sonuc/uzunluk-kopya.tsv
mv -i sonuc/uzunluk-kopya.tsv sonuc/deneme.tsv
rm -i sonuc/deneme.tsv
```

`cmp` is silent and returns success when the files are identical. `rm` generally does not move files to a recycle bin, so avoid deletion with broad wildcards. Creating analysis files with `sudo` can also cause ownership problems later. Check the path, ownership and actual requirement before increasing privileges.

Exercise. Why might `cat veri/uzunluk.tsv` fail when you are already inside `veri`? The relative path now means `veri/veri/uzunluk.tsv`. Confirm your location with `pwd`; here, use `cat uzunluk.tsv` or return to the project root.

5 Combining commands and writing Bash scripts

Running commands individually makes it easier to learn by observing each step. Once a repeated sequence is clear, saving it in a file reduces the risk of copying incomplete commands or the wrong order from history. A script explicitly defines which operations run on which inputs, and in what order.

5.1 Three streams and an exit code

A program has **standard input** (0), **standard output** (1) and **standard error** (2). Separating normal output from diagnostic messages prevents error text from entering a data file. `<` supplies a file as input; `>` writes output, `>>` appends it and `2>` sends the error stream to a separate file.[16]

Run the following example in `~/terminal-pratik`. With `wc -l < file`, `wc` reads standard input rather than a named file, so it prints only the count.

LINUX / BASH

```
cd ~/terminal-pratik
wc -l < veri/uzunluk.tsv
cat veri/yok.tsv > sonuc/okuma.txt 2> sonuc/hata.txt
printf '%s\n' "$?"
cat sonuc/hata.txt
```

The first result is 4. The second command deliberately refers to a missing input; expect a file-not-found message on standard error and a nonzero code in the immediately following `$?`. The language of the message and exact code may vary by tool. Redirection happens before the program starts, so even a failed command can create an empty `okuma.txt`. A file's existence does not establish that an analysis completed successfully.

An exit code of zero usually indicates success; the meaning of other values depends on the program. For example, `grep` returns 1 when it finds no match, which does not mean the input is malformed. `$?` holds only the last command's code. Running another command first loses the previous result.

5.2 Pipes and dependencies between steps

A **pipe** connects one program's standard output to the next program's standard input. The following pipeline skips the header, extracts the length column, sorts numerically and counts repeated values.

LINUX / BASH

```
tail -n +2 veri/uzunluk.tsv | cut -f 2 | sort -n | uniq -c
```

Expect two records of length 8 and one of length 12. Sorting comes first because `uniq` combines only adjacent identical lines. Inspect intermediate outputs separately before extending a pipeline; a chain that selects the wrong column can still finish without an error.

By default, a Bash pipeline's exit code comes from its last command. `set -o pipefail` makes failures earlier in the pipeline affect that result too. `&&` runs the next command only if the preceding one succeeds; `||` runs it only if the preceding one fails.[17]

LINUX / BASH

```
set -o pipefail
cat veri/yok.tsv | wc -l
printf '%s\n' "$?"
```

`wc` may print zero; this does not show that the real input contains zero records. The pipeline's nonzero exit code and the `cat` error show that the input could not be read. This is why we need to inspect contents as well as exit codes.

5.3 Variables, quotes and arguments

Do not put spaces around the equals sign in a variable assignment. Double quotes expand a variable while keeping a value containing spaces together as one argument. Single quotes preserve their contents literally.[18] The variable name `girdi`, retained from the Turkish edition, means input.

LINUX / BASH

```
girdi="veri/uzunluk.tsv"
printf '%s\n' "$girdi"
printf '%s\n' '$girdi'
wc -l "$girdi"
```

The first `printf` prints the path; the second prints the literal text `$girdi`. Make a habit of using file paths as `"$girdi"`. The quoted string `"~/veri"` does not expand to the home directory; use `"$HOME/veri"` when a variable is needed. `$(command)` captures a command's output; this syntax actually executes the command.

5.4 Moving commands into a .sh file

In Bash, open the file with `nano betikler/say_satir.sh`. Enter the contents below; save and exit Nano with `Ctrl+O`, Enter and `Ctrl+X`. If `nano` is unavailable, use the Ubuntu tool-installation step in the next section. Save plain text with LF line endings.

LINUX / BASH

```
#!/usr/bin/env bash
set -euo pipefail

if [[ $# -ne 1 ]]; then
    printf 'Usage: bash say_satir.sh input.tsv\n' >&2
    exit 2
fi

girdi="$1"
if [[ ! -f "$girdi" ]]; then
    printf 'Input file not found: %s\n' "$girdi" >&2
    exit 1
fi
wc -l < "$girdi"
```

The first line, the **shebang**, requests Bash when the file is executed directly. `$#` is the argument count and `$1` the first argument. `[[ ... ]]` tests a condition; `-f` checks for a regular file. The block starting with `if` ends with `fi`. Checking the argument count first prevents reading an unset `$1` when `set -u` is enabled.

`set -e` stops the script on some command failures, with exceptions such as conditions and certain command combinations. `-u` exposes unset-variable use, and `pipefail` exposes failures within pipelines. These options do not validate scientific results or replace explicit input checks.[19]

LINUX / BASH

```
bash -n betikler/say_satir.sh
bash betikler/say_satir.sh veri/uzunluk.tsv
chmod u+x betikler/say_satir.sh
./betikler/say_satir.sh veri/uzunluk.tsv
```

`bash -n` checks syntax without running the analysis. The two execution methods shown here both produce 4. `bash file.sh` does not require execute permission; `./file.sh` launches the script directly and does require it. Using `sh file.sh` may send Bash-specific syntax to the wrong shell.

5.5 Loops and readable scripts

A loop repeats a similar operation over multiple inputs. This example prints the name and line count of each TSV file separately. The `[[ -f ... ]]` check prevents the unexpanded text `*.tsv` from being treated as a file when nothing matches.

LINUX / BASH

```
for girdi in veri/*.tsv; do
    [[ -f "$girdi" ]] || continue
    printf '%s: ' "$girdi"
    wc -l < "$girdi"
done
```

Use meaningful variable names, consistent indentation and explicit input/output paths rather than compressing many operations onto one line. A \ at the end of a long command continues it on the next line; do not place a space after it. A script should be readable by a person; shorter does not automatically mean better.

Exercise. Call the script with no argument and then with a nonexistent file. Expect code 2 for incorrect usage and 1 for a missing file. Finally, verify the result 4 with the real input. An expected failure teaches the script's behaviour as well as a successful example does.

6 The Conda ecosystem and project environments

For a command to work, its dependencies must be compatible as well as the program itself. One analysis may need an older Python release, another a different library. An **environment** is a directory holding software and its dependencies for a particular task. Creating an environment does not create a virtual machine; it changes neither the operating-system kernel nor the CPU architecture.

6.1 Which name refers to which role?

Conda manages packages and environments; it can install tools written in other languages as well as Python packages. **Miniconda** is a small installer supplying Conda and its basic components. **Miniforge** is a separate distribution that supplies Conda and Mamba with an initial configuration centred on conda-forge. You do not need both.[20, 21]

Mamba is a separate command-line tool for managing environments with Conda packages. **libmamba** is a library used in dependency solving. Conda uses it through the `conda-libmamba-solver` plugin; the official documentation identifies libmamba as the default solver since Conda 23.10. Do not compare current Conda and Mamba solely using the speed of Conda's older solver. Downloading, unpacking and writing packages to disk also contribute to installation time.[22, 23]

6.2 Installing Miniconda inside Linux or WSL

If you use WSL, run these steps **inside Ubuntu/Bash**. The Windows `.exe` installer does not install into this Linux environment. Native Ubuntu users follow the same Linux steps. First refresh Ubuntu's package list and install the small helper tools.[8]

LINUX / BASH

```
sudo apt update
sudo apt install curl ca-certificates nano less python3
uname -m
```

`apt update` refreshes the package list from repositories; it does not upgrade all installed packages. `apt install` installs system tools. `sudo` is required here for the system package manager, but not when installing Miniconda in your home directory. We will not replace system Python files with Conda.

The next example is for Linux where `uname -m` reports `x86_64`. If it reports `aarch64`, check the official ARM64 installer's hardware requirements and replace `x86_64` in the filename with `aarch64`. An ARM package's availability does not imply that every bioinformatics tool is available for that architecture.[20, 26]

LINUX / BASH

```
cd ~
mkdir miniconda-kurulum
cd miniconda-kurulum
curl -fLO https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
sha256sum Miniconda3-latest-Linux-x86_64.sh
```

Compare the downloaded file's checksum with the SHA-256 entry for the *same file* in the official Miniconda archive. Do not execute it if they differ. `latest` is a moving target; retain the downloaded version and installer file in your installation record. Also check your institution's terms for using the package repository. Start the installer after verification.[20]

LINUX / BASH

```
bash Miniconda3-latest-Linux-x86_64.sh
```

Read the licence, consider the proposed location in your home directory and respond appropriately to the `conda init` prompt, which modifies Bash startup configuration. Do not run a new installer over an existing Miniconda installation. Close and reopen the Ubuntu/Bash terminal when installation finishes.

If you declined initialization, verify that installation used the default location before using the following path.

LINUX / BASH

```
~/miniconda3/bin/conda init bash
```

Open a new Bash session after this step too. Inspect the version, configuration and package list to verify the installation.

LINUX / BASH

```
conda --version
conda info
conda config --show solver
conda list
```

`conda info` should report a Linux platform. An older configuration may select the `classic` solver; distinguish the default from the actual setting. Keep the base environment for maintaining the package manager rather than filling it with the course's analysis packages.

6.3 Using Mamba from Miniconda

Mamba's official recommendation for new installations is Miniforge. For a student starting with Miniconda, we install Mamba here into a separate management environment containing only conda-forge packages. This lets us learn the Mamba command without mixing channel packages into Miniconda's base environment. conda-forge also documents isolated environments for trying a different package source.[24, 25]

LINUX / BASH

```
conda create -n mamba-tools --override-channels -c conda-forge mamba
conda run -n mamba-tools mamba --version
```

`-n` names the environment. `--override-channels` excludes the user's other channels from this operation. `conda run` executes a command inside the named environment without changing the active environment. Read the proposed installation plan before approving it. If `mamba-tools` already exists, inspect its contents and version instead of deleting it.

Now create the teaching environment with Mamba. `--no-capture-output` displays the installation plan and confirmation prompt directly in the terminal.[35] `-p` installs the environment at an explicit path, keeping the management and teaching environments separate. Python 3.12 is the version family chosen for this exercise, not a claim about the latest release.

LINUX / BASH

```
conda run --no-capture-output -n mamba-tools mamba create \
    -p "$HOME/conda-envs/terminal-bio" \
    --override-channels -c conda-forge -c bioconda \
    --strict-channel-priority python=3.12 seqkit
conda activate "$HOME/conda-envs/terminal-bio"
python --version
command -v python
seqkit version
```

`command -v python` should report a path inside the active teaching environment. `seqkit` demonstrates installing a Bioconda tool; the TSV exercise in this note uses only the Python standard library. If a tool is unavailable for an architecture, do not force another architecture's package; check the supported platforms.

Students starting with Miniforge already have `mamba`; start the same creation command directly with `mamba create`, without a separate management environment. These are alternative routes. Adding startup lines from two installations indiscriminately to the same shell makes it unclear which Conda is running.

6.4 Channels, versions and pip

A **channel** is a repository from which packages are downloaded. conda-forge supplies general dependencies and Bioconda supplies bioinformatics packages. In the order shown, conda-forge has higher priority. `--strict-channel-priority` excludes lower-channel candidates when the same package exists in the higher-priority channel, avoiding arbitrary combinations of whichever versions look newest. Bioconda's current recommendation does not include `defaults`. Mixing conda-forge and `defaults` packages in one environment is not a supported transition route.[26, 25]

Use `conda list` to inspect versions and `conda deactivate` to leave the active environment. Try package changes in a new test environment instead of modifying the environment of a completed analysis. Below, the first command only displays the proposed changes; the second applies them. Run both in Linux/Bash with the teaching environment active.

LINUX / BASH

```
conda install --dry-run --override-channels \
    -c conda-forge -c bioconda --strict-channel-priority python=3.12
conda install --override-channels \
    -c conda-forge -c bioconda --strict-channel-priority python=3.12
```

`python=3.12` constrains the version family but does not pin the exact patch release or dependency builds. The solver selects a compatible combination. Even a successful `--dry-run` does not mean that the package has been installed or tested in an actual analysis.

pip is a Python package manager; it does not manage all Conda packages. If needed, install Conda dependencies first, then use `python -m pip` with the active environment's Python. Do not use `sudo pip` or `pip --user` to place packages outside this project environment. Packages added with pip are not always fully accounted for by Conda's solver; for later changes, recreating the environment from its specification may be clearer. This exercise needs no pip packages.[27]

6.5 Recreating an environment

An environment name alone is not a reproducibility record. Create the following files in the project directory only when files with these names do not already exist; `>` can overwrite them.

LINUX / BASH

```
conda env export --from-history > environment-history.yml
conda env export > environment-full.yml
conda list --explicit > environment-explicit.txt
```

Exporting from history records the top-level packages you requested. It is more portable but does not guarantee an identical package combination. The full YAML also includes resolved dependencies and may contain platform-specific builds and a local `prefix`. An explicit package list reinstalls concrete builds on the same platform; it is not a general lock covering pip-installed packages. Inspect local paths and private channel details before sharing.[27, 23]

LINUX / BASH

```
conda env create -n terminal-tekrar -f environment-history.yml
conda create -n terminal-ayni-platform --file environment-explicit.txt
```

These are two different restoration methods; you need not install both during the exercise. The same `linux-64` package list can be used on compatible x86_64 Linux and WSL2 systems; do not force it onto ARM64 or Windows environments. Verify versions and the small exercise's output again after installation. Package availability does not automatically establish equivalence of operating-system libraries or results.

7 An introduction to Python

Bash is useful for connecting files and programs. When we want to calculate a new quantity from each row of a table, Python makes the data structure easier to express explicitly. The aim here is not advanced programming, but reading a small text file, performing a calculation and producing a new table.

7.1 The interpreter and a script

The **Python interpreter** is the program that runs Python code. In Linux/Bash, with the teaching environment active, open the interactive interpreter using this command.

LINUX / BASH

```
python
```

When the `>>>` prompt appears, you can enter Python code; do not type the prompt itself. Try these lines in order.

PYTHON

```
2 + 3
len("ACGT")
exit()
```

The first two responses are 5 and 4. `exit()` returns to Bash. A `.py` file, by contrast, stores code for execution with `python dosya.py`. In a script, you generally need `print` to display a value; distinguish this from the interactive interpreter displaying the value of an expression.[28]

7.2 Values, names and data structures

A variable name is bound to a value. Text has type `str`, integers `int`, floating-point numbers `float` and true/false values `bool`. Python uses a full stop as the decimal separator; do not substitute a comma used in some languages' prose.

PYTHON

```
sequence = "ACGTACGT"
length = len(sequence)
gc_count = sequence.count("G") + sequence.count("C")
gc_percent = 100 * gc_count / length
is_long = length >= 8
print(length, gc_count, gc_percent, is_long)
```

The output is `8 4 50.0 True`. `=` assigns a value; `==` tests equality. When turning a base count into a percentage, define the denominator explicitly; we will treat ambiguous bases separately shortly.

A **list** stores ordered values; a **dictionary** associates a key with a value. A dictionary can distinguish a record's name from its sequence. Python indexing starts at zero.

PYTHON

```
lengths = [8, 12, 8]
record = {"record_id": "seq_a", "sequence": "ACGTACGT"}
print(lengths[0])
print(record["record_id"])
print(record["sequence"][:4])
```

The results are 8, `seq_a` and `ACGT`, respectively. `[:4]` takes the first four characters; the end index is excluded. Requesting a key absent from the dictionary raises an error. Errors often expose incorrect assumptions about the data.

7.3 Indentation, conditions, loops and functions

Indentation defines Python's code structure; it is not decoration. Indent the lines belonging to a condition or loop by four spaces, and do not mix tabs and spaces. Write the following code into a `.py` file or enter it as a block in the interpreter. In interactive use, a blank line signals the end of the block.[29]

PYTHON

```
lengths = [8, 12, 8]
for length in lengths:
    if length >= 10:
        print(length, "keep")
    else:
        print(length, "short")
```

The output is `8 short`, `12 keep`, then `8 short`. Lowering the threshold from 10 to 8 keeps all three records. This threshold illustrates programming; it is not proposed as a biological quality criterion.

A **function** is a named operation that can be called again. The function below accepts a sequence and returns its length. `return` passes the result to the calling code; `print` writes it to the screen.

PYTHON

```
def sequence_length(sequence):
    return len(sequence)

result = sequence_length("ACGT")
print(result)
```

The result is 4. You do not need to turn every small statement into a separate function; repeated operations, or operations made clearer by a name, are good candidates.

7.4 Reading and writing files

From `~/terminal-pratik` in Bash, create a file with `nano betikler/oku.py` and save the following Python code. The `with` statement closes the file when its block ends. `encoding="utf-8"` explicitly selects the character encoding.[30]

PYTHON

```
with open("veri/uzunluk.tsv", encoding="utf-8") as handle:
    for line in handle:
        print(line.rstrip("\n"))
```

Return to Bash and run it.

LINUX / BASH

```
python betikler/oku.py
```

A script does not automatically change into the directory containing it. A relative input path is interpreted from the working directory where you started Python. If you run the same code from `betikler`, the `veri` directory may not be found. This is why exercises need a documented working directory.

Choose the `open` mode when writing a file. `"w"` truncates existing contents; `"a"` appends; `"x"` creates a new file and fails if the name already exists. We use `"x"` in the exercise to avoid accidentally overwriting results.

PYTHON

```
with open("sonuc/mesaj.txt", "x", encoding="utf-8") as handle:
    handle.write("Three synthetic records\n")
```

Running this code a second time should raise a file-exists error. Repeating an analysis need not silently erase its previous result; you can choose a new output name.

7.5 Interpreting a table row by row

`csv.DictReader` turns each row into a dictionary accessible by column names. For TSV, the delimiter must be `"\t"`; the default comma delimiter would read the columns incorrectly. Numbers read from the file begin as text and need conversion for arithmetic.[14]

PYTHON

```
import csv

with open("veri/uzunluk.tsv", encoding="utf-8", newline="") as handle:
    reader = csv.DictReader(handle, delimiter="\t")
    for row in reader:
        length = int(row["length"])
        if length >= 10:
            print(row["id"], length)
```

The expected output is `seq_b 12`. `import` makes the standard-library module available; `csv` needs no pip installation. `int("sekiz")` is not a valid conversion: the Turkish word for eight is still text. Silently replacing such an error with zero can make malformed input appear to be valid data.

Exercise. Read the same table with a threshold of 8; expect three records. Then change one length to `?` in a copy of the table. Expect conversion to fail instead of producing an incorrect numerical result. Work on a copy without changing the original practice input.

8 Performance and I/O bottlenecks

When an analysis slows down, asking how many cores it uses is only a starting point. The program may be calculating, waiting for disk data, downloading from the network or struggling to fit in memory. A **bottleneck** is the resource limiting the speed of a particular stage; it can change between steps of a workflow.

8.1 Distinguishing CPU, RAM, disk and network

The CPU performs calculations. RAM holds active data and program state. Disk provides persistent storage, while the network provides access to remote resources. Downloading a reference file may be network-limited, decompressing it may depend on CPU and disk, and loading a large index may depend on disk and RAM. These categories are not mutually exclusive.

Many small files increase name lookups, opening, closing and metadata operations even when their total size is small. Reading a large file **sequentially** from beginning to end differs from **random access** to small pieces at different positions. The operating system can retain recently read data in RAM as a cache; a faster second run does not mean the program has changed.

Compression reduces the bytes stored or transferred, but decompression consumes CPU time. Compressed input is therefore not always slower or always faster. Giving more CPU resources to a job that relies heavily on swap may not solve its problem. Inspect Linux memory and process summaries with the following commands. Press `q` to leave `top`; `vmstat 1 5` prints five observations.

LINUX / BASH

```
top
free -h
df -h .
du -sh veri
vmstat 1 5
```

`df` summarizes file-system capacity; `du` summarizes space used by the specified directory. The `si`/`so` fields in `vmstat` help monitor swap-in/swap-out activity. Its first row includes averages since boot and should not be interpreted like subsequent intervals. Do not diagnose a bottleneck conclusively from a single snapshot. In WSL, also inspect Windows Task Manager.[31, 12]

8.2 Three distinct execution and storage paths

On **native Linux**, an application runs through the native Linux kernel and its mounted file system. That file system need not be ext4; network storage is also possible. In **WSL2's Linux directories**, the application runs under a Linux kernel, usually accessing an ext4 virtual disk. When **WSL accesses Windows files**, a mount such as `/mnt/c` introduces a cross-operating-system file-sharing layer.[4, 11]

These paths cannot be reduced to one "Linux versus Windows" comparison. Path choice matters particularly for workloads involving small files and many metadata operations. Windows file-sharing implementations can vary with WSL versions and settings; do not retrospectively apply experimental sharing options in current documentation to older measurements.[10] A common starting point for this course is to run Linux tools on a project in the Linux home directory and copy the required final outputs to Windows.

8.3 What can primary studies tell us?

The studies below do not test the same question under the same conditions. Read hardware, software and access-path information together; we do not fill unreported details with assumptions.

A direct WSL2 comparison. Anshori and colleagues (2026) compare CPU-based language-model inference on an Acer Aspire Lite 14 with an Intel i3-1215U and 8 GB DDR5. They report Windows 11 Home, WSL 2.6.3.0/kernel 6.6.87.2-1, Ubuntu Server 24.04.3 LTS on both native and guest installations, and Ollama 0.24.0. Llama3.2:3B, Qwen2.5:3B and Phi3.5:3.8B receive two warm-up runs and ten measurements, with models preloaded into memory. For example, mean latency for Llama3.2 with a light prompt is reported as 12.73 s on WSL2 and 10.83 s on native Linux. This is neither sequence analysis nor a file-reading measurement. The article does not specify the file-system type, whether model files reside on the Linux or Windows side, or the WSL memory cap; file-system effects cannot be isolated from these results.[32]

Technical measurements of scientific computing. Kinghorn's (2022) primary technical report, which is not a peer-reviewed article, compares Ubuntu 22.04 with WSL2/Ubuntu 22.04 on Windows 11 using the same Ryzen 5800X/64 GB DDR4 computer. WSL memory is increased to 56 GB. HPL, HPCG and NAMD are built for Zen3, prepared in Docker and run with enroot. HPL shows a more pronounced difference, whereas HPCG and NAMD results are close. AOCC 3.2 and AMDFFTW 3.2 are specified; the Windows build, WSL kernel version, complete application versions and file system/working path are not. The report illustrates workload dependence; its containerized computation results cannot be applied directly to all Bash or file operations.[33]

Separating the Docker layer. Sobieraj and Kotyński (2024) compare Docker workloads on Windows 10 22H2, T2-adapted Ubuntu 22.04/6.4.8 and macOS 13.5.1, using a MacBook Pro with an Intel i5-8257U, 16 GB RAM and 256 GB NVMe storage. The article calls the Docker version "4.20", without clearly distinguishing Engine from Desktop in that record. Tests cover computation, Sysbench, files and networking; the Windows path includes both Docker Desktop and WSL2. The WSL kernel, host file system and mount location are insufficiently specified. These results concern Docker deployments and cannot supply a slowdown factor for WSL2 without containers.[34]

This limited evidence from different workloads does not support a universal WSL2 slowdown percentage. WSL1 measurements concern a separate architecture. Native Linux–WSL2 performance has not been

measured for this note; the numbers above are reported by the cited study. Estimating runtime on your computer requires a small but representative trial close to your own workload.

8.4 Designing a fair small comparison

First verify that outputs are correct and equivalent; producing an incorrect answer faster does not make a method superior. Use the same machine, input, tool version, parameters and thread count. Record the Windows build, WSL version, distribution, kernel, memory/swap limits and the actual mount location of the file. These commands inspect the relevant Linux-side path.

LINUX / BASH

```
findmnt -T .
uname -r
uname -m
```

`findmnt` reports mount information; it does not fully describe the virtual disk's physical placement on Windows. Comparing two directories within WSL2 can examine storage-path effects, but is not itself a comparison with native Linux.

Once the exercise in Section 9 is ready, you can try timing syntax as follows. This illustrates a method; no previously measured runtime is supplied. The output name must be new.

LINUX / BASH

```
time python scripts/02_sequence_summary.py \
    work/input.tsv work/timed-summary.tsv 8
```

`real` summarizes elapsed wall time, `user` CPU time in user code and `sys` CPU time in the kernel. In multithreaded programs, CPU time can exceed wall time. These three records are so small that interpreter startup overhead may dominate; they are unsuitable for a meaningful performance ranking. In a larger trial, separate the first run from later cached runs, report several repetitions and their distribution, and try to hold background load and power state constant. Do not indiscriminately clear system-wide caches for a comparison.

Exercise. File opening takes a long time in a project under `/mnt/c`, while CPU use is low. What would you test before concluding that more cores are needed? Try the same command on a copy of the same small input in the Linux home directory, comparing output equality, runtime and disk activity. A difference provides evidence about the access path; it does not isolate every cause by itself.

9 An integrated exercise and reproducibility

We can now combine file management, Bash and Python around a small question: what are the lengths and GC percentages among called bases of three synthetic sequences, and which records meet a selected length threshold? This calculation is not a quality assessment or species identification. The aim is to move from an explicit definition to a result that can be checked.

9.1 Working directory and input

Copy the accompanying `main_scripts/terminal_basics/` directory into your Linux home as `terminal-proje`. Use the following commands from the notes repository's root only when the destination does not yet exist. If it exists, preserve its contents and choose a new name.

LINUX / BASH

```
cp -R main_scripts/terminal_basics "$HOME/terminal-proje"
cd "$HOME/terminal-proje"
ls scripts
conda activate "$HOME/conda-envs/terminal-bio"
```

The last command selects the environment from Section 6. If you downloaded the repository to Windows, copy it from within WSL and keep the final working directory in your Linux home. If you downloaded only the PDF, obtain the companion scripts directory too; the filenames below refer to its contents.

Phase 1 creates a *new* directory named `work` and the synthetic input. Run each phase separately and inspect its result.

LINUX / BASH

```
bash scripts/01_prepare.sh work
cat work/input.tsv
```

EXPECTED OUTPUT / FILE CONTENTS

```
record_id   sequence
seq_a       ACGTACGT
seq_b       GGGCCCAT
seq_c       ATATATNN
```

The actual file uses tab delimiters. `01_prepare.sh` does not overwrite an existing working directory. This input is a two-column teaching table, not FASTA. In FASTA, headers begin with `>` and a sequence can span multiple lines; do not pass a FASTA file directly to this script.

9.2 Defining the quantity and the parameter's effect

Sequence length counts all characters, including `N`. The `called_bases` field counts only A, C, G and T. We define GC percentage using these called bases:

$$\text{GC percentage} = 100 \times \frac{n_G + n_C}{n_A + n_C + n_G + n_T}.$$

This teaching definition excludes ambiguous bases from the denominator. Another tool may use total length; read the definition before comparing identically named columns. An all-`N` sequence has a zero denominator, so we report `NA` rather than 0. Zero GC and unknown GC convey different information.

`min_length` sets the total length required to retain a record. We choose 8 here to examine all three small sequences together, not as a biological threshold. `keep=yes` means only that the length condition is satisfied. A record containing ambiguous bases can also satisfy it; this column is not a quality endorsement.

The following function performs the central calculation in phase 2. The complete script also checks headers, unique identifiers, row structure, the permitted alphabet and output collisions. Lowercase letters and other IUPAC ambiguity codes are outside this teaching input's scope; they are not silently corrected.

PYTHON

```
def sequence_stats(sequence):
    if not sequence or any(base not in "ACGTN" for base in sequence):
        raise ValueError("Expected a nonempty uppercase A/C/G/T/N sequence")
    called = sum(sequence.count(base) for base in "ACGT")
    gc = sequence.count("G") + sequence.count("C")
    percent = 100 * gc / called if called else None
    return len(sequence), called, percent
```

`sum(... for ...)` is a compact iteration that adds the count for each base. `any(...)` tests whether at least one disallowed character occurs. `None` represents the absence of a computable percentage and becomes `NA` when written to the file. Returning three values keeps sequence length and called-base count distinct.

9.3 Running the calculation and checking the result

Run phase 2 from the project directory in Linux/Bash. The input path, output path and threshold are three separate arguments.

LINUX / BASH

```
python scripts/02_sequence_summary.py work/input.tsv work/summary.tsv 8
cat work/summary.tsv
python scripts/03_check_example.py work/summary.tsv
```

Phase 2 should print `Wrote 3 records`; the checker should print an `OK` message confirming agreement with the hand calculation for the three synthetic records. The expected table is:

EXPECTED OUTPUT / FILE CONTENTS

```
record_id  length  called_bases  gc_percent  keep
seq_a      8       8             50.000      yes
seq_b      8       8             75.000      yes
seq_c      8       6              0.000      yes
```

For `seq_a`, four GC bases divided by eight called bases gives 50%. For `seq_b`, six/eight gives 75%; for `seq_c`, zero/six gives 0%. Across the records, total length is 24, called-base count is 22 and GC count is 10. A pooled percentage would be $100 \times 10/22 \simeq 45.45$. The simple mean of the row percentages is 41.67; these quantities differ because their denominators differ. The script produces only row-level results here.

The checker is not a general-purpose sequence validator; it tests the hand-calculated expectation for these three records at `min_length=8`. It checks columns, record identity/order, counts and finite GC values. Reusing the same expected table after changing the threshold or input would be incorrect.

9.4 Guided experiments and negative controls

Increase the threshold to 9 and write to a new output name. All three rows remain, but every `keep` value becomes `no`; lengths and GC percentages do not change. Separating the filtering decision from the calculated quantities makes the parameter's effect clear.

LINUX / BASH

```
python scripts/02_sequence_summary.py work/input.tsv work/summary-min9.tsv 9
cat work/summary-min9.tsv
```

Passing this file to `03_check_example.py` should fail because the checker's fixed teaching expectation uses threshold 8. The failure reflects a changed retention decision, not a changed biological calculation.

Do the other exercises on input copies. Adding a `G` to the sequence containing `N` increases both its length and called-base count; calculate the new GC denominator by hand. Adding `X`, duplicating an identifier or removing a field should stop the script. For an all-`N` record of length 8, expect `called_bases=0`, `gc_percent=NA` and `keep=yes`. This shows that a length filter does not resolve ambiguous bases.

In a copy of the result file, replace `50.000` with `51.000` and run the checker. Expect a nonzero exit and a GC-mismatch message. A non-finite value such as `NaN` must also fail. Success on correct input alone does not show that a check can detect errors; this is why negative examples are necessary.

9.5 Keeping a reproducibility record

Reproducible work requires preserving code, input, parameters and environment information together. Command history is useful but insufficient on its own. The same command can behave differently when launched from another directory, in another environment or against a file that has since changed.

Keep a short project record of which script you called, the working directory, input version, selected threshold, expected output and reasons for changes. These Linux/Bash commands create a new environment record; choose a new name if the file already exists.

LINUX / BASH

```
{
    uname -srm
    bash --version
    python --version
    conda --version
    conda list
} > work/environment.txt 2>&1
sha256sum work/input.tsv work/summary.tsv > work/checksums.txt
```

`2>&1` directs standard error to the current destination of standard output; here it keeps version messages in one record. Do not confuse data output with this diagnostic log. Checksums track whether files have changed; they do not establish biological correctness. Keep working records and machine-specific paths local, and review documents intended for sharing separately.

WSL users should also record the PowerShell outputs of `wsl --version` and `wsl --list --verbose`, along with the chosen memory, processor and swap limits. To recreate the environment, choose a suitable export method from Section 6. This exercise has no stochastic step, so it needs no random seed. Package-version records, file checksums and result checks serve complementary purposes.

9.6 Working through common problems

Command not found. First check that you are in the correct shell and that the environment is active. In Linux/Bash, `command -v python` and `conda info --envs` can expose the wrong Python selection. Adding a Windows Python path to the Linux environment is not the solution.

File not found. Use `pwd` and `ls` to check the working directory, filename case and path quoting. In WSL, do not pass a Windows path such as `C:\...` directly to a Linux command; use its Linux access path.

Permission or line-ending error. Running `bash dosya.sh` does not require execute permission; `./dosya.sh` does. A `bad interpreter` message or a visible `\r` error may result from Windows CRLF line endings. Select LF in the editor and resave only the affected text file; do not blindly convert all data files.

Output already exists. Refusing to overwrite a file is intentional behaviour in the teaching script. Choose a new output name and record which run used which parameters. A disk or write failure may leave a partial file; do not omit content checks beyond file existence or a zero exit code.

Process terminated or very slow. Examine Linux memory, the WSL cap, Windows load and free disk space together. A `Killed` message alone does not identify the cause; logs are needed to confirm suspected memory exhaustion. First try smaller inputs and fewer concurrent jobs instead of masking the problem by increasing swap alone.

Practical note

The synthetic Bash/Python exercise and its negative controls were executed locally on macOS arm64; actual versions are recorded in the companion README. Installation, WSL Settings, .wslconfig and dependency-solving steps for Linux/WSL2 are instructions based on official documentation, not Windows/WSL2 tests executed on this computer. Success on the small example does not establish large-data performance or platform compatibility for every bioinformatics tool.

References

[1] Free Software Foundation. *Bash Reference Manual*. https://www.gnu.org/software/bash/manual/bash.html

[2] Microsoft. *What is PowerShell?* https://learn.microsoft.com/en-us/powershell/scripting/overview

[3] Microsoft. *What is Windows Terminal?* https://learn.microsoft.com/en-us/windows/terminal/

[4] Microsoft. *Comparing WSL Versions*. https://learn.microsoft.com/en-us/windows/wsl/compare-versions

[5] Canonical. *Install Ubuntu Desktop*. Ubuntu Desktop documentation. https://ubuntu.com/desktop/docs/en/latest/tutorial/install-ubuntu-desktop/

[6] Microsoft. *Install WSL*. https://learn.microsoft.com/en-us/windows/wsl/install

[7] Canonical. *Install Ubuntu on WSL 2*. https://ubuntu.com/wsl/docs/latest/howto/install-ubuntu-wsl2/

[8] Microsoft. *Set up a WSL development environment*. https://learn.microsoft.com/en-us/windows/wsl/setup/environment

[9] Microsoft. *Basic commands for WSL*. https://learn.microsoft.com/en-us/windows/wsl/basic-commands

[10] Microsoft. *Advanced settings configuration in WSL*. https://learn.microsoft.com/en-us/windows/wsl/wsl-config

[11] Microsoft. *Working across file systems*. https://learn.microsoft.com/en-us/windows/wsl/filesystems

[12] Free Software Foundation. *GNU Coreutils manual*. https://www.gnu.org/software/coreutils/manual/coreutils.html

[13] Free Software Foundation. *GNU Grep manual*. https://www.gnu.org/software/grep/manual/grep.html

[14] Python Software Foundation. *csv—CSV File Reading and Writing*. https://docs.python.org/3/library/csv.html

[15] Microsoft. *File Permissions for WSL*. https://learn.microsoft.com/en-us/windows/wsl/file-permissions

[16] Free Software Foundation. *Bash Reference Manual: Redirections*. https://www.gnu.org/software/bash/manual/html_node/Redirections.html

[17] Free Software Foundation. *Bash Reference Manual: Pipelines*. https://www.gnu.org/software/bash/manual/html_node/Pipelines.html

[18] Free Software Foundation. *Bash Reference Manual: Quoting*. https://www.gnu.org/software/bash/manual/html_node/Quoting.html

[19] Free Software Foundation. *Bash Reference Manual: The Set Builtin*. https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html

[20] Anaconda. *Miniconda: Linux installer*. https://www.anaconda.com/docs/getting-started/miniconda/install/linux-install; installer archive: https://repo.anaconda.com/miniconda/

[21] conda-forge contributors. *Miniforge*. https://github.com/conda-forge/miniforge

[22] Conda developers. *conda-libmamba-solver: User Guide*. https://conda.github.io/conda-libmamba-solver/user-guide/

[23] Mamba developers. *Mamba User Guide*. https://mamba.readthedocs.io/en/latest/user_guide/mamba.html

[24] Mamba developers. *Mamba Installation*. https://mamba.readthedocs.io/en/latest/installation/mamba-installation.html

[25] conda-forge contributors. *Transitioning from Anaconda's defaults channels*. https://conda-forge.org/docs/user/transitioning_from_defaults/

[26] Bioconda contributors. *Bioconda documentation: Usage*. https://bioconda.github.io/

[27] Conda developers. *Managing environments*. https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html

[28] Python Software Foundation. *The Python Tutorial*. https://docs.python.org/3/tutorial/

[29] Python Software Foundation. *The Python Tutorial: More Control Flow Tools*. https://docs.python.org/3/tutorial/controlflow.html

[30] Python Software Foundation. *The Python Tutorial: Input and Output*. https://docs.python.org/3/tutorial/inputoutput.html

[31] procps-ng contributors. *procps-ng: free, top and vmstat manuals*. https://gitlab.com/procps-ng/procps/-/tree/master/man

[32] Anshori M, Maulidiansyah, Sukron M. LLM inference performance analysis across Windows, Windows Subsystem for Linux 2 and native Linux. *JITK (Jurnal Ilmu Pengetahuan dan Teknologi Komputer)*. 2026;12(1):451–462. https://doi.org/10.33480/jitk.v12i1.8435

[33] Kinghorn D. *WSL2 vs Linux (HPL HPCG NAMD)*. Puget Systems; 2022. Technical report. https://www.pugetsystems.com/labs/hpc/wsl2-vs-linux-hpl-hpcg-namd-2354/

[34] Sobieraj M, Kotyński D. Docker performance evaluation across operating systems. *Applied Sciences*. 2024;14(15):6672. https://doi.org/10.3390/app14156672

[35] Conda developers. *conda run*. https://docs.conda.io/projects/conda/en/stable/commands/run.html