Unit 13 : Editors, Shell and Shell Scripts

Lesson 1 : Editors

1.1. Learning Objectives

On completion of this lesson you will be able to know:

- About necessity of a text editor
- About the different editors in Linux
- About vi editor
- About the commands of the vi editor

1.2. The need of a text editor

A text editor is used to create and manage text files and documents.

A text editor is used to create and manage text files and documents. An editor is application software, which is usually bundled with an operating system (OS). A text editor has a wide variety of application, such as:

- Creating and Maintaining Documents
- Writing programs and utilities
- Writing mail messages

1.3. The editors available in Linux

There are various editors available with Linux. Some of them are:

- vi (visual editor)
- vim (visual editor improved)
- emacs (edit macros editor)
- sed (stream editor)
- ed (line editor)
- red(restricted ed editor)
- joe (Joe's own editor)
- rjoe (restricted joe editor)
- pico (pine composer)
- jpico (version of the joe editor, which emulates the pico editor used in the pine mail program)
- jstar (version of the editor compatible with WordStar)

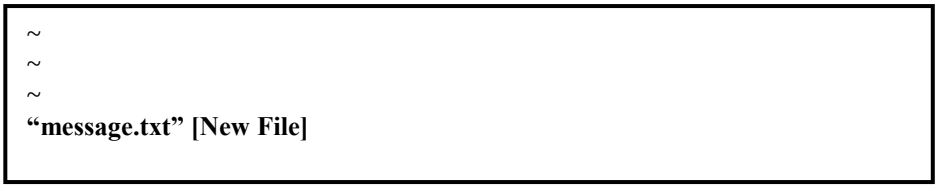
Among the above editors, **vi** is the most widely used. You will learn about **vi** editor in detail in this section. It is capable of handling large files.

1.4. The vi editor

The **vi** editor is a visual editor used for creating and editing text files containing data, documents, or programs. It displays the contents of files on the screen and allows a user to add, insert, delete, or change parts of the text. It is amongst the oldest editors available on UNIX platforms and is available for all the' variants of Unix/Linux.
In Red Hat Linux, **vi** is a symbolic link to the **vim** editor. The **vim** editor is an improved version of the **vi** editor. The view editor is another symbolic link to the **vim** editor, and is used to open the file in the read-only mode. Therefore, you can start the vim editor by typing the **vi** or view command. However, the view command opens the file in the read-only mode**.**

Getting Started with vi
The vi editor is invoked by giving the following command at the Linux prompt:
Syntax:
vi filename <Enter>

Example
[Odroho@localhost Odroho] # **vi message.txt**
If the file, message.txt, does not exist, vi display the following screen:



A New File Created with vi Editor

If the file exists, its contents are read from memory and displayed on the screen.
The **vi** editor works in two modes, the *insert* mode and the *command* mode. In the insert mode, you can insert text to a file. To move from the insert mode to the command mode, you have to press the **<Ese>** The **vi** editor has different commands to add, change, and delete text.

Command	Action
i	Inserts text at current cursor position
a	Appends text after current cursor position

Commands to Enter Text

The commands shown in the table above allow a user to start entering text. To end the insertion or the append mode, you have to press the **<Esc>** key. This takes you back to the command mode.

1.5. The commands of the vi editor

Refer to the table given below for some of the commonly used commands in **vi**.

Command	Action
h	Move to previous character
1	Move to next character
k	Move up one line
j	Move down one line
x	Delete character at current cursor position
dd	Delete line
:wq<Enter>	Save all changes and quit
:w<Enter>	Save the file
:q! <Enter>	Quit without saving changes
:e <filename><Enter>	Open the file specified
:w <filename><Enter>	Write to a different file
:! <command_name><Enter>	Execute a shell command

Commands in vi

1.6. Exercise

1.6.1. Multiple choice questions

a. A text Editor is not used

(i) Create and maintain documents
(ii) To write programs and utilities
(iii) To write main messages
(iv) To edit audio and video files.

b. To end the insertion or the append mode in vi editor, you have to press

(i) <Esc>
(ii) <Enter>
(iii) <Ctrl>
(iv) <Shift>

c. To save all changes and quit from vi editor you use

(i) :q!<Enter>
(ii) :w!<Enter>
(iii) :w<Enter>
(iv) :wq<Enter>

1.6.2. Questions for short answers

a. What is the command used to delete line in vi editor?
b. How do you open the file specified in vi editor?

1.6.3. Analytical question

a. Mention name of editor available with Linux. Discuss one of them.
b. Narrate commonly used commands in vi editor with action.

Lesson 2 : The Linux Shell

2.1. Learning Objectives

On completion of this lesson you will be able to know:

- About the shell of Linux
- About available Linux shells

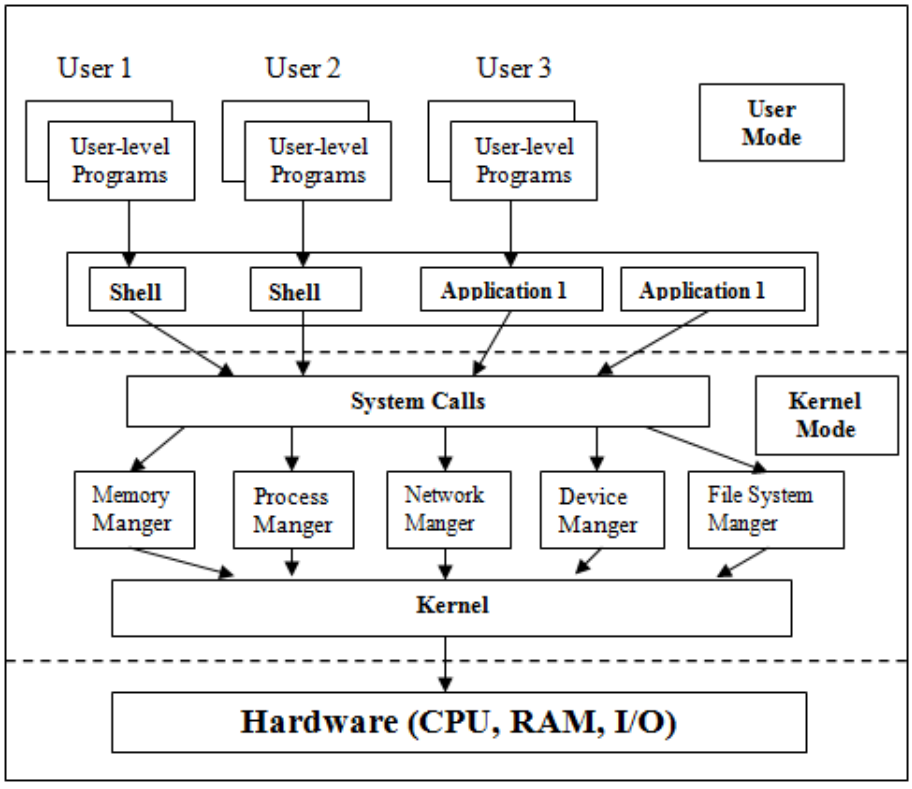
2.2. Introduction to the shell

Linux Shell

In a multi-user environment, the shell has to isolate the user interface from the kernel. This is because when more than one user works on the OS simultaneously, the kernel has to efficiently distribute the processing time of the machine between the active applications. The shell plays the very important role of command interpretation for the kernel and is the interface between the user and the kernel. It is a utility program that comes with the Linux system.

In Linux, the user does not directly interact with the hardware. Instead, the kernel is in charge of the interaction of users with the hardware, scheduling tasks, and allocating resources to the running processes. Each user gets a copy of the shell to work with the kernel. No task is allowed to take direct control of the functioning of the OS.

To understand the importance of the shell and where it fits in the Linux system, let's look at the entire structure of the Linux OS. The figure below gives a complete representation of the functioning of the Linux system.



As you notice in the previous figure, each user gets a copy of the shell where he/she works. The shell interacts with the kernel through the Linux system calls, which are a set of routines that allow an application to access kernel services. Different programs manage each of the services, such as the memory, the processes, the network, the input/output devices, and the file and directory services. These programs are known as daemons. Daemons run at the kernel level and are always active. An example of a daemon is the **cron** daemon, which is used to run commands, scripts, or programs at scheduled times.
The Shell in Linux can be compared to the command.com file in MS DOS, which provides similar functionality. The command.com file is
one of the most important file of MS DOS. In MS DOS, shells are known as "command processors" or "command interpreters". In a Microsoft Windows platform, cmd.exe is an example of a shell.
Though the shell in Linux has always been used as a Command Line Interface (CLI), it is not the only component through which you interact with the system.

2.3. Feature of the Linux shell

Most shells include the following major features:

- **Program execution**: The shell is responsible for the execution of all programs that you request from your terminal. Each time you type in a line to the shell, the shell analyzes the line and then determines what to do.
- **Variable and Filename Substitution**: Like any other programming language, the shell lets you assign values to variables. Whenever you specify one of these variables on the command line, preceded by a dollar sign, the shell substitutes the value assigned to the variable at that point.
  The shell also performs filename substitution on the command line. In fact, the shell scans the command line looking for filename substitution characters *, ?, or [...] before determining the name of the program to execute and its arguments.
- **I/O Redirection:** It is the shell's responsibility to take care of input and output redirection on the command line. It scans the command line for the occurrence of the special redirection characters <, >, or >>.
- **Pipes:** Programs that perform simple functions can easily be connected to perform more complex functions, minimizing the need to develop new programs.
- **Environment Control:** The shell provides certain commands that let you customize your environment. Your environment includes your home directory, the characters that the shell displays to prompt you to type in a command, and a list of the directories to be searched whenever you request.
- **Interpreted Programming Language:** The shell has its own built-in programming language. This language is interpreted, meaning that the shell analyzes each statement in the language one line at a time and then executes it.

2.4. Commonly available shells

The Linux based shells in the freely available distributions are:

- Bash
- Pdksh
- Tcsh
- ASH
- zsh

Bash
Bash is an acronym for 'Bourne Again Shell'. It is an enhancement to the Bourne shell and is the default shell for most Linux systems. The Bash shell is capable of storing the command history of the commands that you have used. Bash is a product of the Free Software Foundation's GNU project. The GNU project consists of a group of voluntary software programmers who create and distribute free software. Bash can also store the commands of you previous sessions. In Red Hat Linux, sh is the symbolic link for the Bash shell, which is stored under the /bin directory.

Pdksh
Pdksh stands for Public Domain Korn Shell and is an enhancement of the Korn shell. It has been written by several voluntary programmers and is currently maintained by Michael Rendell. On Linux Systems, ksh is the symbolic link to the Pdksh shell.

Tcsh
Tcsh stands for Tom's C shell also known as the TC shell, and is an enhancement of the C shell. The symbolic link available for the Tcsh shell on Linux is csh. You can execute the Tcsh shell by typing either csh or tcsh at the command prompt. The C and the TC shell are not compatible with the Bourne shell.

ASH
The A Shell (ash) was developed by Kenneth Almquist of the University of Berkeley. It is a lightweight Bourne shell clone. It is usually suitable on machines that have very limited memory.

zsh
The z shell has the best features of the Tcsh shell. It can emulate all the features of the features of the Korn shell and has the largest number of utilities, with extensive documentation.

2.5. Exercise

2.5.1. Multiple choice questions

a. The shell plays the very important role of

(i) Command interpretation
(ii) Compilation
(iii) Execution
(iv) Creation of program.

b. Which one is Linux shell

(i) Bourn shell
(ii) C shell
(iii) Korn shell
(iv) Bash shell.

2.5.2. Questions for short answers

a. Mention the name of files similar to shell of Linux in MS DOS and Windows.
b. List the name of Linux based shells.

2.5.3. Analytical question

a. Draw the block diagram of architecture of Linux system.
b. Discuss classification of shell.

Lesson 3 : Understanding of shell scripts

3.1. Learning Objectives

On completion of this lesson you will be able to know:

- About shell scripts
- About the execution procedure of shell script
- About the variables of shell scripts

3.2. Shell scripts

If you have a sequence of Linux commands that are used frequently, you can store them in a file. It is then possible to have the shell read the file and executes the commands in it. Such a file is called a script file. Shell scripts allow input/output, manipulation of variables, and have powerful f1ow-of-control and iteration constructs that make programming possible.

3.3. The echo command

The echo command is used to display messages on the screen.

The echo command

Example

```
$ echo "This is an example of the echo command"
```

This is an example of the echo command

The echo command displays the enclosed text on the screen. For better readability, it is preferable to enclose the message to be echoed within double-quotes.

By default, the echo command displays the text and then puts a newline character at the end of the text. The newline character causes the cursor to move to the next line after the text is displayed. Using the -n option, you can keep the cursor on the same line.

Example

```
$ echo -n "This will keep the cursor on the same line"
```

3.4. Executing a Shell Scripts

When you log on to the Linux system, you get a copy of the shell to work in. This shell is known as the login shell. We know that the shell is a utility, so you can execute the shell command (for example, sh for the Bash shell) to create another shell. This new shell is known as the sub-shell or the child shell of the current shell.

The shell creates a child shell (sub-shell) to execute a shell script. This is done so that the current shell is not affected by the script. The shell script is passed to the child shell for execution. The new shell that is created is terminated as soon as the script running on it completes execution.

By default, any file created in Linux does not have the execute permission. This means that until the permissions are changed. the file cannot be directly executed by typing the file name at the shell prompt. Alternatively, a file can be executed by using the sh command (for the Bash shell). The sh command creates a new sub-shell and passes the script name to be executed to this shell. However, to execute any script, you must have the read permission for the script.

You can execute a shell script with a particular shell, by specifying the executable file name of that shell (example, csh for the TC Shell). For example, to execute a script, magic, with the TC shell, the command is:

$ csh magic

You can also specify the script interpreter that should execute the script, in the first line of the script. For example, the following statement in the first line of the script will tell the shell to execute the script in the TC shell.

#!/bin/tcsh

In order to execute a shell script directly at the $ prompt, you can change the File Access Permission (FAP) of the specified shell script by granting the execute permission. Once the execute permission is granted, the shell script can be executed by directly invoking its name at the S prompt. For example, to execute a shell script, magic, the following commands are used.

$ chmod u+x magic Change FAP

$ magic Execute the shell script

When you execute the above shell script, the current shell creates a new shell and executes the script in the newly created shell. You can use the dot (.) command in the Bash shell (source - in case of a C or TC shell), to execute the script in the current shell.

Example

$ • magic	In the Pdksh and Bash shells
$ source magic	In the Tcsh shell

However, there is one major difference between executing the script using the first two methods and this method (with the **.** command). When the above script is executed, no new shell is created and the script is executed in the current shell. This is useful if you want to have access to the variables of the current shell.

If you are using the **.** command to execute the shell script, and you have specified the shell that is to be used for executing the script in the first line of your script, then the shell specified in the script will be ignored, and the script will execute in the current shell.

3.5. Variables in shell scripts

Variables in shell scripts do not have associated data types, which means that, they are not declared as integers or characters. All variables in Linux are treated as character strings. It is possible, however, to mathematically manipulate variables.

Creating Variables

In the Bash shell, variables do not have to be explicitly declared. They can be created at any point of time by a simple assignment of value. A variable can be created without a value by leaving the right-hand side of the assignment operator blank. The syntax for creating a variable is given below:

Syntax

<variable name>=<value>

If the value being assigned contains any deli miters (such as embedded spaces), then it should be enclosed within either single or double quotes.

Example

name = " Nazrul Islam"

If the value does not have spaces, the quotes are optional.

Referencing Variables

The $ symbol is used to refer to the contents of a variable. For example, to assign the value of one variable to another, the command would be:

Variable1 = ${variables2}

The braces are essentially used to delimit the variable name.

Reading a value into a variable

Besides assigning a value to a variable, the shell also allows a user to enter a value from the keyboard into a variable during the execution of a shell script. This is done using the read command.

$ **read fname**

The **read** command can be used at the shell prompt. but is usually used in shell scripts. The read command on execution, waits for the user to enter a value for the variable. When the user presses **<Enter>** after entering the value, the remaining "pan" of the shell script, if any, is executed. Note that the read command does not prompt the user to enter data. To do so, the echo command must be used. The command usage is:

$ **echo "Enter your name"**

$ **read myname**

3.6. Exercise

3.6.1. Multiple choice questions

a. The echo command is used

(i) To refer to the content of a variable.
(ii) To enter a value from the keyboard into a variable
(iii) To execute the shell script.
(iv) To display messages on the screen.

b. Syntax of creating is

(i) <variable name> <value>
(ii) <variable name> = <value>
(iii) <variable name> = =<value>
(iv) <variable name> | <value>.

3.6.2. Questions for short answers

a. What is a shell script?
b. Write the syntax of creating variables.

3.6.3. Analytical questions

a. Discuss the execution procedure of a shell script.
b. How do you read a value into a variable?

Lesson 4 : Graphical user interface (GUI)

4.1 Learning Objectives

On completion of this lesson you will be able to know:

- About the X window system.
- About different types of desktop environments

4.2. The X window system

UNIX was a single-user, command line interface (CLl)-based operating system.

At the time of releasing, UNIX was a single-user, command line interface (CLl)-based operating system. Afterwards, it was developed as a multi-user operating system. UNIX becomes a very powerful and stable network operating system and captures a major share in the network OS market. However, users working on UNIX had to learn and remember numerous commands along with their parameters and syntax. Experienced users in UNIX were satisfied with the speed of the system and were comfortable working on the CLl. But, new users emerged, who were uncomfortable working on the CLl of Unix, and preferred the Graphical User Interface (GUI) available with other operating systems. UNIX had to face tough competition from its GUI rivals. This turned developers of the UNIX system to develop a GUI for UNIX. Becoming a UNIX clone, Linux follows all the standards of GUI for UNIX.

The X Window System was developed in 1984, at the Massachusetts Institute of Technology (MIT), with aid from IBM as part of project "Athena".

X follows the client-server architecture, in which a client requests the server for some service and the server responds to it. A service is a facility provided by the server, such as the display of client data.

The X server is a software program that serves client requests for displaying graphics. X uses a client-server protocol, known as the X protocol. Some of the GNU X servers available are Accelerated X, Metro X, XFree86, and XSuSE. X clients are application programs that use the services of the X server.

The advantages of using the X interface are:

- You can install the client and server software on different machines.
- You can port X client applications to other systems.
- There can be multiple X servers running on more than one machine at a time.

4.3. Window Manager

A window manager is used by a windowing system to keep track of the location of each window on the desktop. It also manages the size and status of each window. The reason for separating the display management from the window management is to give the ability of changing the entire look and feel of the window environment

by changing the window manager. Thus, users have the choice of customizing their desktop to suit their individual needs.

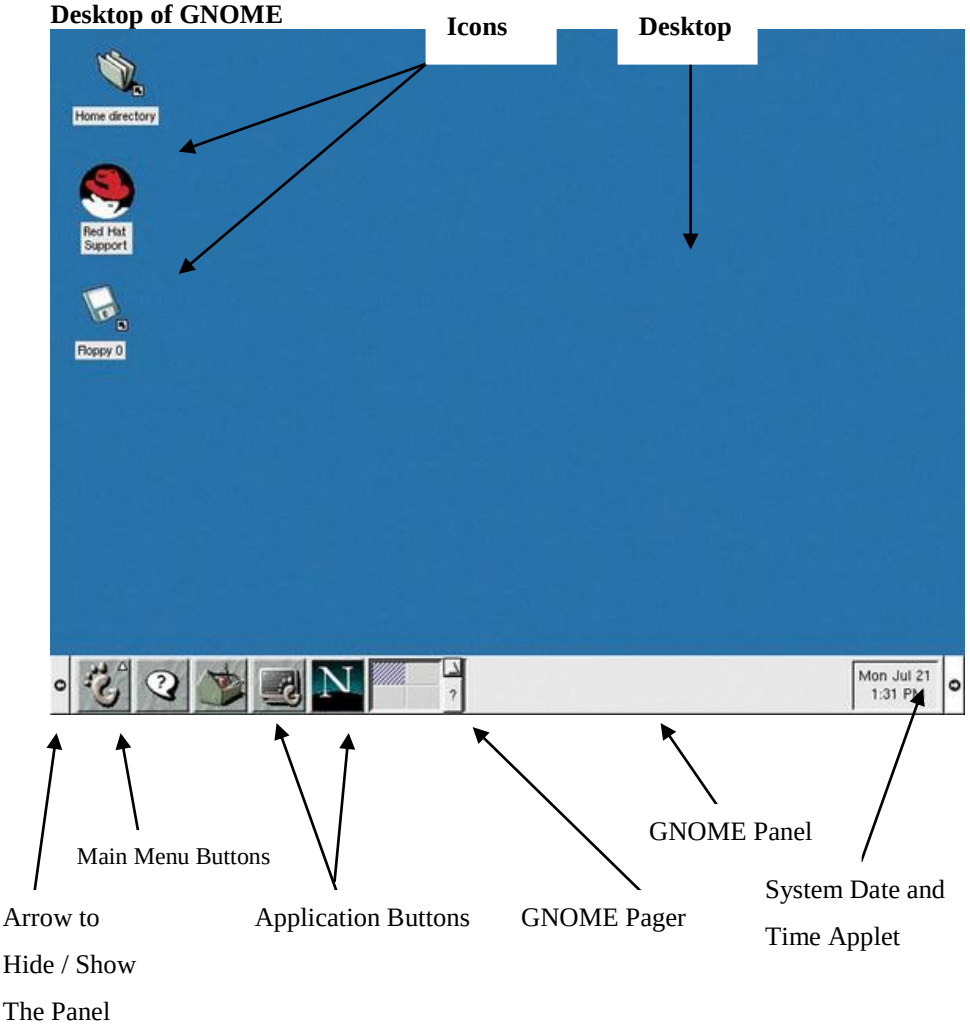
4.4. The X desktop environment

The desktop environment is a user interface that runs on the window manager. It provides a collection of graphical utilities, also known as the X utilities or X clients. Although the window manager may manage the location and the look of the windows, it does not provide utilities for X. This is where a desktop environment comes in use. A window manager customizes the desktop according to your needs. Various desktop environments have been developed for X. A desktop environment is not necessary for the working of the X system. Some popular desktop environments are GNOME, K Desktop Environment (KDE), Common Desktop Environment (CDE), and XFCE.

4.5. The GNOME desktop environment

GNOME is a user-friendly desktop environment that can be run on multiple operating systems. It is the default desktop environment for Red Hat Linux 9. GNOME stands for GNU Network Object Model Environment. It was developed by programmers all over the world and is open source (free) software.

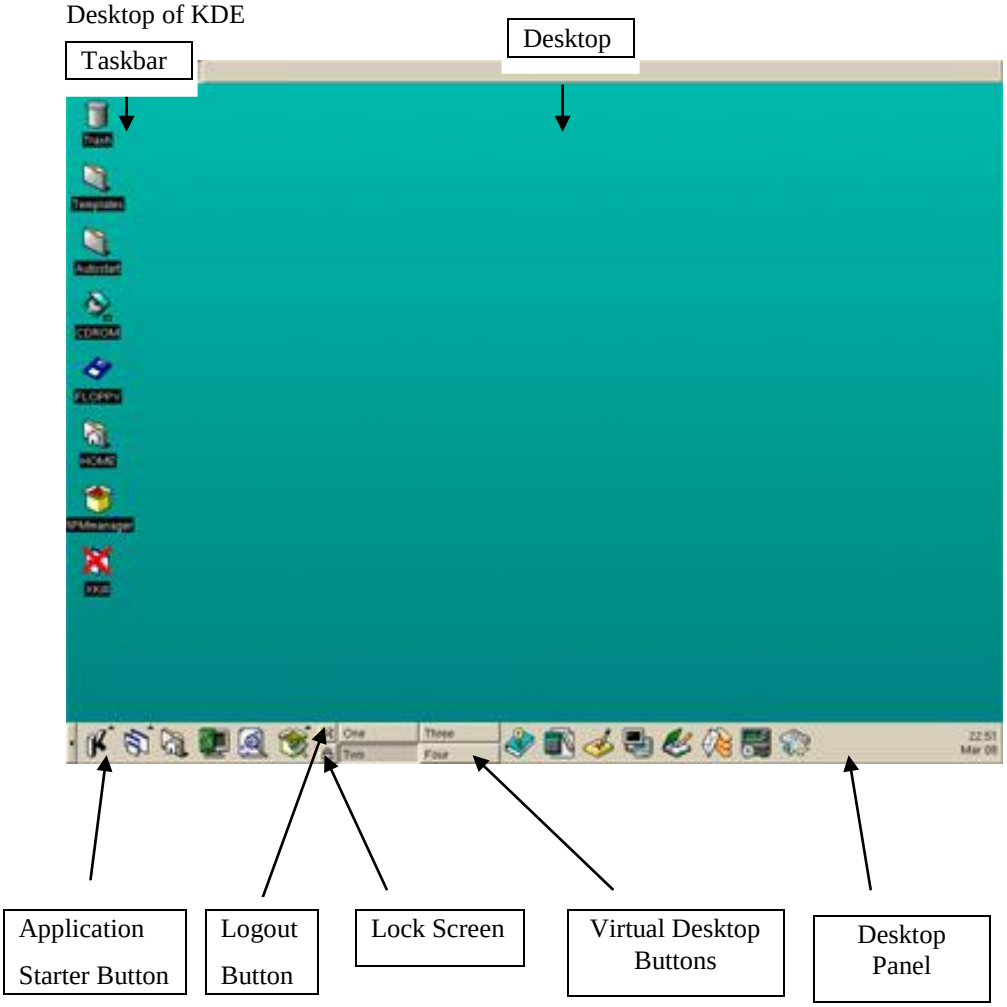
Desktop of GNOME



4.6. The KDE desktop environment

The K Desktop Environment (KDE) is a powerful graphical desktop environment for Linux. KDE provides a complete desktop environment including a window manager and hundreds of X utilities. It uses the K Windows Manager (KWM) as the default window manager, although you can run KDE with different window managers.
Matthias Ettrich, initiated the KDE project in 1996, to give UNIX a powerful GUI environment. KDE is run mostly on Linux machines. However, it can run on other systems like HP-UNIX, Solaris, FreeBSD, and IRIX. KDE gives you online help, from where you can get a lot of information. KDE is free software, and is licensed under the GNU General Public License.

Desktop of KDE



4.7. Exercise

4.7.1. Multiple choice questions

a. GNOME stands for

(i) GNU Network Object Model Environment
(ii) Gnus Network Object Model Environment
(iii) Gtk Network Object Model Environment
(iv) Guarding Naturally Over Mother Earth.

b. KDE stands for

(i) K Desktop Environment
(ii) Kde Desktop Environment
(iii) Kool Desktop Environment
(iv) Knowledge Development Environment.

4.7.2. Questions for short answers

a. What is a windowing system?
b. What does X mean in X Window system of Linux?

4.7.3. Analytical questions

a. Mention the advantages of using the X interface.
b. Describe the GNOME and KDE desktop environments of Linux Operating System.