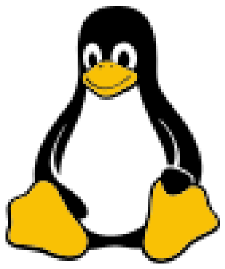
Date: 06.06.2025



Linux Fundamentals

Contents

1. Linux 2
2. Linux Distributions 3
3. Linux Components 4
i. Linux Kernel 4
ii. Components of Linux Kernel Architecture 4
a. Process Scheduler 4
b. Memory Manager 5
c. Virtual File System 5
d. Inter-Process Communicator 5
e. Network Interface 5
4. Linux File System Hierarchy 6
5. Linux Scripting 7
i. Core vs Shell 7
a. Shell: 7
b. Kernel: 7
ii. Different types of Shell 8
a. Bourne Shell (sh): 8
b. Bourne Again Shell (Bash): 8
c. C Shell (csh): 8
d. Z Shell (zsh): 8
e. Korn Shell (ksh): 8
iii. Shell Scripting 9
a. Shebang (or Hashbang) Line 9
b. Scripting with Bash 9
References 13

1. Linux

In 1969, Ken Thompson and Dennis Ritchie of Bell Laboratories developed the UNIX operating system. It was later rewritten in C to make it more portable and eventually became a widely used operating system.

A decade or so later, Richard Stallman started working on the GNU (GNU is Not UNIX) project, the GNU kernel called Hurd, which unfortunately never came to completion. The GNU General Public License (GPL), a free software license, was also created as a result of this.
The kernel is the most important piece in the operating system. It allows the hardware to talk to the software. It also does a whole lot of other things, but we'll dig into that in a different course. For now, just know that the kernel controls pretty much everything that happens on your system. During this time other efforts such as BSD, MINIX, etc. were developed to be UNIX like-systems. However, one thing that all these UNIX like-systems had in common was the lack of a unified kernel.

Then in 1991, Linus Torvalds started developing what we now know today as the Linux kernel.

2. Linux Distributions

As the popularity of Linux grew, various different organizations worked together on creating an Operating System using the Linux kernel catering to a specific use. They later came to be known as **Distributions** or **Distros**.

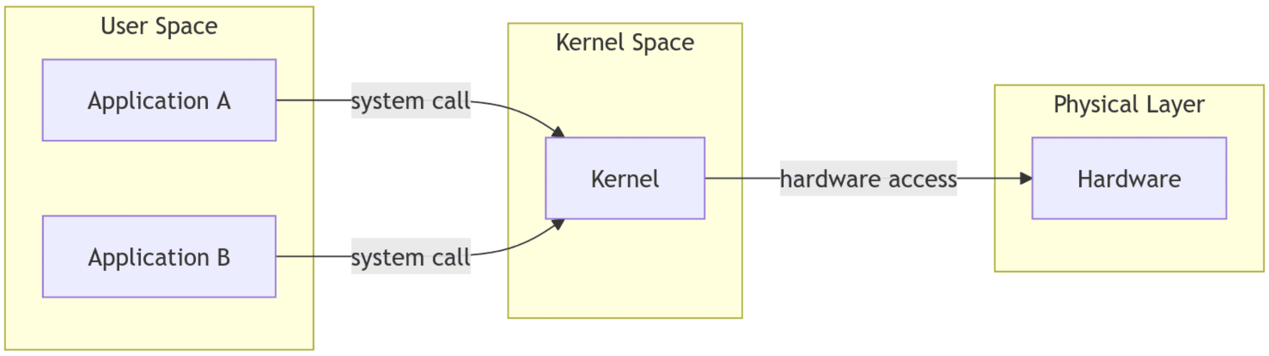
	Debian	**Ubuntu**	**OpenSUSE**	**Fedora**	**RHEL**	**Arch**	**Gentoo**
Uses	Any platform	Any platform, desktop, laptop and server	For desktop and laptop	RHEL without price tag for desktop and laptop	Enterprise, eg. a solid server OS	For desktop and laptop and IoT like Raspberry PI	For desktop and laptop
Based on	-	Debian	-	-	Fedora	-	-
Initial Release	1993	2004	1994	1995 as Redhat	1995 as Redhat	2002	2002
Community / Corporate	Community	Corporate (Canonical)	Community with Corporate Backing (SUSE)	Community with corporate backing (IBM)	Corporate (IBM)	Community	Community
Price	Free	Free	Free	Free	Paid, free subscription for developers	Free	Free
Default File System	ext4	ext4	btrfs	btrfs	xfs	-	-
Package Manager	apt	apt	zypper	dnf	dnf	pacman	portage
CPUs Supported	• amd64 • aarch64 • ppc64le • s390x • riscv64 (testing) • i686	• amd64 • Server img - aarch64 ppc64le s390x riscv64	• amd64 • aarch64 • Server img - ppc64le s390x • riscv64 (testing) • i686	• amd64 • aarch64 • Server img - s390x • riscv64 (testing)	• amd64 • aarch64 • ppc64le • s390x	• amd64	• amd64 • aarch64 • ppc64le • s390x • riscv64 • i686

3. Linux Components

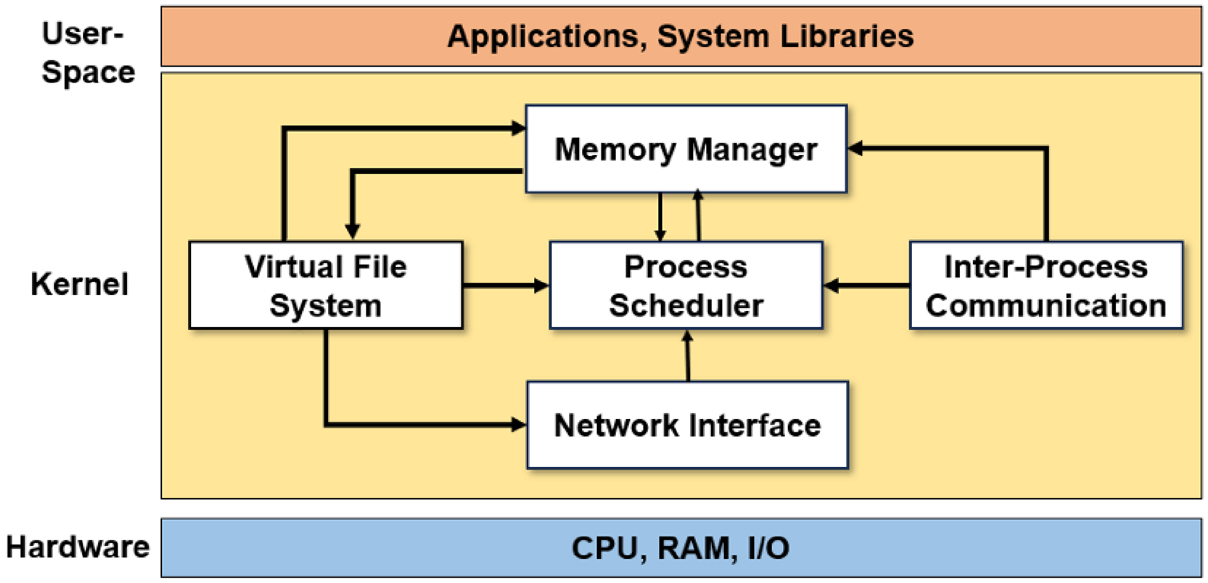
i. Linux Kernel

The Linux kernel is the core of any Linux operating system. It acts as the bridge between applications and the actual data processing done at the hardware level.

It achieves this by managing the system's hardware, providing essential system services, and facilitating communications between your computer's software and hardware.



ii. Components of Linux Kernel Architecture



a. Process Scheduler
It processes the control of the CPUs and also controls the other subsystems.
The process scheduler is divided into four main modules:

- System call interface.
- Scheduling Policy.

- Architecture Independent Scheduler.
- Architecture Specific Scheduler.

b. Memory Manager

It is responsible for the control process and it manages the hardware's resource memory. This memory manager redistributes the unused memory into persistent memory by swapping which increases the virtual memory.
It has three main modules:

- System call Interface.
- Architecture Independent scheduler.
- Architecture Specific Scheduler.

c. Virtual File System

It is the system that shows data stored on the hardware device. It allows the mounting of any logical system on a physical device. It also loads the executable programs.
The modules of the virtual file system:

- Device Driver
- Device Independent Interface
- Logical system
- System call interface
- System Independent Interface.

d. Inter-Process Communicator

It is the subsystem that is responsible for the communication and data exchange between the process scheduler and memory manager as well as other operating subsystems.

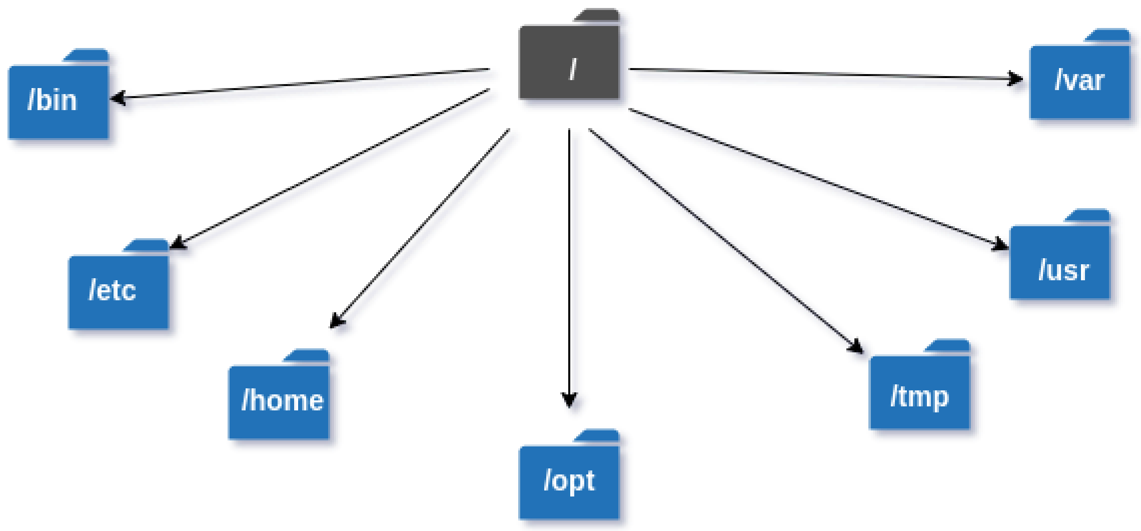
e. Network Interface

It allows the Linux system to connect with other devices over a network. Several hardware devices and networks are used by the network interface. Moreover, user processes and other kernel subsystems can access the network without knowing the configuration.
It includes the modules:

- Network device driver
- Device-independent Interface
- Network Protocol
- System Interface
- Protocol Independent Interface.

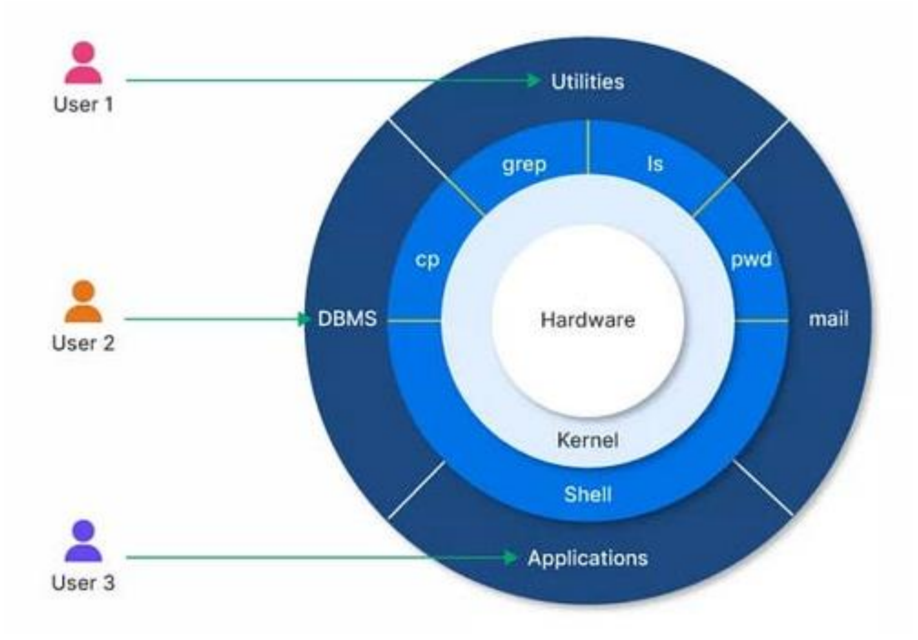
4. Linux File System Hierarchy

a. **Root Directory /**: The starting point of the file system. Everything on the system is inside this directory. It’s the "top-level" directory.
b. **/bin**: Stands for **binary**. Contains essential system programs or commands required for the system to function, like ls, cp, and cat.
c. **/boot**: Contains files that are required to boot the Linux system, including the kernel and bootloader files.
d. **/dev**: Stands for **devices**. Contains device files, which represent hardware devices. For example, /dev/sda might represent a hard drive.
e. **/etc**: Contains configuration files for the system. These are the settings for system-wide services like networking, user management, and system settings.
f. **/home**: Contains user directories. Each user’s personal files and settings are stored here. For example, /home/alex is where the user alex keeps their files.
g. **/lib**: Stands for **libraries**. Contains shared libraries that programs need to run. For example, .so files are dynamic libraries used by programs.
h. **/media**: Used for mounting external devices like USB drives, CDs, and DVDs.
i. **/mnt**: Traditionally used for mounting temporary file systems or remote file systems. It's a standard location for mounting filesystems in use for administrative tasks.
j. **/opt**: Contains optional software packages. It’s a directory for installing additional software that’s not part of the core system.
k. **/proc**: This is a **virtual filesystem** that provides information about the running processes, kernel, and other system information.
l. **/root**: The home directory of the root user (administrator). It’s like /home/username, but specifically for the superuser root.
m. **/sbin**: Contains system binaries that are used for administrative tasks. For example, fsck (used to check filesystems).
n. **/tmp**: Used for temporary files created by programs or the system. Files in this directory are often deleted when the system reboots.
o. **/usr**: Contains user programs and data that are not critical for the system’s operation. It’s where non-essential software packages are stored.



5. Linux Scripting

i. Core vs Shell



a. **Shell:**

The shell is an interface – either command-line (CLI) or graphical (GUI)
It allows users to communicate with the kernel.
The shell is responsible for interpreting user commands and relaying them to the kernel for execution.
Some common shells in Linux systems include Bash, sh, zsh, and csh.

b. **Kernel:**

The kernel is the core layer that handles low-level interactions with hardware.
Users don’t directly interact with the kernel; instead, they issue commands through the shell, which then passes those commands to the kernel.

ii. **Different types of Shell**

a. **Bourne Shell (sh):**

Borne Shell was developed by Stephen Bourne at Bell Labs.
One of the earliest and most widely used shells in UNIX systems.
Also known for its simplicity and essential features, primarily designed for script execution.
It does not have many modern conveniences like advanced command-line editing.

b. **Bourne Again Shell (Bash):**

BASH was developed as an enhanced, open-source replacement for Bourne Shell (sh).
It adds features like command-line editing, job control, and improved scripting capabilities.
It has become the default shell in most Linux distributions today, including Ubuntu.
Bash is backward-compatible with sh, so most scripts written for the original Bourne Shell can be run in Bash without modification.

c. **C Shell (csh):**

C Shell was developed by Bill Joy.
It uses C-like syntax, making it popular among developers familiar with the C programming language.
It introduces features like job control and aliases.

d. **Z Shell (zsh):**

An extended version of the Bourne Shell with features from Bash, csh, and ksh.
Provides advanced autocompletion, command correction, and prompt customization.
Popular among power users for its flexibility and extensive customization options.

e. **Korn Shell (ksh):**

Developed by David Korn.
A combination of features from both Bourne Shell and C Shell.
Used in both commercial and academic UNIX systems.

iii. Shell Scripting

a. Shebang (or Hashbang) Line

It is the character sequence "#!" at the beginning of a script file.
The shebang line tells the operating system which interpreter should be used to parse and execute the script
When a script with a shebang is run, the system uses the specified interpreter to process the script's contents.
This mechanism allows for seamless execution of scripts written in various languages without explicitly invoking the interpreter each time.
Shebang is ignored if shell is explicitly specified during running
Format:
#!/path/to/interpreter [optional arguments]

❖ **Best practices for Shebang**

Use absolute paths in shebang lines to ensure consistent behavior across different environments.
Before using a specific interpreter in a shebang line, ensure it's available on the target system.
Use the chmod command to make your scripts executable: chmod +x your_script.sh
Test your scripts on different systems to ensure the shebang line works consistently.
For improved portability, use the env command in your shebang lines when appropriate.

❖ **Common shebang lines**

Script Type	Shebang
sh	#!/bin/sh
bash	#!/bin/bash #!/usr/bin/env bash
zsh	#!/bin/zsh
python	#!/usr/bin/env python3
nodeJS	#!/usr/bin/env node
perl	#!/usr/bin/env perl
ruby	#!/usr/bin/env ruby

b. Scripting with Bash

❖ **Steps to create a simple "Hello, World" Script**

1. Create a new file called hello-world.sh: touch hello-world.sh

2. Make the script executable by running: chmod +x hello-world.sh

3. Add this code:

```
#!/usr/bin/env bash
echo Hello World
```

4. Execute the hello-world.sh script from the command line using:
   ./hello-world.sh

❖ **Using Variables**

```
#!/usr/bin/env bash
NAME="Aryan"
echo Hello, $NAME
```

Variable Rules:

i. No spaces around equal sign
ii. Variable names should be uppercase by convention
iii. Access variables using $ symbol

❖ **Input and Output**

```
#!/usr/bin/env bash

#input
read -p "Enter your name: " NAME
read -p "Hello $NAME, enter your age: " AGE

#output
echo $NAME is $AGE years old
```

❖ **Operators**

Arithmetic	Operation	Comparison	Operation
+	Addition	-eq	Equality
-	Subtraction	-ge	Greater than equal to
*	Multiplication	-gt	Greater than
/	Division	-le	Less than equal to
**	Exponentiation	-lt	Less than
%	Modulus	-ne	Not Equal to

Arithmetic expressions must be in (())

```
#!/usr/bin/env bash
var=$((2+7))
echo $var
```

❖ **Conditional Statements**

Use with Comparison and Logical Operators: -a (AND), -o (OR)

Type	Syntax
if – then - fi	#!/usr/bin/env bash if [#condition] then

	#statement fi

if –then – else – fi	#!/usr/bin/env bash
	if [#condition]
	then
	  #statement
	else
	  #do this by default
	fi

if – elif – else - fi	#!/usr/bin/env bash
	if [#condition]
	then
	  #statement
	elif [[#condition]] ; then
	  #statement
	else
	  #do this by default
	fi

❖ Looping Statements

Type	Syntax

For loop with Numbers	#!/usr/bin/env bash
	for i in {num1..num2}
	do
	  echo $i
	done

For loop with Strings	#!/usr/bin/env bash
	for x in str1 st2 strN
	do
	  echo $x
	done

While Loop	#!/usr/bin/env bash
	i=1
	while [$i -le N] ; do
	  echo "$i"
	  ((i += 1))
	done

❖ Reading Files

```
#!/usr/bin/env bash
LINE=1
while read -r CURRENT_LINE
    do
```

```
        echo "$LINE: $CURRENT_LINE"
    ((LINE++))
done < "fileName.txt"
```

- **Command Line Arguments for Script**

$@ represents the position of the parameters, starting from one.

```
#!/usr/bin/env bash
for x in $@
do
    echo "Entered arg is $x"
done
```

- **Executing Commands via Script**

```
#!/usr/bin/env bash
var=`df -h | grep tmpfs`
echo $var
```

```
#!/usr/bin/env bash
echo $(df -h | grep tmpfs)
```

Transforming Enterprises for Future

References

1. Linux Journey
2. Comparison of Linux Distributions
3. The Linux Kernel: Basics and Key Concepts - Linux Bash
4. The Linux Kernel | Baeldung on Linux
5. What is Linux Kernel? [A Complete Overview] - LinuxSimply
6. Understanding the Linux Kernel: The Core of an Operating System | Medium
7. Linux Directory Structure | GeeksforGeeks
8. Linux Scripting Series | Baeldung on Linux
9. Shebang Line: A Complete Explanation - DEV Community
10. Bash Book | Goalkicker
11. Linux Shell Scripting: A Step-by-Step Guide for Beginners
12. Shell Scripting for Beginners – How to Write Bash Scripts in Linux
13. Basic Shell Commands in Linux: Complete List | GeeksforGeeks