# PREPARATION Lab 1

# Linux fundamentals and Commands

## I.1. Introduction

Linux is one of the most popular operating systems. Its development was started in 1991 by Linus Torvalds. The operating system was inspired by Unix, another operating system developed in the 1969s by AT&T Laboratories. Unix was geared towards small computers.

At the time, "small" computers were considered machines that didn't need an entire hall with air conditioning and cost less than one million dollars. In addition, an affordable Unix system was not readily available on computers such as office computers, which tended to be based on the x86 platform. So, Linus, who was a student at the time, started to implement a Unix-like operating system that was supposed to run on this platform.

In general, Linux uses the same principles and basic ideas as Unix, but Linux itself doesn't contain Unix code, as it is an independent project. Linux is not backed by an individual company but by an international community of programmers. Since it is freely available, anyone can use Linux without any restrictions. One of the more appealing benefits of Linux is that it isn't a commercial operating system: its source code under the GNU General Public License (GPL)[1] is open and available to anyone to study; if you download the code (the official site is http://www.kernel.org) or check the sources on a Linux CD, you will be able to explore, from top to bottom, one of the most successful modern operating systems.

## I.2. What is Linux?

The Linux operating system is a set of programs that act as a link between the computer hardware and the user. The computer programs that allocate the system resources and coordinate all the details of the computer's internals are called the **operating system** or the **kernel**. Unlike other mainstream Operating Systems, Linux is made freely available and is Open Source.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

- Linux created in the early 1990s by Finnish software engineer Linus Torvalds.
- Several people can use a Linux computer at the same time; hence Linux is called a **multiuser** system.
- A user can also run multiple programs at the same time; hence Linux is a **multitasking** environment.

[1] The GNU project is coordinated by the Free Software Foundation, Inc. (http://www.gnu.org); its aim is to implement a whole operating system freely usable by everyone. The availability of a GNU C compiler has been essential for the success of the Linux project.

## I.3. Linux Architecture

Here is a basic block diagram of a Linux system

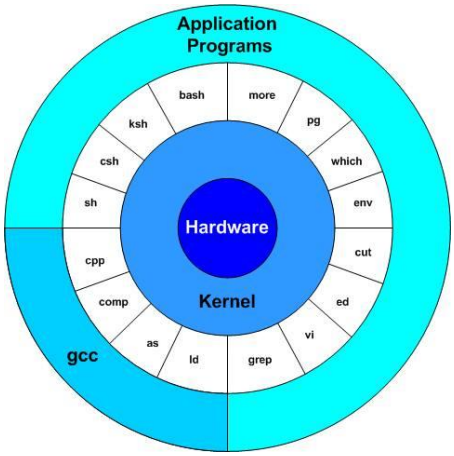



**Figure 1.1 : Linux Architecture**

Linux's architecture is based on a kernel that manages resources, including hardware. Numerous programs provide the interface between the user and the system. They can transmit commands to the kernel and receive responses in text mode.

In Linux, the graphics mode consists of layers superimposed on the base layers. This means that all operations can still be carried out in text mode.

The main concept that unites all the versions of Linux is the following four basics :

- **Kernel:** The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
- **Shell:** The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands.
- **Commands and Utilities:** There are various commands and utilities which you can make use of in your day-to-day activities. **cp, mv, cat** and **grep,** etc. are few examples of commands and utilities.
- **Files and Directories:** All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**

## I.4. Linux File System

The Linux File System is different to that of Windows, in that the Files System is not represented by Hard Drives and CD Drives like c:\ and d:\ etc. Instead, the whole system is represented by the **root directory**, or '**/**'. Files, folders, and drives, all exist within this.

All files on a Linux system are stored on file systems which are organized into a single inverted tree of directories, known as a file system hierarchy. In the inverted tree, the root lies at the top, and the branches of directories and sub-directories stretch below the root.

Within '**/**', exists a standard core directory structure:

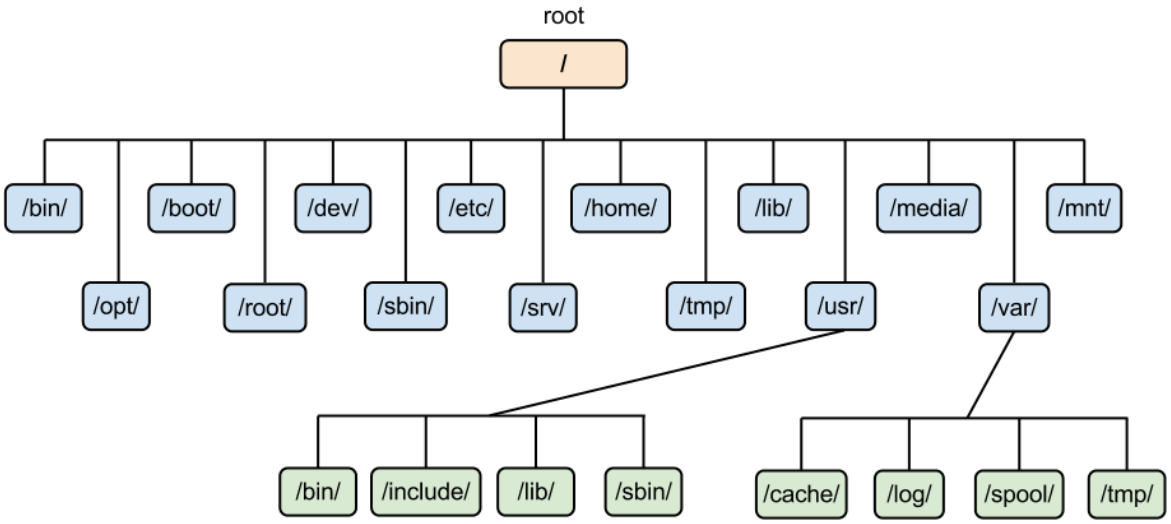



**Figure 1.2 : File System Hierarchy (FSH) of Linux**

Each user can create his or her own directory tree. A complete knowledge of the Linux tree structure is not necessary to use Linux. The description below may help you find the file you're looking for.

| Directory | Comments |
|---|---|
| **/** | The root directory represented by a forward slash. Where everything begins. |
| **/bin** | Contains binaries (programs) that must be present for the system to boot and run. |
| **/boot** | Contains the Linux kernel, initial RAM disk image (for drivers needed at boot time), and the boot loader. |
| **/dev** | Contains special device files which are used by the system to access hardware. |
| **/etc** | Contains all of the system-wide configuration files. |
| **/home** | In normal configurations, each user is given a directory in /home. Ordinary users can only write files in their home directories. This limitation protects the system from errant user activity. |
| **/lib** | Contains shared library files used by the core system programs. These are similar to dynamic link libraries (DLLs) in Windows. |
| **/media** | Used for mounting removable media, such as USB drives, CD-ROMs, and DVDs, etc. that are mounted automatically at insertion. |
| **/mnt** | Used for mounting other file systems, such as network file systems or other hard drives. |
| **/opt** | To store optional software, you may find empty in many systems |
| **/root** | Home directory for the administrative superuser, root. |
| **/sbin** | Contains system binaries that are required for system administration tasks, such as **fdisk** and **iptables**. |

| | |
|---|---|
| **/srv** | Contains data for services provided by the system, such as websites and FTP servers. |
| **/tmp** | Used for storing temporary files that are created and used by applications. |
| **/usr** | Contains user binaries and libraries for the system, such as system administration tools and development libraries. |
| **/var** | Contains variable data for the system, such as system logs and spool files, ... |

## I.5. Basic Commands and Utilities

### I.5.1. What is a Shell?

A shell is a type of computer program called a command-line interpreter that lets Linux and Unix users control their operating systems with command-line interfaces. Shells allow users to communicate efficiently and directly with their operating systems. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation. It is named a shell because it is the outermost layer around the operating system.

On most Linux systems a program called **bash** (which stands for Bourne Again SHell, an enhanced version of the original Unix shell program, **sh**, written by Steve Bourne) acts as the shell program. Besides **bash**, there are other shell programs available for Linux systems. These include: **ksh**, **tcsh** and **zsh**.

### I.5.2. What's a "Terminal?"

It's a program called a terminal emulator. A program which is responsible for providing an interface to a user so that he/she can access the shell. It basically allows users to enter commands and see the output of those commands in a text-based interface. Large scripts that are written to automate and perform complex tasks are executed in the terminal. Some Linux distributions install several. These might include gnome terminal, konsole, xterm, rxvt, kvt, nxterm, and eterm.

To access the terminal, simply search in search box "terminal" and double-click it.

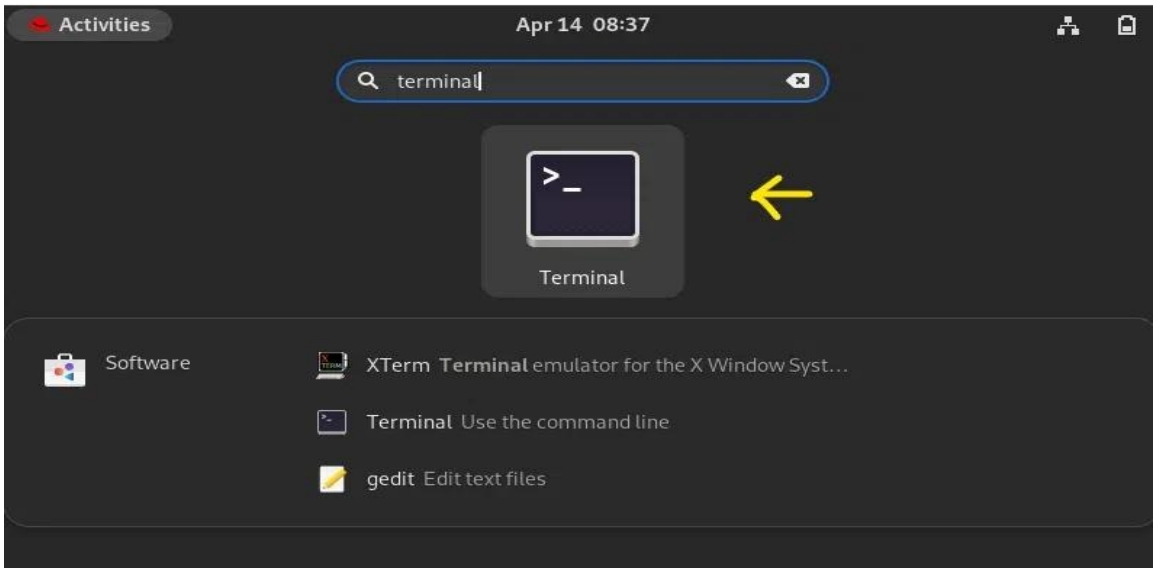



**Figure 1.3 : Open terminal**

Here you can see how the terminal looks of Linux.

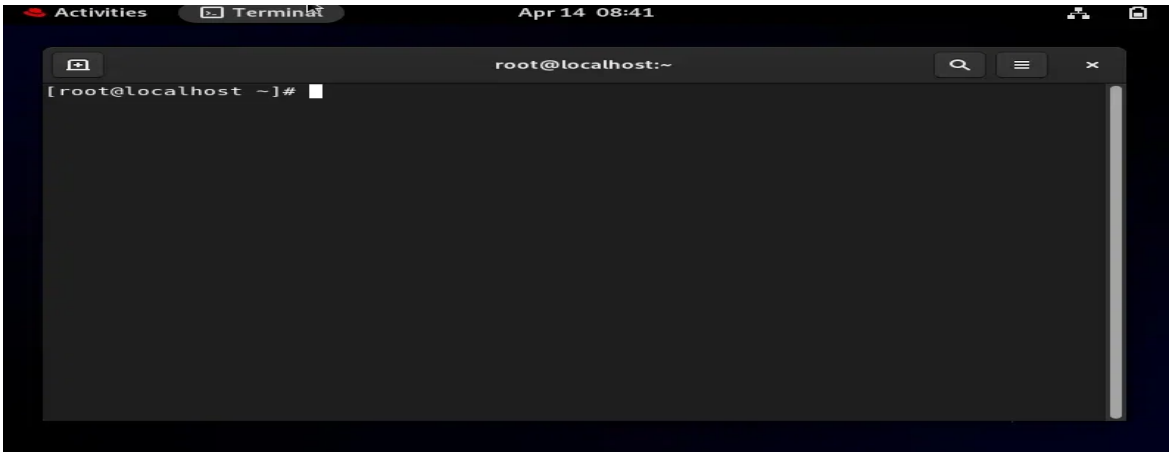



**Figure 1.4 : Terminal**

A shell prompt is the main way to interact with a command line only interface. A typical shell prompt will look something like this: **[username@domain directory]$**

The prompt can vary according to the shell configuration and can contain information about the user, host (system name), current directory, etc. It is defined by the variable $PS1

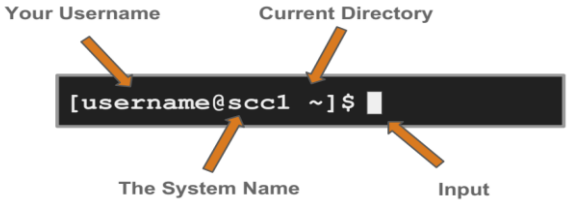



**Figure 1.5 : The "prompt"**

### I.5.3. Linux Commands

| **Help with Commands** | |
|---|---|
| **man ls** | Call help (documentation) for the "**ls**" command |
| **-h** or **--help** | Request help for a command |
| **ls --help** | Request help for the "**ls**" command |
| **clear** | Delete the terminal visible from all previous outputs. |
| **Handling Files and Directories (Folders)** | |
| **pwd** | Display the path of the current directory |
| **ls** | List files and directories |
| **ls -a** | List all files and directories |
| **ls -l** | List all files in the directory |
| **cd** | Change (Position) to home-directory |
| **cd ~** | Change (Position) to home-directory |
| **cd ..** | Change (Position) to parent directory |
| **cd prog** | Change (Position) to named "**prog**" |
| **mkdir prog** | Make (Create) a "**prog**" directory |

| | |
|---|---|
| **rmdir prog** | Deleting the "**prog**" directory |
| **cp prog1.c to prog2.c** | Copy file "**prog1.c**" to "**prog2.c**" |
| **rm prog1.c** | Delete file "**prog1.c**" |
| **mv prog1.c prog2.c** | Rename or move file "**prog1.c**" to "**prog2.c**" |
| **file prog.c** | Type of file "**prog.c**" |
| **wc prog.c** | Count number of lines, words, characters in file "**prog.c**" |
| **cat prog.c** | Display of "**prog.c**" file contents |
| **cat a.txt >> b.txt** | Copy file "**a.txt**" to the end of file "**b.txt**" |
| **more prog.c** | List of contents of "**prog.c**" file, stop at bottom of screen |
| **less prog.c** | List of "**prog.c**" file contents, more enhancement |
| **head prog.c** | Display the first few lines of a "**prog.c**" |
| **tail prog.c** | display the last few lines of a "**prog.c**" |
| **grep "main" prog.c** | Displays all lines in the "**prog.c**" file containing "**main**" |
| **sort a.txt** | Sorting the "**a.txt**" file |
| **cmp a.txt b.txt** | Compare two files "**a.txt**" wit "**b.txt**" |
| **diff a.txt b.txt** | Displays the differences between the two files |
| **touch fich.txt** | Creates an empty file of this name if it doesn't exist, otherwise changes the file's last modified date. |
| **chmod [options] file** | Change access rights for named "**file**" |
| **Compression and Archiving** | |
| **tar tzvf prog.tar.gz prog** | List (v) of the table (t) of files in the "**prog.tar.gz**" archive |
| **tar czf prog.tar.gz prog** | Create (c) an archive file (f) "**prog.tar.gz**" compressed (z) from all "**prog**" tree files |
| **tar xzf prog.tar.gz prog** | Extract (x) files from the "**prog.tar.gz**" archive |
| **gzip fich.txt** | Compressing "**fich.txt**" to "**fich.txt.gz**" |
| **gunzip fich.txt.gz** | Uncompressing "**fich.txt.gz**" to "**fich.txt**" |
| **gzip -d fich.txt.gz** | Uncompressing "**fich.txt.gz**" to "**fich.txt**" |
| **Session Management** | |
| **passwd** | Change password |
| **who** | Logged-in (Connected) users |
| **w** | Users logged in and action in progress |
| **id** | uid and gid (user and group number) |
| **whoami** | User Id of current session |
| **h** | Command history |
| **"** | Previous command |
| **echo "Bonjour"** | Display "**Bonjour**" string |
| **echo $PATH** | Display command path |

| | |
|---|---|
| **printenv** | Display of environment variables |
| **alias** | List of aliases |
| **tty** | Terminal name |
| **export LANG=fr FR** | French diagnostics |
| **export LANG=C** | English diagnostics |
| **locale** | Displays local language options |
| **exit** | Quit shell (or session) |
| **logout** | Quit shell (or session) |
| **Compilation** | |
| **gcc -o prog.o -c prog.c** | Compile the "**prog.c**" file and create the "**prog.o**" object file. |
| **gcc -o prog prog.o** | Compile the "**prog.o**" file and give the executable in the "**prog**" file. |
| **gcc -o prog prog.c** | Compile the "**prog.c**" file and give the executable in the "**prog**" file. |
| **gcc -o prog prog.c - lm** | Compile the "**prog.c**" file and give the executable in the "**prog**" file, with search for functions in the math library |
| **gcc prog.c** | Compile the "**prog.c**" file and give the output file as "**a.out**" file which is default name of output file given by gcc compiler. |
| **prog** or **./prog** | Execution of "**prog**" program |
| **Make** | Execute "**Makefile**" commands |
| **ldd prog** | Shared libraries called by executable program prog |
| **nm prog** | The "**prog**" executable program symbols |
| **gdb prog** | Search for "**prog**" executable program errors |
| **strace date** | Trace **date** command system calls |
| **strip prog** | Remove symbols from "**prog**" executable program |
| **Process Management** | |
| **ps auxr** | List of running processes |
| **ps aux\|more** | List of all processes |
| **top** | Machine activity monitoring |
| **&** | Process background |
| **prog &** | Running a "prog" in background |
| **jobs** | Display list of background jobs |
| **kill %1** | Kill background job [1] |
| **kill 2025** | Kill process with PID = 1492 |
| **bg %1** | allows processes to be managed and moved jobs between foreground and background. |
| **fg %1** | Used to bring a background job into the foreground. |

| | |
|---|---|
| **free** | Available memory space |
| **Command Combinations** | |
| **echo "Bonjour" > fich.txt** | Writing "**Bonjour**" in "**fich.txt**" |
| **ls > list.txt** | Send list of directory files in "**list.txt**" |
| **ls >> list.txt** | Send list of directory files in "**list.txt**", but copied to the end of "**list.txt**" |
| **\|** | Sending the output of one command to the input of the next. |
| **ls\|wc -l** | Number of files in current directory (**wc -l** counts lines displayed by **ls**) |
| **Disk space management** | |
| **df** | Space occupied/available on mounted disks |
| **mount** | Display of mounted disks |
| **mount /cdrom** | CD-ROM mounting |
| **Print** | |
| **lpr -Phpv prog.ps** | Print "**prog.ps**" file on **hpv** |
| **lpq** | Queue on default printer |
| **lpq -Plp** | Queue on **lp** printer |
| **lprm -Plp 356** | Deleting job "**356**" from the **lp** queue |
| **a2ps fichier.txt** | Print text file "**file.txt**" on Postscript printer |
| **Temps** | |
| **date** | Date and time |
| **cal** | Calendar for the current month |
| **cal 6 2025** | June 2025 calendar |
| **calendar** | Calendar management |

## I.5.4. Permissions and Ownership

The concept of Linux File **permission** and **ownership** is crucial in Linux. Here, we will explain Linux permissions and ownership. Let us start with the **permission**.

**Linux File Permissions**

There are three kinds of file permissions in Linux **Read**, **Write**, and **Execute**.

- **Read (r):** Provides access to open and view the contents
- **Write (w):** Provides access to modify (or delete) the content or file
- **Execute (x):** Provides access to run the file as a program

**Linux File Ownership**

The three distinct user-based permission groups in Linux are: **owner**, **group**, and **all users (Others).**

- **Owners:** These permissions apply exclusively to the individuals who own the files or directories.
- **Groups:** Permissions can be assigned to a specific group of users, impacting only those within that particular group.
- **All Users (Others):** These permissions apply universally to all users on the system, presenting the highest security risk. Assigning permissions to all users should be done cautiously to prevent potential security vulnerabilities.

All three owners (user owner, group, others) in the Linux system have three types of permissions defined. Nine characters denote the three types of permissions.

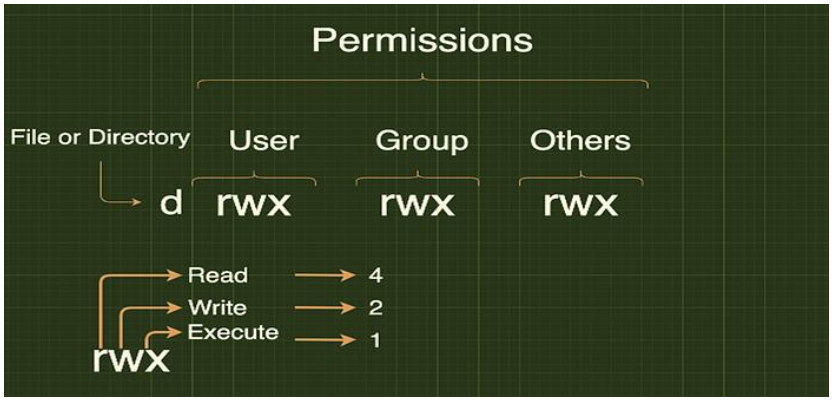



**Figure 1.4 : Basic Concepts of Permissions**

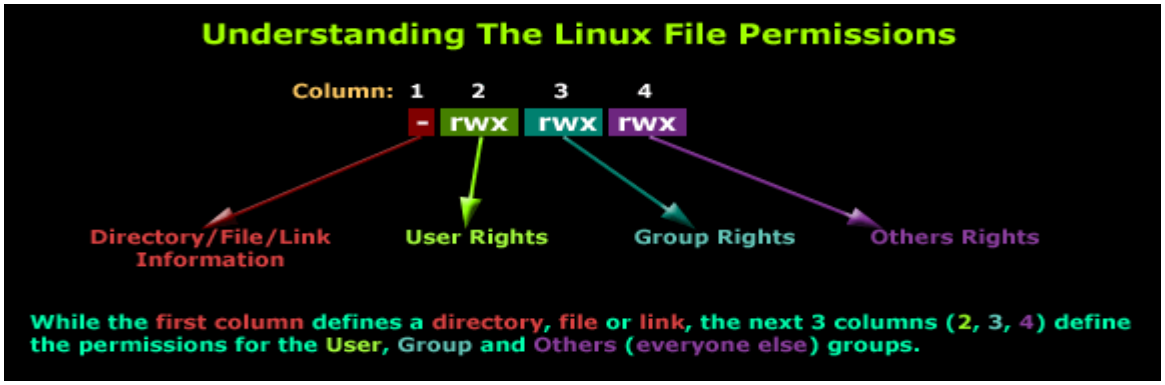
Access rights are displayed (**ls -l**) by a series of **9** characters, associated **3** by **3** (**rwx rwx rwx**) defining the rights of the **3** identities (**u**, **g** and **o**) and A tenth character indicates the **4** file types : **-** : ordinary file, **d** : directory , **l** : Symbolic link, and **c** (or **b**) : Special.

## Example 1 :

When you type in **ls -la** or **ls -l** (listing all files in current directory with details) we usually see something like this:

```
jpp@jpp: /boot
jpp@jpp:/boot$ ls -la
total 39132
drwxr-xr-x  3 root root     4096 2011-05-13 08:52 .
drwxr-xr-x 23 root root     4096 2011-05-04 09:27 ..
-rw-r--r--  1 root root   700761 2011-03-18 16:33 abi-2.6.35-28-generic
-rw-r--r--  1 root root   730039 2011-04-11 01:24 abi-2.6.38-8-generic
-rw-r--r--  1 root root   122616 2011-03-18 16:33 config-2.6.35-28-generic
-rw-r--r--  1 root root   130313 2011-04-11 01:24 config-2.6.38-8-generic
drwxr-xr-x  3 root root    12288 2011-05-04 09:32 grub
-rw-r--r--  1 root root 11008098 2011-04-15 08:58 initrd.img-2.6.35-28-generic
-rw-r--r--  1 root root 13134896 2011-05-13 08:52 initrd.img-2.6.38-8-generic
-rw-r--r--  1 root root   160988 2010-10-22 09:08 memtest86+.bin
-rw-r--r--  1 root root   163168 2010-10-22 09:08 memtest86+_multiboot.bin
-rw-r--r--  1 root root  2344143 2011-03-18 16:33 System.map-2.6.35-28-generic
-rw-------  1 root root  2654256 2011-04-11 01:24 System.map-2.6.38-8-generic
-rw-r--r--  1 root root     1336 2011-03-18 16:35 vmcoreinfo-2.6.35-28-generic
-rw-------  1 root root     1368 2011-04-11 01:26 vmcoreinfo-2.6.38-8-generic
-rw-r--r--  1 root root  4342384 2011-03-18 16:33 vmlinuz-2.6.35-28-generic
-rw-------  1 root root  4523936 2011-04-11 01:24 vmlinuz-2.6.38-8-generic
jpp@jpp:/boot$
```

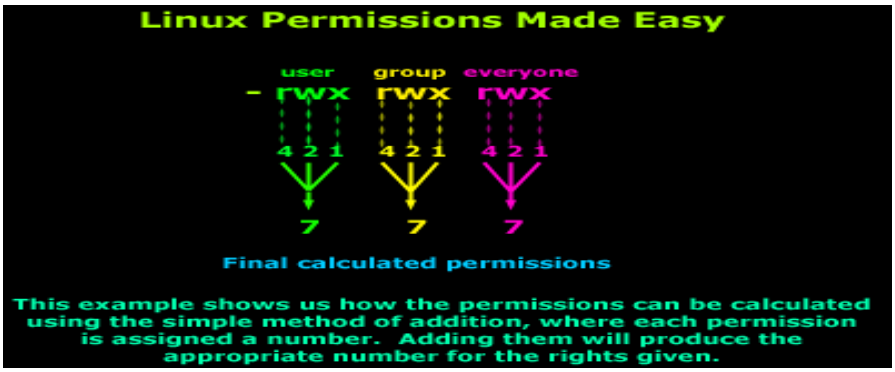
The first column represents the file type and permissions. Therefore, we will focus on the first column, which is the file permissions.



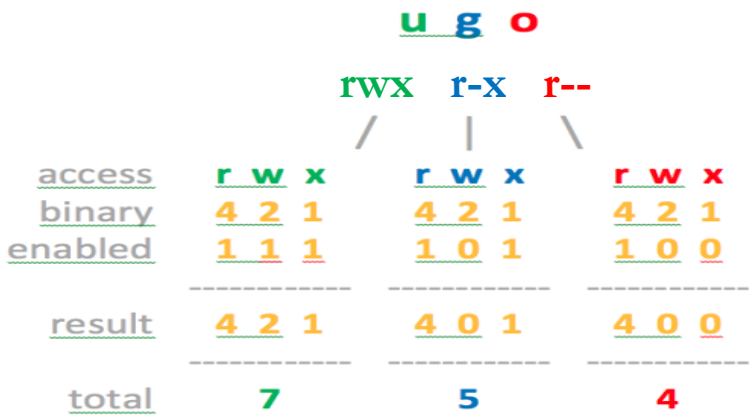

**Eg: - rwx r-x r--** (file **test.txt**)

- **-**: type **-** file (directory would be 'd')
- **rwx**: permission for user who created the file **-** read, write, and execute
- **r-x**: permission for the group the user belongs to **-** only read, and execute
- **r--**: permission for others **-** only read

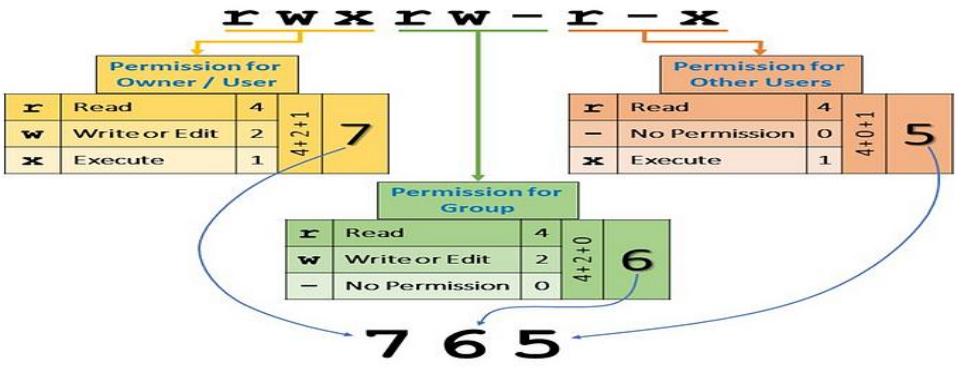
The permissions of read, write, execute also have numeric values.




We can calculate the numerical permission in the above example (**- rwx r-x r--**) as follows:




**Example 2 :**



## I.5.5. How to Change Permissions in Linux

There are two primary methods for changing permissions in Linux:

- **Absolute (Numeric) mode and**
- **Symbolic mode.**

Each method offers its own approach to specifying permissions and provides flexibility in access control. The command used in both methods is the chmod command. Its basic syntax is:

```
chmod [permission] [file_name/directory]
```

The sections below explain each method for managing permissions in your Linux environment.

**Symbolic Mode**

To specify permission settings using alphanumerical characters, you need to define accessibility for the user/owner (**u**), group (**g**), and others (**o**).

Type the initial letter for each class, followed by the equal sign (**=**) and the first letter of the read (**r**), write (**w**) and/or execute (**x**) privileges.

For example, to set a file public for reading, writing, and executing, use:

```
chmod u=rwx,g=rwx,o=rwx [file_name]
```

To set the permissions as in the previously mentioned ***test.txt*** to be:

- Read write, and execute for the user.
- Read and execute for the members of the group.
- Read for other users.

Use the following command:

```
chmod u=rwx,g=rx,o=r test.txt
```

**Absolute (Numeric) Mode**

Another way to specify permission is by using the octal/numeric format. This option is faster, as it requires less typing, although it is not as straightforward as the symbolic mode. Instead of letters, the octal format represents privileges with numbers:

- **r**(ead) has the value of **4**.
- **w**(rite) has the value of **2**.
- (e)**x**(ecute) has the value of **1**.
- **no permission** has the value of **0**.

The privileges are summed up and depicted by one number. Therefore, the possibilities are:

- **7** - for read, write, and execute permission.
- **6** - for read and write privileges.
- **5** - for read and execute privileges.
- **4** - for read privileges.

As you have to define permission for each category (user, group, owner), the command includes three numbers (each representing the summation of privileges).

For instance, let's look at the **test.txt** file that we symbolically configured with the **chmod u=rwx,g=rx,o=r test.txt** command.

The same permission settings can be defined using the octal format with the command:

```
chmod 754 test.txt
```

## I.6. Environment Variables

An environment variable is a user-definable and is set outside the program, value that can affect the way running processes will behave in a computer. Environment variables are part of the environment in which a process runs and can store information about the system environment, user preferences, or various configurations that programs can access at a point in time.

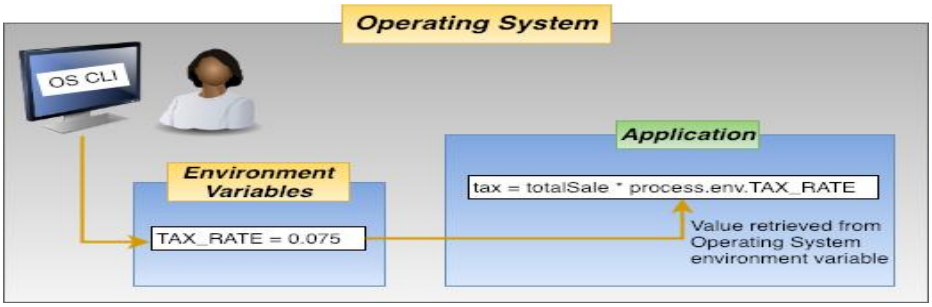



**Figure 1.4 : OS Environment Variables**

The most well-known environment variable is probably PATH which contains the paths to all folders that might contain executables. With PATH, we can write just the name of an executable in our terminal, rather than its full path, since the shell will check the local directory as well as all directories specified in the PATH variable for this executable.

The command used to display all the environment variables defined for a current session is **env**.

Environment variables are used by shell to store information useful for commands and software used during the working session. As shell is true programming languages, users can define the variables they want. The following are the main variable management commands (in bash syntax).

| Operation | Syntax | Examples |
|---|---|---|
| **Assignment** | VARIABLE_NAME | NOM=Kamel<br>X='$PATH='$PATH Liste ='ls'<br>PC=`hostname: `$USER |
| **Display** | **echo** $VARIABLE_NAME<br>**printenv** VARIABLE_NAME | echo $NOM<br>printenv $List |
| **Export** | **export** VARIABLE_NAME =Value<br>**set** VARIABLE_NAME =Value | export NOM = Said |
| **Destruction** | **unset** VARIABLE_NAME<br>VARIABLE_NAME =" | unset NOM |

Syntax must be respected. For example, the following assignment instruction is incorrect: **i = 1**. It must be written without spaces **i=1**. Spaces are significant.

By default, a new variable is only visible in the shell where it was created. It is called a local variable. Exporting a variable makes it public; and accessible to other software.