What is a desktop

The term desktop refers to the graphical environment where you do your work. The desktop usually consists of a workspace (called the root window) with pretty icons and quite possibly a menu that pops up when you click on it, usually a panel on the top or the bottom and/or top of the screen with a menu and a lot of other practical services you may never even notice.

The most important part of any desktop is the window manager this is the application that handles window placements and movements. The window manager is what draws a border (or no border) around your windows and makes them maximize, minimize, moves and behave according to your preferences.

K Desktop Environment (KDE)

K Desktop Environment (KDE) is an Open Source graphical desktop environment for UNIX workstations. Initially called the cool Desktop Environment, KDE is an on-going project with development taking place on the Internet and discussion shield through the official KDE mailing list, numerous newsgroups, and Internet Relay Chat (IRC) channels. KDE has a complete graphical user interface (GUI) and includes a file manager, a window manager, a help system, a configuration system, tools and utilities, and several applications. The most popular suite of KDE applications is KOffice, which includes a word processor, a spread sheet application, a presentation application, a vector drawing application, and image editing tools.

GNOME (GNU Network Object Model Environment)

GNOME (GNU Network Object Model Environment, pronounced gah-NOHM) is a graphical user interface (GUI) and set of computer desktop applications for users of the Linux computer operating system. It's intended to make a Linux operating system easy to use for non-programmers and generally corresponds to the Windows desktop interface and its most common set of applications. In fact, GNOME allows the user to select one of several desktop appearances. With GNOME, the user interface can, for example, be made to look like Windows 98 or like Mac OS. In addition, GNOME includes a set of the same type of applications found in the Windows Office 97 product: a word processor, a spread sheet program, a database manager, a presentation developer, a Web browser, and an e-mail program.

2. KDE vs. Gnome, What is the better desktop environment?

KDE and Gnome are complete desktop environments that consist of a large number of tightly integrated yet still separate pieces of software. GNOME uses a window manager called metacity, KDE uses kwin. Both these desktops can be used with any other window manager if you do not like the default choice.

Linux is like Lego. You can build your own desktop environment. Both KDE and Gnome are just big packages with software aimed to look and feel the same way, work well together and generally give you a nice experience. If you dislike a component, then replace it with something else. It's that simple.

Application that are "made for Gnome" or "made for KDE" can be used with any desktop. This only means that that the program use a set of library functions found in their underlying gnome-libs or kdelibs. You do not need to use the actual desktops to use the applications, software made for KDE and Gnome can be used with any window manager / desktop as long as you got the proper libraries installed. There is no reason to use only applications made for the desktop you prefer, the "best software" for one task is made for KDE, the best for another task is made for Gnome. Use the best from both worlds.

Both KDE and Gnome can be customized to behave exactly the way you want. What desktop you prefer is your own choice and preference. When in doubt, try to learn both. Or experiment with other desktops. Remember, *nix applications are not locked to the desktop they are made for, Gnome applications can be used in KDE and vice versa.

There is no "best desktop", but there is a desktop that's best for you. It's a matter of preference, and hardware.

Linux Operating System and its features

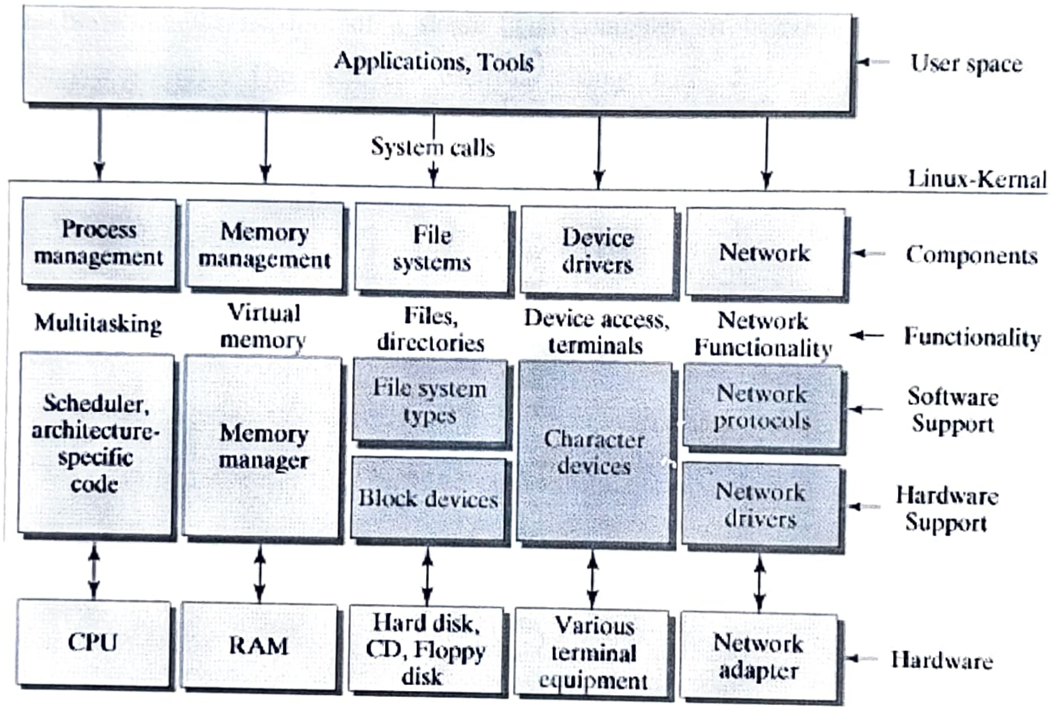
Linux is completely different from other operating systems in many ways. It is an open source OS which gives a great advantage to the programmers as they can design their own custom operating systems. It gives you a lot of option of programs having some different features so you can choose according to your need. A global development community look at different ways to enhance its security, hence it is highly secured and robust so you don't need an anti virus to scan it regularly. Companies like Google, Amazon and Facebook use linux in order to protect their servers as it is highly reliable and stable. Above all you don't have to pay for software and server licensing to install Linux, its absolutely free and you can install it on as many computers as you want. Its completely trouble free operating system and don't have an issue with viruses, malware and slowing down your computer.

- Portable – Portability means softwares can works on different types of hardwares in same way.Linux kernel and application programs supports their installation on any kind of hardware platform.
- Open Source – Linux source code is freely available and it is community based development project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- Multi-User – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- Multiprogramming – Linux is a multiprogramming system means multiple applications can run at same time.
- Hierarchical File System – Linux provides a standard file structure in which system files/ user files are arranged.
- Shell – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

- Live CD/USB: Almost all Linux distributions have Live CD/USB feature by which user can run/try the OS even without installing it on the system.

- Graphical user interface (X Window System): People think that Linux is a command line OS, somewhere its true also but not necessarily, Linux have packages which can be installed to make the whole OS graphics based as Windows.
- Support's most national or customized keyboards: Linux is used worldwide and hence available in multiple languages, and supports most of their custom national keyboards.
- Application Support: Linux has its own software repository from where users can download and install thousands of applications just by issuing a command in Linux Terminal or Shell. Linux can also run Windows applications if needed.

Linux System Architecture is consists of following layers:



- **Hardware layer** – Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** – Core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** – An interface to kernel, hiding complexity of kernel's functions from users. Takes commands from user and executes kernel's functions.
- **Utilities** – Utility programs giving user most of the functionalities of an operating systems.

Linux provide 3 main security concepts are:

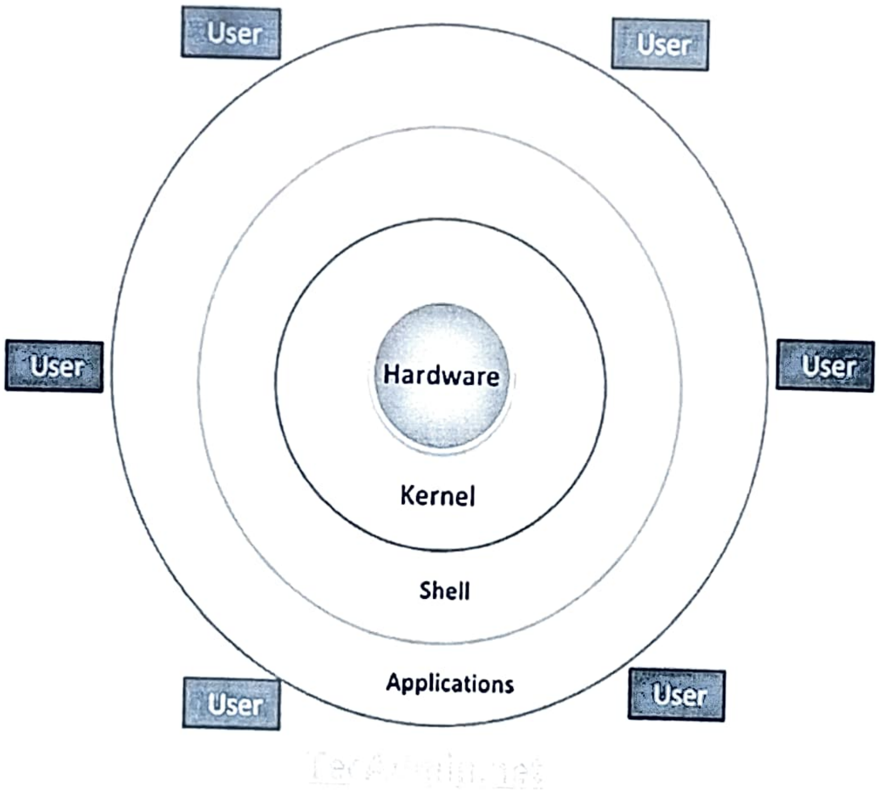
- **Authentication:** This simply implies claiming the person whom you are by assigning passwords and login names to individual users, ensuring that nobody can gain access to their work.
- **Authorization:** At the file level Linux has authorization limits to users, there are read, write and execute permissions for each file which decide who can access a particular file, who can modify it and who can execute it.
- **Encryption:** This feature encodes your files into an unreadable format that is also known as 'cyphertext', so that even if someone succeeds in opening it your secrets will be safe.

Communication: Linux has an excellent feature for communicating with the fellow users, it can be within the network of a single main computer, or between two or more such computer networks. The users can easily exchange mail, data, program through such networks.

Linux Architecture

Linux is a free and open-source operating system that was developed in the early 1990s by Linus Torvalds. It is based on the Unix operating system and has become a popular choice for both personal and enterprise use due to its stability, security, and flexibility.

Linux is built on a modular architecture, which means that it is made up of a number of different components that work together to form a complete operating system. These components are organized into layers, each of which serves a specific purpose and interacts with the other layers to provide the functionality that users expect from an operating system.



Linux System Architecture

The following is a high-level overview of the main layers of the Linux architecture:

- **Hardware layer:** This is the bottommost layer of the Linux architecture and represents the physical hardware components of the computer, such as the processor, memory, and storage. The hardware layer is responsible for interacting with the various hardware devices and providing access to them for the rest of the operating system.
- **Kernel layer:** The kernel is the core of the operating system and is responsible for managing the resources of the computer, such as the CPU, memory, and I/O devices. It also provides services to the other components of the operating system and acts as the intermediary between the hardware and the software layers.
- **System libraries layer:** This layer consists of a set of libraries that provide functions for the applications to use. These libraries include system calls, which are used to invoke kernel functions, as well as other functions that perform tasks such as file manipulation, networking, and memory management.
- **System utilities layer:** This layer consists of a set of programs that perform various system-level tasks, such as managing processes, controlling user accounts, and configuring system settings. These utilities are usually command-line programs that are invoked by the user or by other programs.
- **Desktop environment layer:** This layer is optional and is not present on all Linux systems. It provides a graphical user interface (GUI) that allows users to interact with the operating system using a mouse and keyboard. The most common desktop environments in Linux are Gnome, KDE, and Xfce.
- **Applications layer:** This is the topmost layer of the Linux architecture and consists of the various applications that run on the operating system. These can be anything from productivity software and games to web browsers and media players.

Linux Kernel

The Linux kernel is the core of the Linux operating system and is responsible for managing hardware resources, networking, and system memory. It is a free and open-source software project that was first released in 1991 by Linus Torvalds.

Since its inception, the Linux kernel has undergone significant development and evolution. It is developed and maintained by a global community of developers and is released under the GNU General Public License (GPL).

One of the key features of the Linux kernel is its modular design, which allows developers to add or remove features as needed without having to rebuild the entire kernel. This makes it easy to customize the kernel for specific purposes and to incorporate new technologies as they become available.

The Linux kernel is organized into different layers, with the lowest layer (the "core") responsible for basic functions such as memory management and device drivers. The upper layers (the "shell") provide interfaces for applications to interact with the kernel and manage resources such as processes, files, and networks.

The Linux kernel has undergone numerous versions and updates since its inception. Major versions are released on a regular basis, with each version typically containing new features and improvements. Some of the most significant versions of the Linux kernel include:

- Linux kernel 1.0 (released in 1994): This version comes with various features you would expect in a modern fully-fledged Unix, like: virtual memory, true multitasking, and shared libraries, etc.
- Linux kernel 2.6 (released in 2003): This version introduced support for large file systems, improved support for multiprocessor systems, and enhanced security features.
- Linux kernel 3.0 (released in 2011): This version introduced support for the ext4 file system and improved power management features.
- Linux kernel 4.0 (released in 2015): This version introduced support for the btrfs file system and improved support for containers.
- Linux kernel 5.0 (released in 2019): This version introduced support for the WireGuard virtual private network (VPN) protocol and improved support for graphics processing units (GPUs).
- Linux kernel 6.0 (released in 2022): This version introduced support for Intel's fourth generation Xeon server chips Sapphire Rapids, and their 13th generation Raptor Lake core chips.

The development of the Linux kernel is a continuous process, with new versions and updates being released on a regular basis. The Linux community is constantly working to improve the kernel and to incorporate new technologies and features.

Overall, the Linux kernel is a vital component of the Linux operating system and plays a critical role in the development and evolution of the open-source community.

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:

```
./configure
make
make install
```

dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

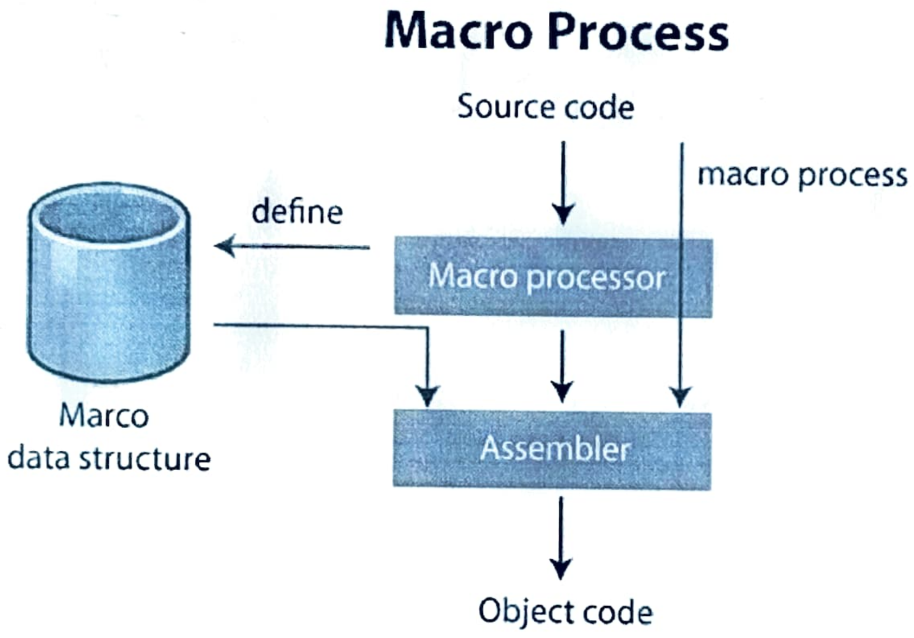
Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

Macro Processor

A Macro Processor is a system software that plays a crucial role in programming. It represents a group of commonly used statements in the source programming language. The Macro Processor replaces each macro instruction with the corresponding group of source language statements. This operation is known as the expansion of macros.

Using Macro instructions, a programmer can leave the mechanical details to be handled by the macro processor. This makes writing code more convenient as it allows for defining new language constructs that can be expressed in terms of existing language components.



The working of a Macro Processor involves several steps:

1. Macro Definition

A macro definition is a section of code that a programmer writes (defines) once. The macro is defined by a #define directive. Each parameter in the macro definition begins with the character &. The macro definition may be written using formal parameters. Here's an example of a macro definition in C/C++:

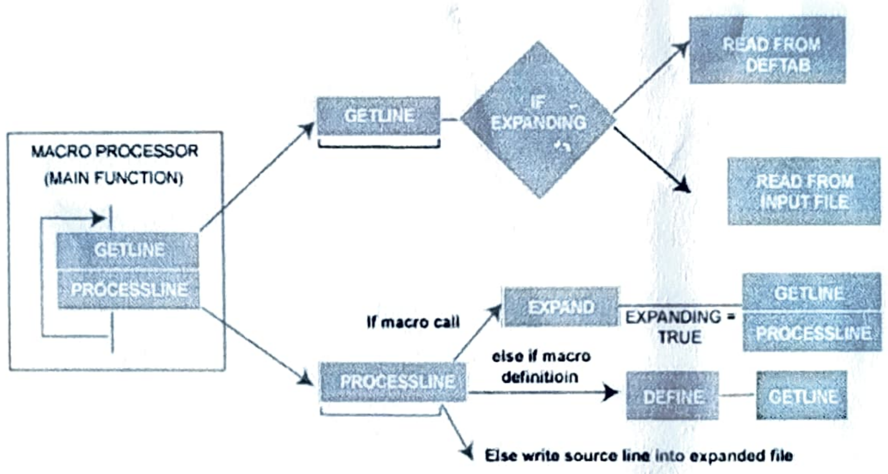
2. Macro Invocation

Macro invocation is whilst the macro is known as or invoked. The macro call gives the call of the macro education being invoked and specifies the arguments for expanding the macro.

3. Macro Expansion

This is the process where the Macro Processor replaces each macro instruction with the corresponding group of source language statements. This operation is known as the expansion of macros. Using Macro instructions, a programmer can leave the mechanical details to be handled by the macro processor.

1. **Macro Assembler:** It expands each macro call in a program into a sequence of assembly language statements and assembles the resultant assembly language program.
2. **Macro Pre-processor:** It only processes the macro call. Other statements are processed with the help of an assembler. A macro pre-processor merely performs the expansion of a macro in a program.



Salient Features of Macro Processors

Macro Processors have several salient features:

- Macro represents a group of commonly used statements in the source programming language.
- Macro Processor designs are not directly related to the computer architecture on which it runs.
- Macro Processor involves definition, invocation, and expansion.

Tasks involved in macro expansion:

1. Identify the macro calls in the program.
2. The values of formal parameters are identified.
3. Maintain the values of expansion time variables declared in a macro.
4. Expansion time control flow is organized.
5. Determining the values of sequencing symbols.
6. Expansion of a model statement is performed.

• Each macro invocation statement will be expanded into the statements that form the body of the macro. Arguments from the macro invocation are substituted for the parameters in the macro prototype (according to their positions).

1. In the definition of macro: Parameter.
2. In the macro invocation: Argument.

- Comment lines within the macro body will be deleted. Macro invocation statement itself has been included as a comment line. The label on the macro invocation statement has been retained as a label on the first statement generated in the macro expansion. We can use a macro instruction in exactly the same way as an assembler language mnemonic.
- Each parameter begins with the character and which facilities the substitution of parameters during macro expansion.
- The macro name and parameters define a pattern or prototype for the macro instruction used by the programmer. Body of the macro definition is defined by the MACRO directive statements. Macro expansion generate these statements. End of the macro definition is defined by the MEND assembler directive.

- Macro invocation statement that gives the name of the macro instruction being invoked and the arguments to be used in expanding the macro.

Each macro invocation statement has been expanded into the statements that form the body of the macro, with the arguments from the macro invocation substituted for the parameters in the macro prototype. Parameters and arguments, both are associated with one another according to their positions. After macro processing, the expanded file can be used as input to the assembler.