Linux Commands

Introduction

In Class 1, we covered the basics of Linux setup and navigation. Now, we'll focus on using **Linux commands** effectively, which is a crucial skill for handling bioinformatics data. You'll learn how to manipulate files, manage processes, and work efficiently in the command-line environment. These skills will form the foundation for more complex tasks like RNA-seq analysis in upcoming classes.

Key Linux Commands for Setup and Navigation

Before diving into advanced commands, let's take a look at the essential Linux commands:

- `pwd`: Prints the current working directory.
- `mkdir <directory>`: Creates a new directory.
- `rmdir <directory>`: remove a directory (that is empty).
- `ls`: Lists the contents of a directory.
- `cd <directory>`: Changes the working directory.

Best Practices

- **Always Press Enter After Typing Commands**: Ensure you execute commands by pressing Enter.
- **Leave a Space After Command Words**: Commands and their options/arguments should be separated by spaces.
- **Use Descriptive Directory and File Names**: Helps in easily identifying and navigating through directories.
- **Regularly Use `pwd` to Confirm Your Location**: Avoid confusion by checking your current directory.
- **Leverage `man` Pages for Learning**: When unsure about a command or its options, refer to the manual.

Linux commands are fundamental for moving around the filesystem, managing directories, and handling files.

1. `pwd`: Prints the current working directory.

Example:
```
pwd
```

Output:

```
/home/username/Bioinformatics_Project
```

2. **`mkdir <directory>`**: Creates a new directory.

Example:

```
mkdir Research_Project
```

This command creates a directory named `Research_Project` in the current directory.

```
mkdir -p /home/username/Bioinformatics_Project/data/raw
```

The `-p` option creates nested directories, resulting in the path `/home/username/Bioinformatics_Project/data/raw`.

3. **`rmdir <directory>`**: remove a directory (that is empty).

Example:

```
rmdir Research_Project
```

N.B: If the directory is not empty (contains other files or directories), the `rm -r` `command is used to delete the directory.`

4. **`ls`**: Lists the contents of a directory. Common options for `ls`:
   - **Options**: (also known as flags or switches are special arguments that modify the behavior of commands)
     - **`-a`**: Show all files, including hidden files (those starting with `.`).
     - **`-R`**: Recursively list subdirectories.
     - **`-r`**: Reverse the order of the output.
     - **`-t`**: Sort by the last modified time.
     - **`-S`**: Sort by file size.

- **`-l`**: Long listing format (includes permissions, owner, size, and modification date).
- **`-1`**: List one file per line.
- **`-m`**: Display output in comma-separated format.
- **`-Q`**: Quote filenames.

Examples:

```
ls -l        # Lists detailed file information

ls -aR       # Lists all files recursively, including hidden ones

ls -t        # Sorts files by modification time
```

Options provide additional instructions to a command. For example, some commands have basic functionality (like listing files or changing directories), but with options, they can perform more advanced tasks (like listing files in a specific format or changing directories while showing additional details). Options make commands more flexible and powerful.

Single Dash (`-`) Options: Short options are usually a single letter preceded by a dash, like `-l` or `-a` in the `ls` command. You can **combine multiple single-dash options** together, like `-la` instead of `-l -a` (for example, `ls -la`).

Double Dash (`--`) Options: Long options are typically words preceded by two dashes, like `--help` or `--version`. **They cannot be combined** in the same way as single-dash options; each double-dash option must be specified separately.

The distinction exists mainly for usability and convention. Older Unix tools originally used single-letter, single-dash options to keep commands short and quick to type. With the evolution of more user-friendly tools, long-form options with double dashes (`--`) were introduced to provide clearer, self-explanatory commands.

5. **`cd <directory>`**: Changes the working directory.

Example:

```
cd Research_Project
```

This command changes the working directory to `Research_Project`.

Summary of `cd` Options

Command	Description
`cd <directory>`	Navigate to the specified directory
`cd ..`	Move up one directory level
`cd ../../..`	Move up multiple levels
`cd -`	Return to the previous directory
`cd` or `cd ~`	Return to the home directory
`cd /`	Go to the root directory
`cd $HOME`	Navigate to the home directory using the environment variable

---

System Information Commands

These commands reveal details about the system, kernel, and mounted filesystems, providing insight into system configuration and setup.

6. **`uname`** - Show system and kernel information.
    - **Options**:
        - **`-a`**: Shows all information (kernel, system, version, and more).

        - **`-r`**: Displays only the kernel release.
        - **`-o`**: Shows the operating system.

Examples:

```
uname -a   # Complete system details

uname -r   # Kernel release only

uname -o   # Operating system name
```

7. **`mount`** - Show mounted filesystems.

Examples:

```
mount                # Displays all mounted filesystems
```

Date and Time Commands

Commands for checking the system’s current date and time are useful for tracking logs, scheduling tasks, and monitoring system activity.

8. **`date`** - Show the current system date and time.
    - **Options**:
        - **`+"%Y-%m-%d %H:%M:%S"`**: Custom formatting to display in a specific format.
        - **`+"%B %Y"`**: Show only the month and year.

Examples:

```
date                      # Display current date and time

date +"%Y-%m-%d %H:%M:%S" # Custom date-time format for logs

date +"%B %Y"             # Display month and year only
```

Uptime and User Information

These commands provide information on system uptime and the current user, useful for checking system status and permissions.

9. **`uptime`** - Show system uptime and load averages.
    - **Options**:
        - **`-p`**: Display uptime in a more readable format (e.g., "up 3 days, 4 hours").

Examples:

```
uptime           # Shows system uptime and load averages

uptime -p        # Shows uptime in a more readable format
```

10. **`whoami`** - Show the current logged-in user.

Example:

```
whoami
```

Manual Pages

The `man` command provides comprehensive documentation for Linux commands, making it essential for learning about command options and usage.

11. **`man <command>`** - Show the manual page for a command.

Examples:

```
man ls          # View the manual for the ls command

man uname       # View the manual for the uname command
```

Additional Useful Commands

12. **`history`**: Displays command history, allowing you to repeat or reference past commands.

Example:

```
history
```

13. **`which <command>`**: Locates the executable path of a command.

Example:

```
which python
```

Output:

```
/usr/bin/python
```

14. **`mnt`**: Lists mounted filesystems, drives and partition mounts, helpful for checking attached devices.

File and Directory Operations

Managing files and directories is a core aspect of working with bioinformatics datasets. You'll often be dealing with large text files, sequence data, and script files.

15. **`cp <source> <destination>`**: Copies a file from the source to the destination.
16. **`mv <source> <destination>`**: Moves a file or renames it.
17. **`rm <file>`**: Deletes a file.

Example:

```
cp sequences.txt backup_sequences.txt  # Copy a file
mv backup_sequences.txt archive/       # Move or rename a file
rm sequences.txt                       # Remove a file
```

Inspecting Files

Often, you'll need to view files to understand their contents, especially in bioinformatics where large text-based data files (e.g., FASTA, VCF) are common.

18. **`cat <file>`**: Outputs the entire file to the terminal.
19. **`head <file>`**: Displays the first 10 lines of the file.
20. **`tail <file>`**: Displays the last 10 lines of the file.
21. **`less <file>`**: Allows you to scroll through a file interactively.

Example:

```
head sequences.fasta  # View the first 10 lines of a FASTA file
tail -n 20 sequences.fasta  # View the last 20 lines of a FASTA file
less sequences.fasta  # Scroll through the FASTA file
```

Use Case in Bioinformatics: When analyzing sequence files, use `head` and `tail` to get a quick preview of the data, such as the start or end of a gene sequence.

Searching for Data Within Files

You'll frequently need to search for specific patterns, gene names, or IDs in large files. The **grep** command allows you to search within files.

22. **`grep <pattern> <file>`**: Searches for a pattern in a file.
- **`grep -i <pattern> <file>`**: Case-insensitive search.
- **`grep -r <pattern> <directory>`**: Recursive search through directories.

Example:

```
grep "ATCG" sequences.fasta  # Search for the sequence 'ATCG'
grep -i "geneX" genes.txt  # Case-insensitive search for 'geneX'
```

Use Case in Bioinformatics: You can use `grep` to find specific gene sequences, annotations, or other relevant patterns in large datasets.

File Types and Text Editors in Linux

- **Shell Script Files (`.sh`)**: `.sh` files contain sequences of commands for automation, written in languages like Bash or sh.
- **Comparing Text Editors**:
  - **Nano**: A simple, beginner-friendly editor with single-mode operation and a menu of commands at the bottom.
  - **Vim**: Advanced, with multi-mode operation (Insert, Normal, Command), suitable for users needing powerful text manipulation.

    Usage: To create or edit a file, type `vim filename.txt`.

  - **Insert Mode**: Press `i` to start writing.
  - **Command Mode**: Press `Esc` to exit Insert Mode.

        - **Save**: Type `:w` to write changes.
        - **Quit**: Type `:q` to exit without saving or `:wq` to save and quit.
        - **Force Quit**: Use `:q!` to quit without saving changes.
- **Modes in Vim**: Vim has two main modes:
    - **Insert Mode (`i`)**: Insert text before the cursor.
    - **Append Mode (`a`)**: Insert text after the cursor.
- **File Recovery in Vim**: If needed, `vim -r filename.txt` can be used to recover an unsaved file.

Setting Permissions, Monitoring, and Managing Processes

Setting up a username and password on Ubuntu

Setting up a username and password on Ubuntu can be done in several ways, depending on whether you're creating a new user or changing the password for an existing user. Here's how to do both:

☐ Create a New User with a Password

If you want to add a new user:

- Use the `adduser` command to create a new user and set their password:
  ```
  sudo adduser newusername
  ```
- Replace `newusername` with the desired username.
- You'll be prompted to create a password and enter user information. You can skip the additional information by pressing Enter.
- After the process completes, the user will be created with the specified username and password.

☐ Set or Change Password for an Existing User

If the user already exists, you can change their password:

- Use the `passwd` command followed by the username to set a password:
  `sudo passwd existingusername`
- Replace `existingusername` with the username for which you want to change the password.
- You'll be prompted to enter and confirm the new password.

☐ Allow the New User to Have `sudo` Privileges

To grant the new user administrative (sudo) rights:

- Add the user to the `sudo` group with the following command:
  `sudo usermod -aG sudo newusername`
- Replace `newusername` with the username you created.
- The user will now be able to use `sudo` for administrative tasks.

23. `sudo` command stands for **"superuser do"** and allows a permitted user to execute a command with **superuser (root) privileges**. Essentially, it provides temporary elevated permissions to perform tasks that require administrative access, without needing to log in as the root user.

☐ Verify the User Setup

You can log out and log back in with the new username and password to ensure it's set up correctly.

Handling Permissions

In bioinformatics, collaboration on shared files is common, so knowing how to handle permissions is important:

24. `chmod 775 <file>`: Changes the permission of a file to be read, written, and executed by the owner and group, and read-only by others.

The `chmod` command in Linux is used to change the permissions of files and directories, specifying who can read, write, and execute a file. Permissions are represented by numbers or letters, with **three main categories**: the **owner** (u), **group** (g), and **others** (o).

Breakdown of `chmod 775 <file>`:

- **775** is a numerical mode that represents the permission settings for owner, group, and others.
    - **7** (Owner): Read, write, and execute (`4 + 2 + 1 = 7`)
    - **7** (Group): Read, write, and execute (`4 + 2 + 1 = 7`)
    - **5** (Others): Read and execute only (`4 + 1 = 5`)

With `chmod 775`, the **owner and group** of the file can **read, write, and execute**, while **others** can only **read and execute**.

Syntax:

```
chmod [permissions] <file>
```

25. `chown <user>:<group> <file>`: Changes the owner and group of a file.

The `chown` command changes the **owner** and **group** of a file or directory, enabling control over who can access or modify a file based on their user or group identity.

Breakdown of `chown <user>:<group> <file>`:

- `<user>` is the username of the new owner.
- `<group>` is the group name for the file.
    - If you want to change only the owner or the group, you can leave one field empty (e.g., `chown <user>:` or `chown :<group>`).

Syntax:

```
chown [options] <user>:<group> <file>
```

Examples:

Changing the Owner and Group for a Single File:

```
chown alice:staff report.txt
```

- Assigns ownership of `report.txt` to user `alice` and the group `staff`.

Changing Only the Owner:

```
chown bob report.txt
```

- Changes the owner of `report.txt` to `bob` but leaves the group unchanged.

You'll also manage tasks with commands like `ps` to view running processes and `kill` to terminate processes when necessary.

When running analyses, you may have long-running tasks that consume a lot of computational resources. Linux provides several tools to help you monitor and manage these processes.

26. **`top`**: Displays a real-time list of system processes.
27. **`ps`**: Shows a snapshot of current processes.
28. **`kill <PID>`**: Terminates a process with the given Process ID (PID).

Example:

```
ps  # Show running processes
kill 12345  # Terminate the process with PID 12345
```

Use Case in Bioinformatics: When running resource-intensive bioinformatics workflows, `top` helps you monitor system load and `kill` allows you to terminate any tasks that are consuming too much memory or CPU.

Redirection and Piping

Redirection allows you to control where output goes, while piping (`|`) lets you chain commands together. These are essential for building powerful data processing pipelines.

29. **`>`**: Redirects output to a file.
30. **`>>`**: Appends output to a file.
31. **`|`**: Pipes the output of one command to another command.

Example:

```
grep "gene" sequences.fasta > gene_matches.txt  # Redirect output to a
file
ls | grep "txt"  # Pipe the output of 'ls' into 'grep' to find text
files
```

Example with Word Count (`wc`):

```
wc -w < Hello.txt
```

- Here, `<` feeds `Hello.txt` as input, displaying the word count without the filename.

Use Case in Bioinformatics: Redirection is useful for saving search results to a file, while piping helps streamline your analysis workflow by passing output from one tool directly to another.

Bash Shortcuts & Variable commands

Command Control and Navigation Shortcuts

1. **CTRL-c - Stop Current Command**

Example: If you start a process, like a ping, by typing:

```
ping google.com
```

- You can press **CTRL-c** to immediately stop the command from running.

The `ping` command in Linux (and other operating systems) is used to test the connectivity between your device and a specified network host.

2. **CTRL-z - Sleep Program (Suspend)**

Example: If you start a text editor, like:

```
nano myfile.txt
```

- Pressing **CTRL-z** will suspend (pause) the program and return you to the command line. You can later resume it by typing `fg` to bring it back to the foreground.

3. **CTRL-a - Go to Start of Line**

Example: In the middle of typing:

```
echo "Hello, world!"
```

- If you want to go to the start of the line without using arrow keys, press **CTRL-a**.

4. **CTRL-e - Go to End of Line**

Example: If your cursor is at the beginning of a long command line:

```
echo "This is a long sentence."
```

- Press **CTRL-e** to jump directly to the end of the line.

Text Manipulation Shortcuts

5. **CTRL-u - Cut from Start of Line**

Example: While typing:

```
echo "Hello, world!"
```

- Press **CTRL-u** to delete everything from the cursor position to the start of the line.

6. **CTRL-k - Cut to End of Line**

Example: In the command:

```
echo "Hello, world!"
```

- Place your cursor in the middle, then press **CTRL-k** to delete everything from the cursor position to the end of the line.

Command History and Reuse

7. **CTRL-r - Search History**

Example: Press **CTRL-r** and start typing part of a previous command, like:

```
CTRL-r (then type "echo")
```

- This will search your history for the most recent command containing "echo."

8. **!! - Repeat Last Command**

Example: If the last command was:

```
ls -la
```

- Typing `!!` will execute `ls -la` again.

9. **!abc - Run Last Command Starting with "abc"**

Example: If you previously ran:

```
apt-get update
```

- Typing `!apt` will repeat the last command starting with "apt".

Key Environment Variable Commands and Examples

1. **export NAME=value - Sets an Environment Variable**

Example 1: Define a new variable:

```
export PROJECT="Bioinformatics Project"
echo $PROJECT
```

Output:

```
Bioinformatics Project
```

Example 2: Set a variable to specify a tool's directory path:

```
export TOOL_PATH="/usr/local/bioinformatics/tools"
echo $TOOL_PATH
```

Example 3: Use `export` to set an environment variable for a session:

```
export GENOME_VERSION="GRCh38"
echo $GENOME_VERSION
```

Output:

```
GRCh38
```

2. **echo $NAME - Displays the Value of a Variable**

Example 1: Display a user-defined variable:

```
export DATASET="RNA_Seq_Data"
echo $DATASET
```

Output:

```
RNA_Seq_Data
```

Example 2: Show the home directory variable:

```
echo $HOME
```

Output:

```
/home/username
```

Example 3: Confirm a tool’s location path:

```
echo $TOOL_PATH
```

Output:

```
/usr/local/bioinformatics/tools
```