Handout: Introducing the command line

Table of Contents

Welcome to GenEpi-BioTrain Training in Genomic Epidemiology and Public Health Bioinformatics "Bridging the Gap" - Third edition (TB and AMR)! This course will equip you with the basic skills needed to navigate Unix environments, automate tasks, and harness its capabilities to advance your understanding and practice in bioinformatics. This initial segment will be introducing Unix, focusing specifically on understanding the command-line.

Unix

Unix is a powerful operating system renowned for its stability, security, and versatility. In bioinformatics, Unix provides essential tools and a powerful command-line interface for analyzing biological data, managing computational resources, and running sophisticated algorithms efficiently. Originally developed in the 1970s, **Unix has become the foundation for many modern operating systems, including Linux and macOS**.

Linux

Linux is an open-source operating system based on Unix. **Unlike Unix, which is often commercial and closed-source, Linux is freely available and can be modified by anyone**. It supports a variety of distributions like Ubuntu and CentOS, making it ideal for diverse computing needs. Widely used in bioinformatics, Linux provides essential tools for managing large datasets and automating analyses.

Command line

The command-line, also known as the terminal or shell, is a simple, text-based interface used to interact with a computer's operating system. **Unlike graphical user interfaces (GUIs) that rely on clickable visual elements, like icons, buttons, and menus, the command-line requires users to type commands to perform tasks.** This interface provides powerful and flexible control over the system, allowing users to navigate the file system, manipulate files and folders, install and manage software, and execute complex scripts and programs. Key advantages of using the command-line include its speed, automation capabilities through scripting, and greater control over system operations. **It is a fundamental tool for bioinformaticians**. You type commands, press Enter, and the computer will do what you ask. A typical command follows this structure:

```
Command [options] [arguments]
```

- **Command**: The action you want to perform
- **Options**: Optional flags or settings that modify the behavior of the command. Options often start with a single - followed by a letter or two -- followed by a word.
- **Arguments**: The targets on which the command acts, such as files, folders, etc.

Overview of Important Unix Commands

These commands will be further elaborated throughout this handout.

Navigating the file system

pwd	**p**rint **w**orking **d**irectory	`pwd`
cd	**c**hance **d**irectory *move closer to root*	`cd [directory_path]` *`cd ..`*
ls	**ls**t content	`ls`

Managing the File System

touch	create new file	`touch [file_name]`
mkdir	**m**a**k**e new **dir**ectory	`mkdir [directory_name]`
rm	**rm**ove file	`rm [file_name]`
rm -r	**rm**ove (**r**ecursive) directory	`rm -r [directory name]`
cp	**cp**y file	`cp [file_name] [destination]`
cp -r	**cp**y (**r**ecursive) directory	`cp -r [directory_name] [destination]`
ln -sr	**ln**k file (**s**ymbolic, relative)	`ln -s [path_to_original] [path_to_link]`
mv	**mv**e file/directory	`mv [source] [destination]`
mv	rename	`mv [old_name] [new_name]`

Viewing & Editing files

cat	display content in terminal	`cat [file_name]`
cat	con**cat**enate files	`cat [file1] [file2] > [new_file]`
less	display content on page	`less [file_name]` (press q for quit)
head	display **first** few lines	`head [file_name]`
tail	display **last** few lines	`tail [file_name]`
nano	text editor	`nano [file_name]`

Handling & Compressing Files

>	**redirect** output of one command to a file	`[command1] > [file_name]`
>>	**append** output of one command to a file	`[command1] >> [file_name]`
tar -x	decompress directory	`Tar -xf [file_name]`
tar -zcvf	compress directory	`tar -zcf [file_directory]`

Advanced

grep	search for text within file	`grep 'pattern' [file_name]`
wc	**w**ord **c**ount	`wc`
| *(pipe)*	redirect output of one command to another	`[command1]	[command2]`
chmod	**ch**ange **mod**e (permission)	`chmod [mode] [file_name]`

Chapter 1: Navigating The File System

Navigating the file system in Unix involves understanding and using a set of commands that allow you to move through folders (**henceforth referred to as directories**), manage files, and explore the structure of your system. Unlike graphical user interfaces (GUIs), which rely on visual representations and mouse clicks, Unix uses a command-line interface (CLI) where you type commands to interact with the file system.

The Unix file system is organized in a hierarchical, **tree-like structure** that starts at the root directory. Understanding how to move through and interact with this structure will enable you to manage files, install software, and configure the system. The root directory is the topmost level of the file system hierarchy. It is represented by a single forward slash (/). Think of it as the trunk of a tree, from which all other branches (directories and subdirectories) grow. All files and directories on your Linux system originate from this root.

Let's say you have a Unix-based system, and your username is "alice". Here's an example of the hierarchical structure of your file system:

```
/
├── home                          # /home: User home directories.
│   └── alice                     # /home/alice (Home directory for user "alice"; shortcut ~).
│       ├── Documents             # ~/Documents: User's documents.
│       │   ├── resume.docx
│       │   ├── project_notes.txt
│       │   └── work
│       │       ├── report1.pdf
│       │       └── presentation.pptx
│       ├── Downloads              # ~/Downloads: User's downloads.
│       └── ...
├── etc
└── ...
```

The tilde (~) is a shortcut representing the home directory of the current user. For example, if your username is "alice", ~ would represent /home/alice. This shortcut simplifies navigation and file management within the terminal.

Print working directory

The pwd (print working directory) command in Unix is a simple yet essential tool used to display the current directory you are working in. When you enter `pwd` in the command line, it **outputs the full path of the directory you are currently in**, helping you keep track of your location within the file system. This is particularly useful when navigating complex directory structures, ensuring you always know where you are before performing file and directory operations.

Change directory

The cd (change directory) command is used to **navigate between directories** in the command-line interface. It allows you to move to different directories within the file system. It is similar to navigating folders in a graphical user interface (GUI). **Instead of clicking on folders, you type commands to move between directories**. Let's delve into how it works, focusing on the concepts of absolute and relative paths, which are essential for effective navigation:

- **Absolute Path**: An absolute path specifies the complete directory path from the root directory, similar to a full URL in a web browser. For example: the command `cd /home/user/documents/` navigate directly to the "documents" directory regardless of the current location.
- **Relative Path**: A relative path specifies the path relative to the current directory. For example: if you are currently in the "user" folder and you type `cd documents/`, you will move into the "documents" folder inside "user". It is like double-clicking on the "documents" folder when you are already in the "user" folder.

You can also **navigate up to the parent directory** using "..", similar to clicking the "back" button in a GUI. For example, if you are in /home/user/documents and you type `cd ..` you will move up to the "user" folder (/home/user). Typing `cd ../downloads/` moves you up one level to the "user" folder and down one level into the "downloads" folder inside the "user" folder (/home/user/downloads). Typing `cd ../..` moves you up two levels (/home).

****The tab key is your friend**** when navigating the command-line. Pressing tab while typing a path will auto-complete directory and file names if they are unique, saving time and reducing errors. For example, if you type `cd doc` and **press tab**, it will complete to cd documents if "documents" is the only directory starting with "doc". **If tab completion works, it's a good indication that you are in the right directory** or typing the correct path, providing a helpful sanity check. If tab doesn't work right away, try pressing it again, as you may have multiple completions to choose from.

List Content

Listing the contents of directories helps you understand and manage the files and directories within your system. The primary command used for this purpose is `ls`.

- **Basic Usage**: Typing `ls` displays the names of files and directories in the current directory. For examples: if your current working directory is /home/user/documents, typing `ls` will display the content of "documents". Typing `ls ../downloads/` will

display the contents of home/user/downloads even if it is not your current working directory. Here are some useful options for the `ls` command:

- **Detailed Information:** Use `ls -l` to get a long format listing, showing detailed information such as owner, size, and modification date.
- **Sorting by modification time**: `ls -lt` combines the long list and sorts files and directories by modification time.
- **Human readable sizes**: `ls -lh` combines the long format with human-readable file sizes, converting bytes sizes into KB, GB, etc.

Get path

Readlink displays the absolute path of a file or directory, showing you the full path from the root to the target. This is useful for confirming the exact location of a file in the filesystem. For example: Typing `readlink -f filename.txt` will give the full path to filename.txt.

Chapter 2: Managing the File System

Managing the file system is crucial for maintaining an efficient and organized workspace in Unix. A few basic operations form the foundation for organizing your workspace, backing up important data, and keeping a clean and navigable directory structure. This chapter will guide you through the essential commands and techniques for handling files and directories, ensuring you can effectively manage your environment.

Create

- **Creating Files**: The `touch` command is commonly used to create empty files. For example: if your current working directory is /home/user/documents, typing `touch file.txt` will create an empty file called file.txt (/home/user/documents/file.txt). Typing `touch ../newfile.txt` will create an empty file inside the directory "user" (/home/user/newfile.txt).
- **Creating Directories**: The `mkdir` (make directory) command is used to create new directories. For example: if your current working directory is /home/user/documents, typing `mkdir assemblies` will create a directory called "assemblies" inside "documents" (/home/user/documents/assemblies).

Remove

It is important to handle these operations with care because, **unlike graphical user interfaces, there is no "undo" button in Unix**. Once a file or directory is deleted, it is typically gone for good. Here is how to safely and effectively remove files and directories:

- **Removing Files**: The `rm` (remove) command is used to delete files. For example: To delete the file called newfile.txt inside /home/user you can simply type `rm newfile.txt` if "user" is your current working directory.
- **Removing Directories**: The `rm -rf` (remove -recursively and forcefully) command is used to delete directories and all of their contents recursively. This command is powerful and should be used with caution, as it **permanently removes the directory and everything inside it**. For example: typing `rm -rf /home/user/` will delete "user" and all files and directories inside it, i.e. "documents", "downloads", and their subfolders.

Copy

- **Copying Files**: The `cp` (copy) command is used to copy files. For example: typing `cp newfile.txt /home/user/documents/` will copy newfile.txt from your current directory to /home/user/documents. If you are in /home/user and type `cp /home/user/documents/newfile.txt /home/user/`, it will copy newfile.txt into /home/user.
- **Copying Directories**: Use the `-r` (recursive) option with `cp` to copy directories and their contents, just as you would with individual files. For example: typing `cp -r /home/user/documents/ /home/user/backup/` copies the entire "documents" directory to the "backup" directory.

Symbolic links

Symbolic links, also known as symlinks or soft links, are special types of files in Unix that act as pointers or **shortcuts to other files or directories**. They take up no space compared to copying files, since they point to the path of the target file or directory, rather than duplicating the file's content. Symbolic links can be created using either absolute or relative paths, depending on how you want the link to reference its target:

- **Absolute path:** An absolute symbolic link uses the full path from the root of the filesystem to point to a target. You create it with the `ln -s` command. For example: `ln -s /home/user/documents/newfile.txt symlink.txt` creates a symlink named "symlink.txt" in your current directory that points to "newfile.txt". This

type of link is reliable when the target's location is fixed, but it will break if the target is moved or deleted.

- **Relative path:** A relative symbolic link points to a target using a path relative to the location of the link itself. You create it with the `ln -sr` command. For example, `ln -sr ../documents/newfile.txt symlink.txt` creates a symlink named "symlink.txt" in your current directory that points to "newfile.txt" located one directory above. Relative symlinks are useful when the symlink and target may move together within the directory structure, but they can break if the relative path between them changes.

It's **crucial to include the -s option** when creating a symbolic link; otherwise, you'll create a hard link instead, which behaves differently and can have unintended consequences.

Move

- **Moving**: The `mv` (move) command is used to **move both files and directories** from one location to another, similar to how you would copy them with the cp command. For example: typing `mv newfile.txt /home/user/documents/` will move newfile.txt from your current directory to /home/user/documents. Likewise, typing `mv /home/user/downloads/ /home/user/documents/` will move the entire "downloads" directory and its contents to the "documents" directory.

  The `mv` (move) command does not require the `-r` (recursive) option to move directories and their contents. Unlike `cp`, `mv` inherently moves entire directories and their contents.

Rename: The `mv` command serves the dual purpose of moving and renaming files and directories. Moving changes the directory path, while renaming changes the file or directory name within the same path. Conceptually, both actions are forms of relocation.

- Basic usage: To change the name of oldname.txt to newname.txt use the command `mv oldname.txt newname.txt`. By taking advantage of the dual purpose of the `mv` command, you can move and rename a file with a single, simple command: `mv oldname.txt /home/user/newname.txt`.

Special Characters

Wildcard: Wildcards allow you to perform operations on multiple files and directories with similar names or patterns, making tasks like listing, copying, moving, and deleting files more efficient.

- **Basic Usage**: The `*` wildcard matches any number of characters in a file or directory name. For example: typing `ls *.txt` lists all files with a .txt extension in the current directory. Typing `rm E_coli*` removes all files starting with E_coli.

The table below highlights the affected columns (green) for each example row.

***.fasta**	E_coli.fasta	E_coli.fasta.gz	coli.fasta	E_coli123.fasta	E_coli.txt	fasta.gz
E_coli*	E_coli.fasta	E_coli.fasta.gz	coli.fasta	E_coli123.fasta	E_coli.txt	fasta.gz
E_coli*.fasta	E_coli.fasta	E_coli.fasta.gz	coli.fasta	E_coli123.fasta	E_coli.txt	fasta.gz

Current Directory Reference: The dot (.) **represents the current directory** and can be used to simplify commands. For example, when you want to move or copy files into the current directory, using "." makes the command concise and clear.

The following commands will all move newfile.txt from /home/user/documents to /home/user/downloads:

- If you are located in /home/user/**documents**:
  - `mv newfile.txt /home/user/downloads/`
  - `mv /home/user/documents/newfile.txt ../downloads/`
  - `mv newfile.txt ../downloads/`
- if you are located in /home/user/**downloads**:
  - `mv ../newfile.txt /home/user/downloads/`
  - `mv /home/user/documents/newfile.txt .`
  - `mv ../newfile.txt .`
- Regardless of current location:
  - `mv /home/user/documents/newfile.txt /home/user/downloads/`

Chapter 3: Viewing and Editing Files

Effectively viewing and managing the contents of files is a fundamental skill for any user. Unix provides a variety of powerful commands to you help you inspect and manipulate file contents directly from the terminal:

View

- **Display and Merge Files**: The `cat` command is used to concatenate and display the full contents of files directly in the terminal. For example: running `cat newfile.txt` displays the entire content of newfile.txt in the terminal. One of its powerful features

is the ability to combine the contents of multiple files into a single output. This is particularly useful when you want to **merge several files together** into one. For example: `cat newfile1.txt newfile2.txt newfile3.txt > combined.txt` will redirect the output of "newfile[1-3].txt" into "combined.txt".

- **Display File in Separate Screen**: The `less` command is used to view the contents of a file one screen at a time. It is especially useful for reading large files, as it allows you to scroll though the file, both forwards and backwards, without loading the entire file into memory. For example: running `less newfile.txt` will open a separate screen with the file contents. You can find specific text within the file by typing `/` followed by a search term. To exit the screen, **enter "q" to quit**.
- **Display Beginning of Files**: The `head` command displays the first few lines of a file directly in the terminal, allowing you to quickly see the beginning of the file. For example: `head newfile.txt` will display the first ten lines of newfile.txt by default. The head command also has the option of specifying the number of lines displayed. For example: `head -n 20 newfile.txt` will display the first 20 lines of newfile.txt

- **Display End of Files**: The `tail` command displays the last few lines of a file directly in the terminal, allowing you to quickly see the end of the file. For example: `tail newfile.txt` will display the last ten lines of newfile.txt by default. Similar to head, the `tail` command also has the option of specifying the number of lines displayed. For example: `tail -n 5 newfile.txt` will display the last five lines of newfile.txt

Edit

Unix provides various text editors, both command-line-based and graphical, to facilitate editing directly within the terminal. **Nano** is one of the most user-friendly and accessible text editors available. To open a file using nano, simply type nano in front of the file you wish to edit; for example: `nano newfile.txt`.

In the very bottom of the nano text editor, you will see a section of key options, including saving changes (Ctrl + O), exiting (Ctrl + X), etc. To perform one of these options, press Ctrl and the letter associated with the option. You can save the script when exiting.

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Chapter 4: Handling and Compressing Files

Redirect

Redirection (`>`) is a method used to direct the output of a command to a file or another command, rather than displaying it on the screen. This allows you to capture output, store it, or pass it to another process for further use. For example: `ls > filelist.txt` will list all files and directories in the current working directory and save the output to filelist.txt.

Append

Appending (`>>`) refers to adding data to the end of an existing file without overwriting its current contents. This is useful when you want to accumulate data over time or when you need to preserve existing information while adding new output. For example: `ls /home/user/project1/reads >> filelist.txt` will list all files and directories from /home/user/project1/reads and append the output to filelist.txt.

Compress and Decompress

`tar` (short for tape archive) is a command for creating and extracting archive files, which bundle multiple files or directories into a single file.

- Use `-c` to create a new archive.
- Use `-x` to extract files from the archive.
- Use `–z` to compress the archive using gzip.
- Use `-v` to verbosely list the files being processed (shows progress in terminal).
- Use `-f` to specify the name of the archive file.
- Use `-h` to follow symbolic links and include the actual files they point to.

You can combine these options to perform tasks like this:

Decompress a tar.gz File: `tar -xzvf archive.tar.gz` will decompress archive.tar.gz while displaying the files being processed in the terminal.

Compress Directory: `tar -czvf archive.tar.gz home/user/document/reads` will create a compressed the directory "reads" into an archive called archive.tar.gz while displaying the files being processed in the terminal.

Compress Directory with Symbolic Links included: `tar -chzvf archive.tar.gz home/user/document/reads` will compress the directory "reads" into an archive called archive.tar.gz while displaying the files being processed in the terminal. If there are any symbolic links in "reads", tar will follow those links and include the actual files they point to in the archive, rather than just archiving the symbolic links themselves.

Chapter 5: Advanced

Search

`grep` is used for **searching text or files for lines that match a specified pattern**. It reads input from files and prints lines that match the search pattern. For example: `grep "search_term" filename.txt` will search for the term "search_term" in "filename.txt" and print all lines containing that term. Here are some useful options:

- **Case Insensitive Search**: Use `grep -i` to ignore case distinctions.
- **Search Recursively**: Use `grep -r` to search directories and their subdirectories.
- **Count Matches**: Use `grep -c` to count the number of lines that match the pattern.
- **Show Line Numbers**: Use `grep -n` to display line numbers with matching lines.
- **Contextual Search**: use `grep -A [number]` to display lines after each matching line.
- **Only Matching Part**: Use `grep -o` to display only the matching part of each line.

For example: `grep -in "Once upon a time" fairytales.txt` performs a case-insensitive search for the phrase "once upon a time" in the file "fairytales.txt", displaying both the matching lines and their line numbers.

For example: `grep "^Once upon a time" fairytales.txt` searches "fairytales.txt" for lines that begin with "Once upon a time" and displays those lines. The ^ character indicates the start of a line, so grep will only match lines that start with "Once upon a time".

Word count

The `wc` (word count) command is used to count the number of lines, words, and characters in a file or input. It's commonly used for quickly summarizing text data. Basic usage:

- **Count lines**: Use `wc –l` to count the number of lines in a file.
- **Count Words**: Use `wc -w` to count the number of words.
- **Count Characters**: Use `wc -c` to count the number of characters.

For example: `wc -w filename.txt` is used to count the number of words in filename.txt. If you don't specify an option, for example `wc filename.txt`, it will count the number of lines, words, and characters in filename.txt.

Pipe

The pipe (`|`) allows you to connect the output of one command directly into the input of another. For example: `ls | grep "txt"` will list all files in the current working directory and then filters the list to show only those containing "txt" using grep. Another example: `ls | wc -l` is used to count the number of files and directories in the current directory.