Introduction to Linux Commands

This guide is designed to complement the one-hour interactive seminar on Linux commands. It provides clear explanations, examples, and practice ideas to help you build confidence with basic and essential Linux terminal operations. Topics include using the Windows Subsystem for Linux (WSL), working with files, navigating directories, managing processes, and compiling C programs using gcc.

1. What is Linux?

Linux is a family of open-source operating systems based on the Linux kernel. There are hundreds of Linux distributions (distros) like Ubuntu, Debian, Fedora, Kali and Mint. Each distro offers a unique combination of user interface and tools, but the underlying system principles remain the same. Linux is widely used in servers, embedded systems, and programming environments.

2. Accessing the Command Line

Overview: The Command Line Interface (CLI) allows you to communicate directly with your operating system through text commands. You can access it via a Terminal app (on Linux or macOS) or using **Windows Subsystem for Linux (WSL)** on Windows.

WSL allows Windows users to run a full Linux environment directly within Windows, without the need for dual boot or virtual machines. It is perfect for software development, networking, and system management tasks.

- **wsl**: Runs Linux commands directly from the Windows terminal. Example: wsl ls lists files in the Linux home directory.
- **wsl --install**: Installs the default Linux distribution on Windows 10/11. After installation, you can access Linux by typing 'wsl' in the Command Prompt.
- **wsl --list --online**: Displays all available Linux distributions you can install (e.g., Ubuntu, Debian, Fedora).
- **wsl --shutdown**: Gracefully stops all running WSL instances, freeing system resources.
- **Tip**: You can open a Linux terminal directly from the Windows Explorer by typing 'wsl' in the address bar.

3. System and Process Information

- **Overview**: Linux provides many tools to monitor and manage processes, users, and overall system state.
- **whoami**: Displays the username of the current user.
- **id**: Shows user and group identity information, useful for permission checks.
- **date**: Displays or sets the system date and time.
- **uptime**: Shows how long the system has been running, along with average system load.
- **hostname**: Displays or sets the name of the computer.
- **ps**: Lists currently running processes. Example: ps aux shows all active processes.
- **top**: Interactive command for real-time process monitoring. Press q to exit.
- **kill**: Terminates a process using its PID. Example: kill 1234 stops the process with ID 1234.
- **Exercise**: Run top, find your shell process using ps, and note its PID.

4. Navigation and File Management

- **Overview**: The Linux filesystem is hierarchical, starting from the root ('/'). Navigation commands let you move around, view, and manage files efficiently.
- **echo**: Prints text or variables to the terminal. Example: `echo Hello World!` or `echo $HOME`. Use `>` to save output: `echo data > file.txt`.
- **pwd**: Prints the current working directory — useful to know your location in the filesystem.
- **ls**: Lists files and directories. Use -l for detailed info, -a to include hidden files, and -lh for human-readable file sizes.
- **cd**: Changes the current directory. Example: cd .. moves up one directory, cd ~ moves to your home directory.
- **mkdir**: Creates a new directory. Example: mkdir projects creates a folder named 'projects'.
- **rmdir**: Removes an empty directory. To remove non-empty directories, use rm -r.
- **touch**: Creates a new empty file or updates the timestamp of an existing one.
- **cp**: Copies files or directories. Example: cp file.txt backup.txt creates a duplicate copy.
- **mv**: Moves or renames files. Example: mv old.txt new.txt renames a file.
- **rm**: Removes files. Use caution with rm -r to delete directories recursively.

Example:

```
mkdir lab1
cd lab1
touch notes.txt
echo "Hello Linux" > notes.txt
cat notes.txt
```

5. Viewing and Inspecting Files

- **Overview**: These commands allow you to quickly read, inspect, and summarize file contents without needing a text editor.
- **cat**: Displays file contents directly in the terminal. Useful for short files.
- **more / less**: Allows paged viewing of large files. less supports scrolling and searching.
- **head**: Displays the first few lines of a file (default: 10). Example: head -n 5 notes.txt
- **tail**: Shows the last few lines of a file. Example: tail -f logfile.txt lets you follow a log in realtime.
- **wc**: Counts lines, words, and bytes. Example: wc -l data.txt shows the line count.
- **grep**: Searches for text patterns inside files. You can combine it with options like **-i** for case-insensitive search or **-r** for recursive directory search.
- **Exercise**: Create a text file with several lines, then use head and tail to view different portions of it.

  Example:

  - `grep "main" hello.c`
  - `grep -i "error" logfile.txt`
  - `ps aux | grep bash`
  - `grep -r "TODO" ~/projects/`

6. Redirection and Pipes

- **Overview**: One of Linux's strengths is combining commands using redirection and pipelines to perform complex tasks efficiently. You can view file contents directly in the terminal or redirect outputs into other files.
- **>**: Redirects output to a file. **Example:** ls > files.txt saves directory contents into a file.
- **>>**: Appends to an existing file.
- **|**: Connects the output of one command as input to another. **Example:** ps aux | grep bash filters processes for those containing 'bash'.
- **<**: Redirects input from a file instead of the keyboard. **Example:** sort < names.txt
- **Exercise**: Use ls | wc -l to count the number of files in your current directory.

Example:

```
cat file.txt
head -n 5 file.txt
tail -n 10 file.txt
ls | grep ".txt"
echo "new line" >> file.txt
```

7. File Permissions and Help

Overview: Linux permissions ensure security and control over who can access or modify files. Every file in Linux has permissions controlling who can read, write, or execute it. Use `` `ls -l` to `` view details and `` `chmod` `` to change access. When in doubt, the system manual is your best friend.

- **chmod**: Changes file permissions. Example: chmod 755 script.sh makes it executable by all users.
- **chown**: Changes the owner and group of a file. Example: chown user:group file.txt
- **man**: Displays detailed manual pages for any command. Example: man ls
- **--help**: Provides a concise help summary for most commands. Example: grep --help
- **history**: Shows the list of previously executed commands. You can reuse them using !.
- **Exercise**: Change a file's permission to read-only and try editing it to observe the result.

Example:

```
ls -l
chmod 755 script.sh
chmod u+w file.txt
```

8. Writing and Running Shell Scripts

Overview: You can automate tasks by writing simple shell scripts. Scripts must start with **#!/bin/bash** and be made executable using `chmod`.

Example:

```
#!/bin/bash
echo "Hello $USER!"
pwd
date
Save the file as hello.sh, then run:
chmod +x hello.sh
./hello.sh
```

9. Networking and Downloading

- **Overview**: Linux includes powerful networking tools for retrieving and managing online resources.
- **curl**: Transfers data from or to a server using various protocols (HTTP, HTTPS, FTP, etc.). Example: curl -O https://example.com/file.txt downloads a file.
- **wget**: Alternative to curl for downloading files, supporting recursive downloading and resuming.
- **ping**: Checks network connectivity to a host. Example: ping google.com

These commands let you communicate with networks or download files.

```
ping google.com
curl -O https://example.com/file.txt
wget https://example.com/file.txt
```

- **SSH (Secure Shell)** allows you to securely connect to remote servers and execute commands on them. You can also transfer files using **`scp`** or set up key-based authentication for convenience.

```
ssh user@remote_host
scp localfile.txt user@remote_host:/home/user/
ssh-keygen
ssh-copy-id user@remote_host
```

10. Compiling and Running C Programs

- **Overview**: Linux is a popular platform for software development. The GNU Compiler Collection (gcc) is used to compile C and C++ programs. You can compile single or multiple files and run them from the terminal.
- **gcc file.c -o output**: Compiles a C source file into an executable named 'output'. Example: gcc hello.c -o hello
- **./output**: Runs the compiled C program.
- **gcc file1.c file2.c -o combined**: Compiles multiple source files into a single executable.
- **gcc -Wall file.c -o output**: Compiles with all warnings enabled, helping catch potential errors.
- **Exercise**: Write a simple C program printing 'Hello, World!', compile it with gcc, and execute it using ./a.out.

Example:

```
gcc hello.c -o hello
./hello
gcc file1.c file2.c -o combined
```

Conclusion

Linux mastery comes through exploration. Try chaining commands, managing permissions, and compiling your own programs.
Remember: every command has a manual page (use 'man command'), and learning by doing is the fastest path to confidence in the terminal.