# Operating System Fundamentals: Architecture & Practical Linux Usage

This session introduces the key concepts of operating systems (OS) and provides hands‑on practice with the Linux command line. The material is intended for adults with little or no technical background. The session is divided into two parts: the first half covers the architecture and core responsibilities of an operating system, while the second half teaches basic Linux commands through practical examples.

## 1. Operating System Basics

### 1.1 What Is an Operating System?

An **operating system** (OS) is the most important software on a computer. The OS manages the computer’s memory and processes, as well as all of its software and hardware. It allows you to communicate with a computer without knowing how to speak the computer’s language, and without it, a computer is useless.

The operating system provides an environment where user programs can run conveniently and efficiently. It runs at all times and manages the assignment of resources—such as memory, processor time and input/output devices—to different processes. It also prevents user programs from interfering with the proper operation of the system.

### 1.2 Goals and Functions of an Operating System

The primary goals of an OS are to make the computer easy to use and to allocate resources fairly among users and programs. An OS provides user convenience, facilitates program execution, manages CPU, memory and I/O devices, and protects data from unauthorized access. Secondary goals include efficient resource utilisation and system reliability.

* **User convenience:** Provides a friendly interface and simplifies interaction with the computer.
* **Program execution:** Creates an environment in which programs can run effectively.
* **Resource management:** Allocates CPU time, memory and devices to programs to ensure fair use.
* **Security and reliability:** Protects data and ensures the system operates smoothly.

### 1.3 Common Operating Systems

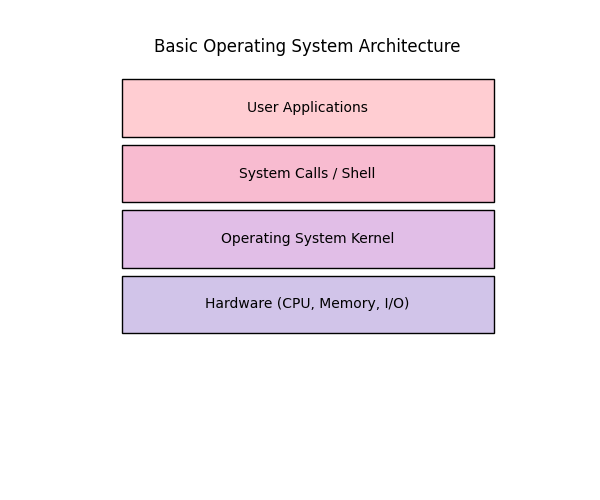
The three most common operating systems for personal computers are Microsoft Windows, macOS and Linux. Each provides a graphical user interface (GUI) that lets you click icons and menus rather than typing commands. Windows is known for its user‑friendly interface and broad hardware support; macOS offers a sleek design and tight integration with Apple hardware; and Linux is open source, highly customizable and popular for servers and development.

## 2. Operating System Architecture

An OS acts as an intermediary between user programs and hardware. Its internal design can follow various architectures. A **monolithic kernel** has a single large kernel responsible for functions like file management, memory management and device management. The kernel interfaces directly with application programs and hardware.

A **microkernel** divides the operating system into smaller, specialized kernels. Each microkernel handles a specific service, making the system more stable; if one kernel fails, others can continue to provide functionality. However, microkernels are more complex to design and may be less efficient.

**Layered** and **modular** architectures organise the OS into layers or modules. In a layered architecture, each upper layer relies only on the services of the layer below it, which improves verifiability but may reduce performance. Modular architectures load services as separate modules at boot time or on demand, making it easier to customize and extend the kernel.

 A simple layered view of an operating system showing user applications at the top, system calls or shell in the middle, the kernel below and hardware at the bottom.

## 3. Processes and Memory

### 3.1 Processes

A **process** is a program in execution. When you run a program, the code is loaded into memory and becomes an active entity called a process. A program is a passive set of instructions, whereas a process is the active execution of those instructions. Running the same program multiple times creates multiple processes.

The OS manages processes by allocating CPU time, memory and I/O resources, and by switching between processes to give the illusion of multitasking. Each process has a state (running, waiting, ready) and is described by a data structure called the Process Control Block (PCB), which stores information such as process ID, register values and memory pointers.

### 3.2 Process Memory Layout

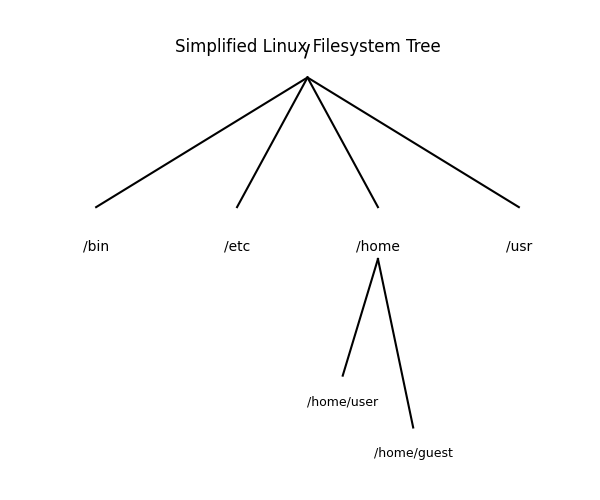
In memory, a process is divided into sections with specific purposes:

* **Text (code) segment:** Contains the program instructions.
* **Data segment:** Stores initialized global variables.
* **BSS segment:** Stores uninitialized global variables.
* **Heap:** Used for dynamic memory allocation during program execution.
* **Stack:** Holds temporary data such as function parameters, return addresses and local variables.

 A simplified view of a process in memory with separate segments for code, initialized data, uninitialized data, dynamic heap and the call stack.

### 3.3 Memory and File Management

The OS manages primary memory (RAM) by tracking which parts are in use and by whom, allocating memory to processes and reclaiming it when no longer needed. It also organizes data on disk into files and directories. The file system allows users to store and retrieve data in a hierarchical structure.

 A simplified Linux filesystem tree. The root directory / contains subdirectories like /bin (essential binaries), /etc (configuration files), /home (user home folders) and /usr (user programs). Within /home, individual users have their own directories.

## 4. Getting Started with Linux

Linux is a free and open‑source operating system known for its stability and flexibility. Many distributions exist, such as Ubuntu, Fedora and Debian. Linux features strong security, can run on older hardware and is widely used for servers and development.

Most Linux interactions occur through a command‑line interface (CLI). The CLI accepts text commands and provides precise control over the system. Commands are case‑sensitive and often follow the pattern command [options] arguments.

### 4.1 Basic Navigation and File Commands

The table below summarizes some essential Linux commands. These commands are safe to practice on a test system and help you navigate, view and manage files.

|  |  |
| --- | --- |
| Command | Purpose |
| ls | List directory contents |
| pwd | Print working directory |
| cd | Change directory |
| mkdir | Create directories |
| rmdir | Remove empty directories |
| rm | Remove files |
| cp | Copy files |
| mv | Move or rename files |
| touch | Create empty files |

### 4.2 Viewing and Editing Files

To view the contents of a file, use cat file.txt to print it to the terminal or less file.txt to scroll through it one page at a time. To create or edit text files, a beginner-friendly editor is nano:

$ nano notes.txt

This opens a simple text editor inside the terminal. After typing, press Ctrl+O to save and Ctrl+X to exit. You can also use graphical editors such as gedit or Visual Studio Code in desktop environments.

### 4.3 File Permissions and Ownership

Linux uses permissions to control who can read, write or execute files. Each file has permissions for the owner, group and others. For example:

$ ls -l myfile.txt

-rw-r--r-- 1 alice staff 1024 Aug 12 12:00 myfile.txt

The string -rw-r--r-- means the owner can read and write the file, the group can read it, and others can read it. Use chmod to change permissions. For instance, chmod u+x script.sh grants execute permission to the file’s owner.

### 4.4 Searching and Managing Processes

Use grep to search within files (e.g., grep "error" log.txt). The ps command shows running processes, and top provides a real‑time overview of CPU and memory usage. To terminate a process, use kill PID, where PID is the process ID.

### 4.5 Getting Help

Most commands include built‑in documentation. Use command --help or man command to view detailed usage instructions. For example:

$ ls --help

Displays options for the ls command. The man pages provide more complete documentation.

## 5. Hands‑On Practice

To build confidence using Linux, try the following tasks on a test machine:

* Open a terminal and find your current directory using pwd. Then list its contents with ls.
* Create a new directory called practice with mkdir practice and move into it using cd practice.
* Create an empty text file named notes.txt using touch notes.txt or by opening it in nano.
* Write a short message in notes.txt using nano, save and exit.
* Copy notes.txt to notes\_copy.txt using cp, then rename notes\_copy.txt to archive.txt using mv.
* Remove the archive.txt file with rm archive.txt and remove the empty practice directory with cd .. followed by rmdir practice.

## 6. Conclusion

Operating systems are the backbone of modern computing, managing hardware and software resources to provide a stable environment for applications. Understanding OS architecture helps demystify how computers work, while hands‑on practice with Linux commands builds confidence in using the command line. Keep exploring and practicing to become more comfortable with both concepts.