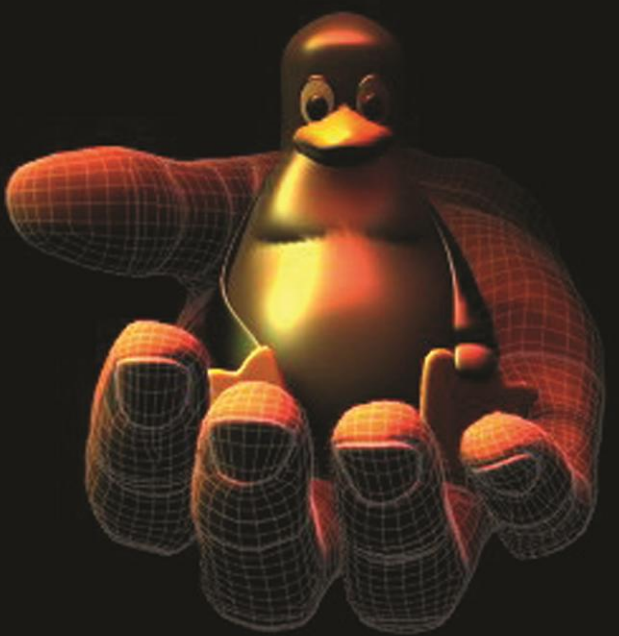
# LINUX
## FOR
## BEGINNERS



*K. Lakshmi Narayana*

NIST Publications

# ■ 1 ■
# Introduction to OS

## 1.1 What is an Operating System (OS)

An operating system is a software program designed to act as an interface between the user and the computer. The primary objective of an operating system is to make computer system convenient to use and utilize computer hardware in an efficient manner. The OS acts as a manager to allocate the computer resources and to schedule tasks. Computer resources includes all the hardware: the Central Processing Unit (CPU), memory, processor, storage devices, file system, and Input/Output devices. It keeps track of the status of each resource and decides who will have a control over computer resources, for how long and when.

## 1.2 Types of Operating Systems

There are two types of OS.

### Single User

A personal Computer (PC) is a popular single–user system. The PC, however, was designed for use by one person at a time, since it is single–user oriented with MS–DOS operating system. Single–user systems became very popular due to the low cost of hardware and the side range of software available for these machines.

**Examples:** DOS/MSDOS (Microsoft Disk Operating System)

### Multi User

Larger Systems, which can be used by more than one person at a time, is known as multi–user systems. Multi–user systems are required when a number of programs have to be run simultaneously, or common resources like printers and disks are to be shared by a number of users. Such an OS is more efficient and more sophisticated than single–user OS.

**Examples:**

- VMS (Virtual Memory System) for VAX machines
- MVS (Multi–user Virtual Memory system) for IBM mainframes
- Solaris for SUN workstations and SPARC machines
- Windows95, 98, 2000, Millenium, XP, NT, 2000 & 2003 Server etc.
- UNIX, and Linux

## 1.3 Open Source Software (OSS)

The term OSS, refers to software in which the source code is freely available to all. It's purpose is to encourage collaborative work often through broad participation in software projects across business and geographical boundaries.

**OSS has the following features:**

1. The source code of the software is freely distributable.
2. All must be able to modify the source code and create derived works.
3. To maintain the integrity of the original author's work, the license may require that changes to the code be provided in patch form.
4. The license must be in inherited, so that those who receive a distribution are subject to the same terms.

**Software is free if it satisfies the four freedoms:**

1. The software must be freely executable for any purpose.
2. The source code must be available so that others can study how it works.
3. The software must be freely redistributable.
4. All are free to modify the software.

## 1.4 C and UNIX

UNIX was written in C language. That's the way world seems to be moving. If you know these two, then your mobility is assured.

### C Programming

**C programming was developed by Dennis Ritchie in AT & T (American Telephone and Telegraph) Bell Labs.** In 1971, Ritchie created the first version of a new programming language based on B, a language he called C.

### UNIX

**UNIX is simple, but it just needs a genius to understand its simplicity. --Dennis Ritchie.**

**Ken Thompson and his group in AT & T Bell Labs developed UNIX in 1969**. UNIX was designed by a couple of programmers as a fun project, and it evolved through the efforts of hundreds of programmers.

**By 1973, Thompson and Ritchie rewrote the UNIX operating system in C for portability and speed.** Around 1974, UNIX was licensed to universities for educational purpose and a few years later, was made commercially available. It is interesting to note that MS–DOS was created much later than UNIX, by which time the industry had begun to accept UNIX as the Standard Operating System. UNIX features have influenced the design of MS–DOS.

### Why UNIX

UNIX is much more like a spoken language, with commands acting as verbs, command options acting as adjectives, and the more complex commands acting similar to sentences. How you do a specific task can, therefore, be completely different from how your UNIX–expert friend does the same task.

## 1.5 Linux

The OSS movement, begin with a commitment to make software and the underlying source code freely available to all. An early advocate for the open source movement was the Free Software Foundation, a group created to found the GNU project, which was created to develop a **UNIX-like OS**.

**In 1991 Linus Torvalds creates open source, UNIX-like kernel released under the GPL and ports some GNU utilities. Today's Linux Kernel + GNU utilities give the complete, open source, UNIX-like OS.**

### Why Linux

1. Fresh implementation of UNIX APIs
2. Open Source development model
3. Supports wide variety of hardware, many networking protocols and configurations.

## 1.6 Characteristics of the Linux

1. Linux was written in C, so it is a case–sensitive OS
2. Interactive, it has a consistently good user interface.
3. Based on the philosophy of primitives to build complex programs.
4. Easy to implement and maintain. Hierarchical file system.
5. Easy application development. Consistent format for byte stream files.
6. Simple and consistent interface to peripheral devices, compatible file devices and inter–process I/O
7. Easier to write portable programs– hides machine architecture from the user.
8. Supports any language that has an interpreter or compiler.

**In general, like any OS, Linux provides the following two classes of services:**

**User services:** shell, mail, text processing, source code control system (SCCS) etc.

**System calls:** Operating system services for application development. System calls provide the interface to a running program and the operating system. User programs receive operating system services through the set of system calls.

## 1.7 Features and Benefits of Linux System

**Free:** Linux is available for free on the Internet by number of distributions.

**Open Source:** Open source means we get not only the executables but also the source code. User can access the source code and make improvements with it.

**Well Documented:** They are written by common people with simple language, to provide online documentation and help.

**Customizable:** Linux users have the option for using/installing new software, which is either commercial or free software available on the net.

**Hardware Support:** Linux can run on almost any hardware, like 386, 486, Pentium II, Pentium III, Pentium IV, SPARC, Dec Alpha, AMD ATHLON, Motorola 68000 series.

**Machine–independence:** The system hides the machine architecture from the user, making it easier to write applications that can run on micros, minis and mainframes.

**Multi–Tasking:** It allows you to run more than one job/program/task at a time. A program under execution is called a process.

**Multi–User:** A UNIX/Linux system is capable of supporting more than 1000 simultaneous users, each concurrently running a different set of programs. A multi user system allows many people to use the system resources almost simultaneously.

**TCP/IP Networking:** TCP/IP networking support is built into the 'kernel' itself. Linux is among the best operating systems in terms of networking. It includes programs like 'telnet', 'ftp', 'rlogin' and other such programs.

**High Level Security:** One of the main advantages of Linux is that it provides a very high level of security by using user authentication. It also stores passwords in an encrypted form. The password once encrypted cannot be decrypted. Linux also includes the file system security that enhances the existing security.

**Programming Support:** It supports a variety of programming languages C, FORTRAN, BASIC, PASCAL, ADA, COBOL, C++, JAVA, Lisp, Prolog, and many other languages.

**GUI:** It has a very good Graphical User Interface (GUI) with GNOME/KDE.

**Reliable:** It has been seen that Linux servers are very reliable – they have been up for hundreds of days compared to the regular reboots seen in the case of other OS.

**Portable:** The system is written in a high–level language making it easier to read, understand, change, and therefore, move to another machine. The code can be changed and compiled on a new machine.

**Time–sharing:** Since there is only a single CPU to take care of the various programs to be executed, the programs are queued and CPU time is shared among them. Each program is attended for a specific period of time, and then put back on the queue.

**Support for Utilities:** The UNIX system includes a family of several hundreds of utility programs. For example: **sort**. The sort utility can put the lists in alphabetical or numerical order, order by part of the number etc. Other utilities allow you to display, print, copy, search and delete files, **man utility provides online documentation of the UNIX system**.

**Web Server:** Linux can be used to run a web server such as "Apache Tomcat" to server application protocols such as HTTP or FTP.

**Linux Server provides following number of services:**

- A DHCP (Dynamic host configuration Protocol) Server
- A Web Server
- An FTP Serer
- An NFS Server
- An NTP (Network Time Protocol)
- A send-mail server
- An NIS (Network Information Systems) server

## 1.8 Kernel

The Kernel is the core (or heart) of the Operating System. It is the collection of programs mostly written in C that directly communicates with the hardware. There is only one kernel for any Operating System. It loads into RAM when a machine boots, where it runs until it is shutdown. It receives request (called System calls) from programs and initiates processes that carry out these requests. In addition it performs many low–level and system–level functions, to interpret and send instructions to the system's hardware, to schedule and run processes, and to manage input and output. The scheduling of various programs running in memory or allocation of CPU to all running programs also lies with the kernel. The kernel program is usually stored in a file called **/vmlinuz** in Linux. Where as the shell program is usually stored in a file called **/bin/bash** in Linux.

### Functions of the Kernel

1. System initialization: detects hardware resources and boots up the system.
2. Process scheduling: determines when processes should run and for how long.
3. Memory management: allocates memory on behalf of running processes
4. Security: Constantly verifies file system permissions and firewall rules.
5. Provides buffers and caches to speed up hardware access.
6. Receives user commands from the shell and communicates with the hardware.

Most of the OS is not Linux but rather a collection of applications that make use of the facilities provided by the Linux kernel. In particular, the Free Software Foundation's GNU project provides a great deal of utilities and core libraries that have made Linux based operating systems possible. The kernel documentation can be found under **/usr/share/doc/kernel-doc-*/Documentation**.

## 1.9 Shell

The Shell is the interface between the user and the kernel. It examines and evaluates commands typed at its prompt, then passes them to the kernel. It then receives the results back from the kernel and displays the results at terminal. Like any high–level language, the shell provides variables, flow control constructs, arrays and functions. Shell offer features geared especially for interactive use rather than to augment Programming Language. These interactive features include job control, command line editing, history and alias.

## Shell as a command Interpreter

Shell is the command interpreter. It is an intermediary program, which interprets the commands that are typed at the terminal, and translates them into commands that the kernel understands. The shell thus acts as a blanket around the kernel and eliminates the need for the programmer to communicate directly with the kernel.

A unique feature of the Linux operating system is that all Linux commands exist as utility programs. These programs are located in individual files in one of the system directories, such as /bin, /etc or /usr/bin. The shell can be considered as a master utility program, which enables a user to gain access to all other utilities and resources of the computer.

**The mechanism of command interpretation by the shell is as follows:**

1) On logging in command mode, the shell displays a *$* or # sign.
2) On logging in GUI mode, the shell displays a **kln@linux:~>** prompt.
3) The user now issues a command,

**For Example**

kln@linux:~> **ls mydir**

## 1.10 UNIX / Linux Architecture

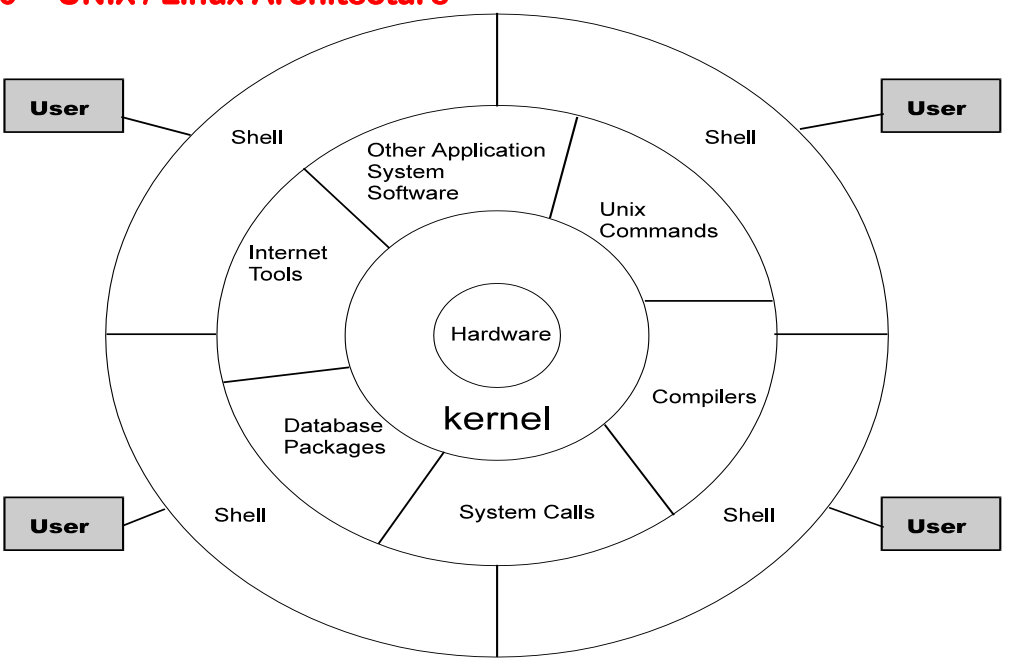



***Fig: Structure of the Unix/Linux System***

■ 2 ■

# File System

Many applications require that information be written to or read from storage device. Such information is stored in the device in the form of file. All utilities, applications data in Linux is stored as files. **Even a directory is treated as a file, which contain several other files.**

## 2.1 The hierarchical File structure

Like the family tree it resembles, the UNIX system file structure is also a hierarchical tree.

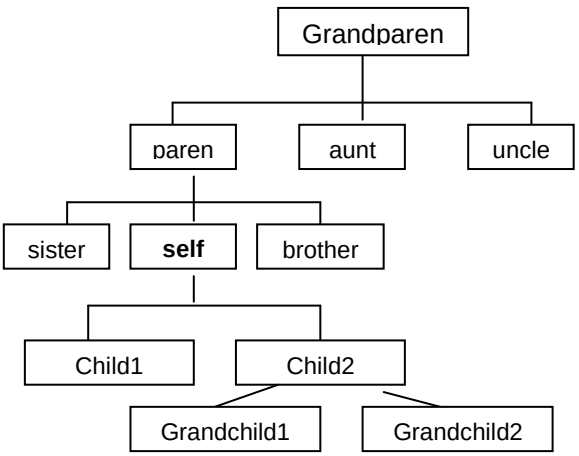



***Fig 2.1: Family Tree***

From this single directory, you can make as many subdirectories as you like, dividing subdirectories into additional subdirectories. In this manner, you can continue expanding the structure to any level according to your needs.

## 2.2 File Types

All files in Linux have a byte stream format. A byte stream is just a sequence of bytes. Directories are also treated as files. Treating everything as a file allows Linux to organize and exchange data more easily. Files are a collection of data items stored on disk. The interface to a device such as the screen or keyboard is designated as a file. A directory file contains information about a directory, organized in a special directory format. There are 3 types of files:

1) An **ordinary file** contains data, text or program instructions.
2) A **directory file** is special file that stores any kind of files.
3) A **special file** is related to various input and output (I/O) devices– printers, terminals, tape drives, hard drives, CD**–**ROM drives, modems, and Ethernet Adapters can be connected to the Linux system and carries the special meaning to Linux.

## Hidden Files

A filename beginning with a "**.**" is called an invisible (hidden) filename because it is not displayed by **ls**. Use **ls –a** to see all files, including hidden files. Two special invisible entries, a single and double period (**. and ..**), appear in every directory, representing the current (present) directory and the parent directory, respectively.

## Rules for naming files

1. Linux filenames are case sensitive, kln.txt and **Kln.txt** are two different files.
2. Filenames are generally composed of lower or uppercase letters or numbers.
3. Some limited punctuation marks, such as **.** (dot) or – (hyphen) or _ (underscore) are allowed. But don't use the hyphen(–) as the first character of a filename.
4. A filename is from 1 to 14 characters, some systems allow up to 255 characters.
5. Don't use the forward slash (/) in file-names, the only exception for root directory, which is always named and referred to by this single character **/**.
6. File names and directory names should not have spaces in them, if you want to give spaces between them then put them within double quotes. For example,

   kln@linux:~> **mkdir "Nist College" ; vi "Lakshmi Narayana"**
7. Don't use the illegal characters *, ?, <, >, &, $, #, @, or vertical bar(|) in a filename. These symbols represent different things to the command interpreter.
8. Like children of one parent, no two files in the same directory can have the same name. Files in different directories, like children of different parents, can have the same name.
9. A suffix is the portion of the filename after the period. Linux does not require the use of suffixes, also called extensions. **File extensions can be useful for categorizing files.** A filename extension is the part of the filename following a period.

## Commonly used suffixes to Filenames

| | |
|---|---|
| **.avi** | Windows AVI video file |
| **.bak** | Backup file |
| **.c, .cpp, .java** | C, C++ and JAVA source codes |
| **.cf, .cfg, .cong, .confg** | Configuration file |
| **.h** | C Programming header file |
| **.htm, .html** | Web pages |
| **.log** | Log file |
| **.mpg, .mpeg** | MPEG video file |
| **.o** | Object code |
| **.pdf** | Portable Document Format (Adobe Acrobat) file |
| **.tmp** | Temporary file |
| **.txt** | Text file |
| **.wav** | Windows WAV sound file |

In most cases, however, filename extensions are optional. Use extensions freely to make filenames easy to understand.

## Command Editing Tricks

The short-cut keys to command editing are:

| | | | |
|---|---|---|---|
| **Ctrl–a** | Moves to beginning of line. | **Ctrl–arrow** | Moves left or right by word. |
| **Ctrl–e** | Moves to end of line. | **Ctrl–shift–c** | Copies selected text. |
| **Ctrl–u** | Deletes to beginning of line. | **Ctrl–shift–v** | Pastes text to the prompt. |
| **Ctrl–k** | Deletes to end of line. | | |

## The Tab Key

Type the Tab to complete command lines. For the command name, it will complete a command name, for an argument, it will complete a file name.

**Examples:** kln@linux:~> **xte<TAB>**

Xterm

## 2.3 File Structure in Linux

The file system resembles an upside down tree. The main directory, called the root directory, is denoted with a simple slash (/). All directories on a Linux system are subdirectories of the root. Branching from the root there are several other directories called sub directories. The directory itself may contain any other sub directories and files.

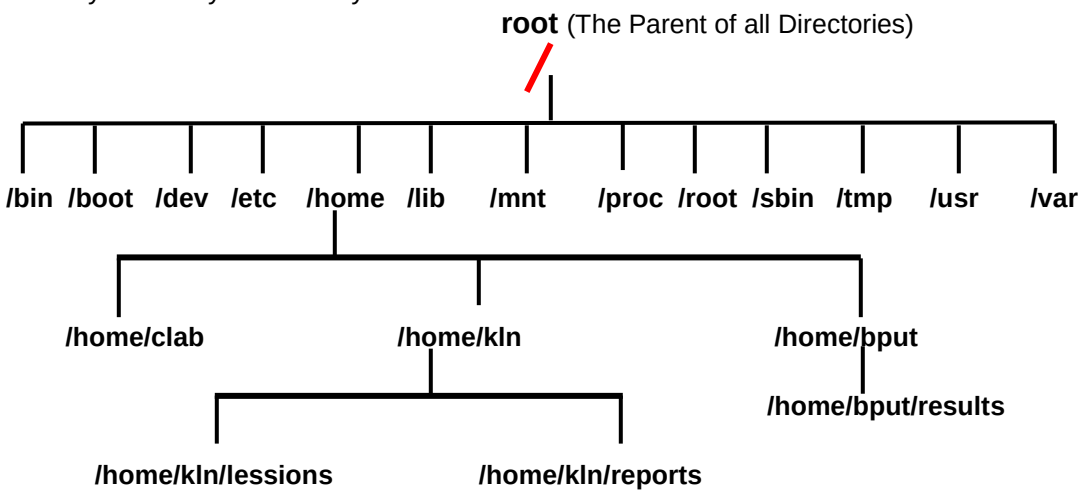



***Fig 2.2: Directory Structure***

## System Directories

The root directory that begins Linux file structure contains several system directories. The system directories contain files and programs used to run and maintain the system. Many contain other subdirectories with programs for executing specific features of Linux. For example, the directory /usr/bin contains various Linux commands that users execute, are cp and mv.. The detailed list of the system directories are shown in the below table.

| Name | Description |
|---|---|
| **/ (root)** | The / directory is parent of all file structures it lies on top of directory hierarchy. |
| **/home** | Contains each user's or clients home directory ( **/home/kln** or /home/clab ). |

| | |
|---|---|
| **/bin** | The common location for executable and binary files for Linux Command and utilities. It contains most commonly used commands. When a program is the result of a compilation, it's in binary form and is called an executable binary. |
| **/boot** | Static files of the boot loader. |
| **/dev** | This contains sub–directories for all special files, such as keyboard, terminal, mouse and other devices. |
| **/etc** | Machine Local System Configuration– Holds administrative, config and other system files. Holds /etc/passwd file that contains a list of all users who have permission to the System. It contains all System administration files. |
| **/lib** | It is the central storage for files that are commonly used by other programs. |
| **/mnt** | Mount point for temporary partitions. (Example pen-drive and CD-ROM). |
| **/proc** | Kernel and process information (Virtual File System). Under /proc, you will find information about running processes, as well as system configuration information such as interrupt usage, I/O port use, and CPU type. |
| **/root** | Home directory for root user (i.e. System Administrator). |
| **/sbin** | Essential System binaries– Contains utilities for system administration. |
| **/tmp** | Programs usually need extra space to store data on disk. This directory is such a storage place, it contains all temporary files. |
| **/usr** | The location for subdirectories for local programs and executables for user and administrative commands. It contains all user accounts and some commands. |
| **/var** | Variable data– Contains files whose contents change as system runs. The location for spooling, logging and other data. |
| **/opt** | The location for optional packages like GNOME and KDE. |

## 2.4 Directory Symbols and names

| Symbol | Description |
|---|---|
| **.** | The current or working directory |
| **. .** | The parent directory |
| **~** | The home directory of the current user |
| **/** | The root directory (not /root) |
| **. ./directory** | Refers to another directory of the same path |
| **/home/kln/lesions** | The absolute path name. |
| **~/.bash_profile** | Means the .bash_profile in my home directory. |
| **~student/.bash_profile** | Means the .bash_profile in student's home directory. |

■ 3 ■

# Linux Commands

## 3.1 How to Run Commands

When you're typing a command, nothing will happen until you hit the "return" key.

**Format:** Command–name [ options] [Filename/arguments]

Each item is separated by a space, and options are start with hyphen (–)

For example, the "ls" command has options "–l" (long form listing) and "–t" (sort in order of creation time). You normally put options before any other arguments.

kln@linux:~> **`ls -l *.c`**

There are two different ways of specifying more than one option. Either you can specify the options separately, e.g. "**–l –t**", or you can combine them into a single option "**–lt**".

## 3.2 Types of Commands

The Shell recognizes three types of commands. External, Internal and Shell Scripts.

**External Commands:** The shell creates a process for each of these commands that it executes, while remaining their parent. For example: **cat, who, ls** etc.

**Internal/Built-in Commands:** A command that is implemented internally by the shell itself, rather than by an executable program somewhere in the file system. Some of them like **cd** and **echo** don't generate a process, and are executed directly by the shell.

**A partial list of built–in commands:**

| | | | | |
|---|---|---|---|---|
| **:** null builtin | alias | bg | bind | break |
| **.** executes program in the current process. | continue | declare | echo | eval |
| **dir** lists the directory stack | exit | export | fg | help |
| **getopts** pass arguments to a shell script. | jobs | kill | let | pwd |
| **hash** remember location of commands in search path. | readonly | return | set | shift |
| **time** displays times for current shell and its children. | umask | unalias | unset | wait |
| **cd** changes the directory. | exec | history | read | test |

**Shell Commands**

The shell executes scripts by spawning another shell, which executes the commands listed in the script. When you are execute a command the shell normally creates a new process and waits for it to finish. Other commands which are running in the background may be run without waiting for it to finish. **All the commands run through terminal or console.**

**Format:** **Command–name [options] [file–list]**

| | |
|---|---|
| **Ctrl +c** | Cancel the Command |
| **Ctrl +z** | Suspends the current process. |

## 3.3 The bc Infix Calculator

The UNIX/Linux infix–notation calculator. It reads from the standard input. To use the bc calculator, enter the following command:

kln@linux:~> **bc**

The bc, waits for user input. When you invoke the bc without arguments, the input has to be keyed in, each line terminated by pressing <Enter–Key>. After you have finished your work, use <**ctrl d**> or **quit** to exit from bc.

### Numbers

The most basic element in **bc** is the number. The numbers may have both the integer part and fractional part. The two attributes of numbers are **length** and **scale**. The **length** is the total number of significant decimal digits in a number and the **scale** is the total number of decimal digits after the decimal point.

**.2912** Has the length of 4 and scale of 4
**29.12** Has the length of 4 and scale of 2

### Variables

Numbers are stored in two types of variables, simple variables and arrays (All array variable names will be followed by square brackets "**[ ]**"). Variable–name begins with an alphabet followed by any number of alphabets, digits, and underscores. **All alphabets must be lowercase**.

### Expressions

The bc command comes quite handy when converting numbers from one base to the other. The default for both input and output is base 10. Before converting a numbers from one base to other, you have to set the **ibase (input base)** or/and **obase (output base)**. The legal values of **ibase** are 2 through 16. Input numbers may contain the characters 0–9 and A–F.
**Note:** The expressions must be **capitals**. Lowercase letters are used for variables.

| Notation | Description of Function |
|---|---|
| **var=expr** | The variable is assigned the value of the expression. |
| **+ , –, *, /, %** | For add, subtract, multiply, divide and remainder. |
| **Expr ^ expr** | The result of the expression is the value of the first raised to the second. The second expr must be integer. **2 ^ 4** results to **16**. |
| **sqrt(expr)** | Square root of expr. Where expr is integer number. |
| **l(expr)** | The result is the logarithm of expr. |

```
kln@linux:~>  bc
(37*1.334)+44
93.358
2^32
4294967296              # The maximum memory possible in a 32–bit machine.
9/5
1                       # Decimal portion truncated.
```

**By default, bc performs truncated division, and you have to set scale to the number of digits of precision before you perform any division.**

```
scale=2
9/5                    # Truncate to 2 decimal places
1.80
quit                   # To exit from the bc.

kln@linux:~>  bc
obase=2
10.5
1010.1000              # Convert decimal to binary.
15.25
obase=10
ibase=2
10000
16                     # Convert binary to decimal.
quit
kln@linux:~>  bc
obase=16
ibase=2
10101100
AC                     # Convert binary to Hexadecimal.
101101100000
B60
obase=2
111+1010               # Binary Arithmetic Addition.
10001
111–1010               # Binary Arithmetic Subtraction.
–11
quit
```

The **bc** also used with variables that retain their values until you exit the program. Use semicolon (;) for command line separator.

```
kln@linux:~>  bc
x=3;  y=4; z=5
p=x + y + z
p
12
++x
4
y+=6            #  Shorthand Assignment Operator.
```

If you wanted to calculate the sine of 4.5243 to the third power:

```
kln@linux:~>  bc –l math
s ( 4.5243 ^ 3 )
–.99770433540886100879
quit
```

## The man command

The man command display the manual page for the command specified. The **man** displays text–only manual pages using the less text reader. **Example:** **man ls**

■ 4 ■

# Managing File System

## 4.1 Different levels of Directories

When you log into the system, you are placed within your home directory. The name given to this directory by the system is the same as your login name. You can create any number of directories and files within your home directory.

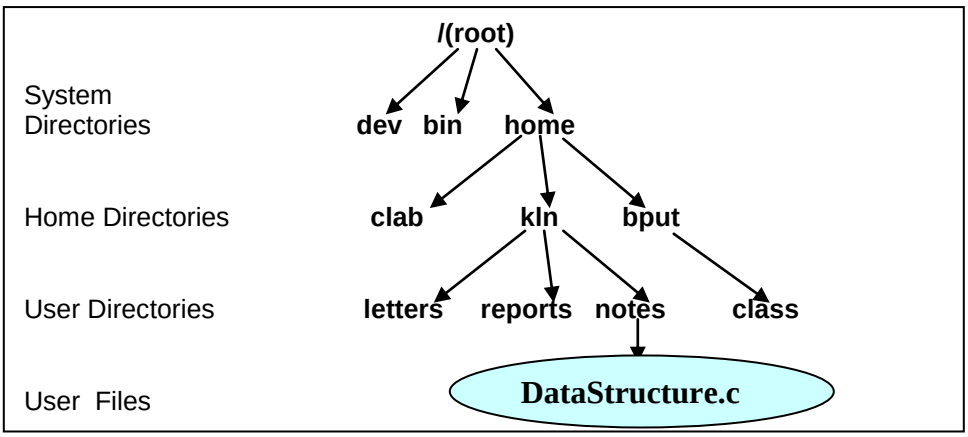



***Fig 4.1: Different Levels of Directories***

You can access a directory either through its name or by making it the current directory. Each directory is given a name when it is created. You can use this name in file operations to access files in that directory. You can also make the directory your default directory. If you do not use any directory names in a file operation, then the default directory will be accessed. The default directory is referred to as the *working directory*.

## 4.2 Creating and Removing of Directories

You can **create** new directories with **mkdir**, under your home directory or it's below levels.

**Format:**

| | |
|---|---|
| **rmdir directory–list** | It is used to remove the empty directory only. |
| **rm –r directory–list** | It is used to remove the contents of directory recursively, i.e. it will remove the directory and all it's sub–directories and files also. |

The **directory–list** contains one or more pathnames of directories. You must have write and execute access permission to the parent directory of the directory you are creating. Following command creates a directory notes, and letters as a subdirectories of the working directory.

```
kln@linux:~> mkdir notes
kln@linux:~> mkdir /home/kln/letters
```

The mkdir command takes one or more arguments and creates the directories as specified.

kln@linux:~> **mkdir iit iit/Orissa**
Creates a directory iit under it another directory Orissa, both in one operation.

kln@linux:~> **mkdir –p BPUT/{NIST, Apex}**
Creates a directory BPUT under it another two directories NIST, and Apex in one operation.
The **rmdir, rm –r** commands removes directories from the file system by removing links of those directory-lists.

kln@linux:~> **rmdir letters**
kln@linux:~> **rm –r /home/kln/notes**

**Home Directory( ~ )**

You can use the ~ to represent the absolute path name of the home directory.

| Command | Purpose |
|---|---|
| **cd** | To go to Home directory i.e. /home/kln |
| **cd ~** | To go to Home directory i.e. /home/kln |

## 4.3 Listing of files/directories

The ls command will display the list of all files (including the directories), which are present under the current working directory.
**Format:** **ls [options] [file–list]**

The **options** determine the order in which the files will be displayed. The **ls** command lists the files alphabetically, containing the names of files and directories.

To distinguish between different file names, you need to use the **ls –F** command. Display the file type with a character appended to the filename output.

| Symbol | File type |
|---|---|
| * | Executable File |
| / | Directory |
| = | Socket |
| \| | Named pipe FIFO |
| @ | Symbolic link |

kln@linux:~> **ls**
profile.txt Reports Letters Notes file1.txt a.out

kln@linux:~> **ls bput**
The above command will display the list of all files ( including the directories ), which are present under the **bput** directory.

kln@linux:~> **ls -F**
profile.txt file1.txt Reports/ Letters/ Notes/ a.out*

**Listing Directories with options**

| | |
|---|---|
| **ls –a** | Displays the list of all files including hidden files, which are starting with dot(.). |
| **ls –r** | List the entities alphabetically in reverse order. |
| **ls –R** | Displays all nested subdirectories and files below the working directory. |
| **ls –s** | Print the size of each file in blocks. |
| **ls –1** | List one file per line alphabetically. |

kln@linux:~> **ls -R /root**
kln@linux:~> **tree /root**
The above command will display the tree structure of the directory contents.

## 4.4 Change Directory (cd or chdir)

When you log in, you are always placed in your home directory. You can move form one working directory to another directory through **cd** command.

kln@linux:~> **cd [directory]**
The directory is the pathname of the directory that you want to change as the current working directory. If you are not specifying any directory name, the **cd** command takes you to the user's home directory.

**For example:**
kln@linux:~> ***mkdir*** **notes**
kln@linux:~> **cd notes**
kln@linux:~> **pwd**
/home/kln/notes (It Displays the present working directory)

You can also change to another directory by using its full path name.
kln@linux:~> **cd /home/kln/letters**

The cd command can be used to progress through a series of nested directories.

| | |
|---|---|
| **cd** | Switches to user's home directory |
| **cd ~** | Switches to user's home directory |
| **cd ..** | Switches to the parent directory (the next directory up) |
| **cd /** | To go to topmost directory i.e. root ( / ) directory. |

## 4.5 Creating a File

You can create the files by using one any of the editors. For example through vi editor:

**Format:** **vi filename**
The following command creates a file **DataStructure.c** in the **notes** directory.
kln@linux:~> **vi /home/kln/notes/DataStructure.c**

To create the file profile.txt under the home directory the command is:
kln@linux:~> **cd ~**
kln@linux:~> **vi profile.txt**

## 4.6 Copying Files

To copying files, **cp** command will take two arguments. The first argument is the source file (existing file) and the second argument is the destination file/directory.

**Format:** **cp [options] source–file destination–file**
**cp [options] source–file–list destination–directory**

| | | | |
|---|---|---|---|
| **cp –i** | Prompt before overwrite. | **cp –R** | Copy directories recursively. |
| **cp –v** | Copy with verbose. | **cp –a** | Archive, it is used to copy directories. |

The cp has two modes of operation. The first copies the contents of one file into another file. The second copies one or more files to a directory. The **source–file** is the pathname of the file that **cp** is going to copy. The **destination–file** is the pathname that **cp** will assign to the resulting copy of the file. If the destination–file exists before you execute **cp**, **cp overwrites the file**, destroying the contents, if the destination–file doesn't exists, it will create a new file, and then copy the source-file contents into destination–file.

**For example,** the user copies a file called **kln.txt** to a new file called **newkln.txt**.

```
kln@linux:~>  cp  kln.txt  newkln.txt
```

If the newkln.txt file already exists the above **cp** command overwrites that file. It may be safer to use the cp command with the –i option to detect the overwrite condition. With this option cp will first check to see if the file already exists. If it does, you will be then asked if you wish to overwrite the existing file. If you enter **y**, the existing file will be destroyed and a new one is created as the copy. If you enter anything else, it will be taken as a negative answer, and the cp command will be interrupted, preserving the original destination-file.

```
kln@linux:~>  cp  –i  oldfile  newfile
```

The following command copies all the **.c** files and **.cpp** files into the **notes** sub–directory.

```
kln@linux:~>  cp  *.c  *.cpp  notes
```

## 4.7 Moving or Renaming of Files through mv

The **mv** moves, (or renames) one or more files. It has two formats. You can use the mv command either to **rename a file** or **move a file** from one directory to another.

**Format:** **mv existing–file new–file–name**
**mv existing–file–list directory–name**

| | | | | | |
|---|---|---|---|---|---|
| **mv –f** | Don't prompt. | **mv –i** | Prompt before overwrite. | **mv –v** | Move with verbose |

In the first form of **mv**, the **existing–file** is a pathname that specifies the plain file that you want to rename. The **new–file–name** is the new pathname of the file.

The UNIX/Linux system implements **mv** as **ln** and **rm**. When you execute the **mv** utility, it first makes a link(**ln**) to the new–file and then deletes (**rm**) the **existing–file**. If the **new–file–name** is already exists, **mv** deletes it before creating the link.

In the second form, the **existing–file–list** contains the pathnames of the files that you want to rename, while the **directory** specifies the new parent directory of the files. The files that you rename will have the same names as the simple filenames of each of the files in the **existing–file–list**.

If the **existing–file** and the **new–file** or **directory** is on different physical devices, it implements **mv** as **cp** and **rm**. In this case **mv** actually moves the file instead of just renaming it. After a file is moved, the user who moved the file becomes the owner of the file.

When using mv to **rename a file**, you simply use the new file name as the second argument. The first argument is the current name of the file that you are renaming.

kln@linux:~> **`mv profile.txt resume.txt`**

You can move a file from one directory to another by using the directory name as the second argument in the mv command.

kln@linux:~> **`mv *.c /home/kln/notes`**

## 4.9 Deleting or removing Files (rm)

The rm command can take any number of arguments, allowing you to list several file names and erase them all at the same time.

**Format:** **rm [options] file–list**

| | |
|---|---|
| **rm –f** | Never prompt before removing. |
| **rm –i** | Prompt before any removal. |
| **rm –v** | Remove with verbose, i.e. it will display the message what is being done. |
| **rm –r** | Remove the directory recursively, it will remove the directory as well as its subdirectories and files. |

The **rm** removes links to one or more files. When you remove the last link, you can no longer access the file and the system releases the space the file occupied on the disk for use by another file (i.e., the file is deleted). To delete a file, you must have execute and write access permissions to the parent directory of the file itself. Be careful when using the rm command. Once a file is removed, it cannot be restored. You can use the rm commands with –i option to confirm that you want to erase a file or not.

kln@linux:~> **`rm /home/lakshmi/DataStructure.c`**

# ■ 5 ■

# File Display and Print

## 5.1 more, less, head, and tail commands

A file may contain more number of pages, so it is not possible to display all the pages at a time. If you are trying to view those kinds of files, all pages will scroll, and it will display only last page on screen. These commands are helpful when you want to view parts of text files.

| Format | Purpose |
|---|---|
| **more filename**<br><br>**For Example:**<br>`more profile.txt` | The more command enables you to move forward only. Pressing the **spacebar** move forward text by one screen–full, and by pressing **Enter–key** it will advance to one line at a time. Press DEL or q to terminate from the more command. |
| **less filename** | The less command enables you to move backward and forward while viewing a file. Pressing the **spacebar (or Page–Down)** move forward by one screen–full, and by pressing **Enter–key(or Down– arrow)** move forward one line at a time. By pressing **Page–Up** move backward one screen–full, and by pressing **Up–arrow** move backward one line at a time. Press q to quit from less command. |
| **head filename** | The head command let you slice lines from the front(head) of a text file. It prints the first 10 lines of a file by default. |
| **head –15 filename** | To print more than 10 lines, use –n option. Here n may be any number. This command will display the 15 lines. |
| **head –lines 20 filename** | This command will display the 20 lines. |
| **head file1 file2 file3** | You can print the head of more than one file by specifying all the filenames at the prompt. |
| **tail ±number [options] [file]** | The tail command displays the last part of a file. The **tail** prints the last 10 lines of a file by default. It takes its input from the file specified on the command line or from the standard input. If the number is preceded by a plus sign, **tail** counts the **number** of units specified by the number and any option., form the beginning of the file. If the number is preceded by hyphen(–), **tail** counts from the end of the file. |
| **tail –3 memo** | Displays the last three lines of the file. |
| **tail +5 memo** | Displays the file, starting at line eight (+8, no option). |

## 5.2 cat

Displays the contents of the specified filename on the standard output.

**Format:** **cat [option] [filename]**

| Option | Purpose |
|---|---|
| **–n** | Number all output lines, even blank ones. |
| **–s** | Show no more than one blank line. |
| **–b** | Number the output lines unless they are blank. |

**For Example:**

**cat program1.c**

Displays the program1.c contents on the standard output.

kln@linux:~> **cat file1 > file2**

The above command causes the output of the cat command file1 goes to a file called file2 instead of the standard output, and it causes the overwritten when the file2 is already exists.

kln@linux:~> **cat file1 >> file2**

The above command ensures that file2 is not overwritten, but contents of file1 are added after the contents of file2.

kln@linux:~> **cat file1 file2 > file3**

Above command overwrites the contents of the file3 with file1 and file2, if the file3 does not exist then it creates a new file file3, and write the contents of the file1 and file2, to the file3.

kln@linux:~> **cat file1 file2 >> file3**

The above command appends the contents of file1 and file2, to file3, if the file3 does not exist then it creates a new file file3, and appends the contents of file1 and file2, to the file3.

kln@linux:~> **cat > foo**
kln@linux:~> **cat – > foo**

In the above two commands, input is from the keyboard, while the output is saved in foo.
When the input is taken from multiple sources, a file, as well as the standard input, the **–** symbol must be used to indicate the sequence of taking the input.

kln@linux:~> **cat file1 – file2 > newfile**

Here we are putting the contents of file1, standard input contents and file2 contents in newfile.

# ■ 6 ■

# Filters and Pipes

If you've ever learned a foreign language, you know that the most common approach is to start by building your vocabulary (almost always including the names of the months, for some reason), and then you learn about sentence construction rules. The UNIX command line is a lot like a language. Now you've learned a lot of UNIX words, so it's time to learn how to put them together as sentences using file redirection, filters, and pipes.

Redirection changes the assignments for standard input and standard output. The *redirection*, uses the special commands in UNIX that instructs the computer to read from a file, write to a file, or even append information to an existing file.

Each of these acts can be accomplished by placing a file–redirection command in a regular command line:

**1) < redirects input,**
**2) > redirects output, and**
**3) >> redirects output and appends the information to the existing file.**

## 6.1 Output Redirection

When output from a process is redirected to sources other than the standard output, it is called output Redirection.

kln@linux:~> **`ls > dlist.txt`**

kln@linux:~> **`date > dob.txt`**

kln@linux:~> **`uname > systeminfo.txt`**

Your can create the new file by typing the command:

kln@linux:~> **`cat > newfile.txt`**

Then, whatever you typed at the keyboard was written to a file newfile.txt until you typed ctrl+d.

The ctrl+d signals an end–of–file to the shell, which in turn saves and close the file.

| | |
|---|---|
| **`cat >> newfile.txt`** | The entered text will be appended to the file newfile.txt |
| **`cat newfile.txt`** | It will display the contents of the newfile.txt |
| **`cat f1 f2 > file3`** | Which concatenates f1 and f2 and replaces them into file3. |
| **`cat f1 f2 >> file3`** | Which concatenates f1 and f2 and appends them to file3. |

**Output Redirection using file Descriptors**

To redirect the standard error output of a file stack.c to a file named stackerror.txt , use the following command:

```
kln@linux:~> cc  stack.c  2>stackerror.txt
```

All the error messages of the c program stack.c that would have gone to the terminal will now go to the stackerror.txt file.

**Note that there are no spaces between 2, >, and stackerror.txt string.**

```
kln@linux:~> cc  prime.c

kln@linux:~> ./a.out  1>primenumbers.txt
```

In the above command output of the file will go to the primenumbers.txt

| | |
|---|---|
| **2> file** | # direct stderr to **file** |
| **> file 2>&1** | # direct both stdout and stderr to **file** |
| **>> file 2>&1** | # append both stdout and stderr to **file** |
| **2>&1 \| command** | # pipe stdout and stderr to **command** |
| **command 1> out_file 2> err_file** | # redirect stdout and stderr to two separate files |

## 6.2 Input Redirection

We can redirect the input for a command using input redirection.

## sort

The **sort** utility sorts and/or merges one or more text files in sequence. This command takes its input from standard input and writes its output to standard output. The sort utility is used to sort the contents of a file. The command takes lines from one or more input files, sorts them and sends to standard output. It arranges lines in alphabetic or numeric order.

**Format:**

**sort [options] [field–specifier–list] [file–list]**

The **sort** utility uses the **sort–fields** to sort the lines.

The **field–specifier–list** selects one or more sort–fields within each line to be sorted.

The **file–list** contains pathnames of one or more plain files that contain the text to be sorted.

The **sort** sorts and merges the files unless you use the **–m** option, in which case, **sort** only merges the files.

**Example:**

| | |
|---|---|
| kln@linux:~> **cat days**<br>Monday<br>Tuesday<br>Wednesday<br>Thursday<br>Friday<br>Saturday<br>Sunday<br>Ctrl + d | kln@linux:~> **sort < days**<br>Friday<br>Monday<br>Saturday<br>Sunday<br>Thursday<br>Tuesday<br>Wednesday |

Passing the Long listing to sort command through pipe (|)

kln@linux:~> **ls –l | sort –n**

Enter your class student's names in the names.txt file, and marks in the file marks.txt according to roll number wise.

To list names of the students in Alphabetical order in the screen:
kln@linux:~> **sort < names.txt**

To list marks of the students, in ascending order:
kln@linux:~> **sort –n < marks.txt**

To save the alphabetical ordered names of the students in the file newnames:
kln@linux:~> **sort < names.txt > newnames**

To save the ascending ordered marks of the students in the file newmarks:
kln@linux:~> **sort < marks.txt > newmarks**

By default text files can be sorted on first fields, to sort other than the first fields.

**Format:**

**sort +1 filename**

Sort skips one field and starts starting on the second field.

**sort –t" " +1 filename**

## 6.3 Pipe

A *pipe* is a way to connect the output of one program to the input of another program directly. A *pipeline* is a connection of two or more programs through pipes. The vertical bar character | tells the shell to set up a pipe line: You can use a pipe to direct the standard output of one program into the standard input of another and avoid creating unnecessary temporary files. A pipe is frequently used with a member of the class of UNIX utility programs that accept input either from a file specified on the command line or from the standard input. The one of these utilities is **lpr** (printer).When you follow lpr with the name of a file, **lpr** places that file in the

printer queue. If you do not specify a filename on the command line, **lpr** expects input from its standard input. This allows you to use a pipe to redirect input to **lpr**. A pipeline is a sequence of simple commands separated by ' **|** '.

**Format:**

**command1 [ | command2 ]**

The output of each command in the pipeline is connected via a pipe to the input of the next command.

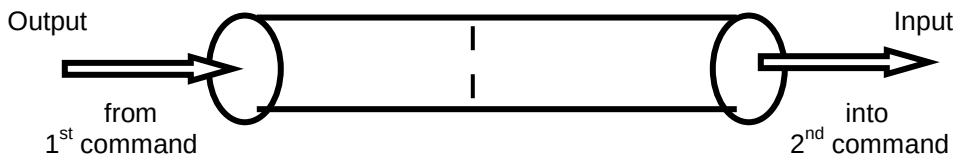



***Fig: the pipe data flow***

Suppose you want to count the size of all the files in your directory, and to sort the list numerically you can choose the one of the following commands:

kln@linux:~> **`ls –s > tempfile.txt`**
kln@linux:~> **`sort –n tempfile.txt`**
kln@linux:~> **`ls –s | sort –n`** (Sort cut command by using PIPE)

## 6.4 wc

Display the number of lines, words, and characters contained in one or more files. If you specify more than one file on the command line, **wc** display totals for each file and totals for the group of files.

**Format:**

**wc [options] file–list**

The **wc** takes its input from files specified on the command line or from the standard input.

| Option | Purpose |
|---|---|
| **–c (characters)** | This option causes **wc** to display only the number of characters in the file. |
| **–l (lines)** | This option causes **wc** to display only the number of lines in the file. |
| **–w (words)** | This option causes **wc** to display only the number of words in the file. |

The **file–list** contains the pathnames of one or more text files that **wc** analyzes.

The following command line displays an analysis of the file named **file1.txt**.

kln@linux:~> **`wc file1.txt`**

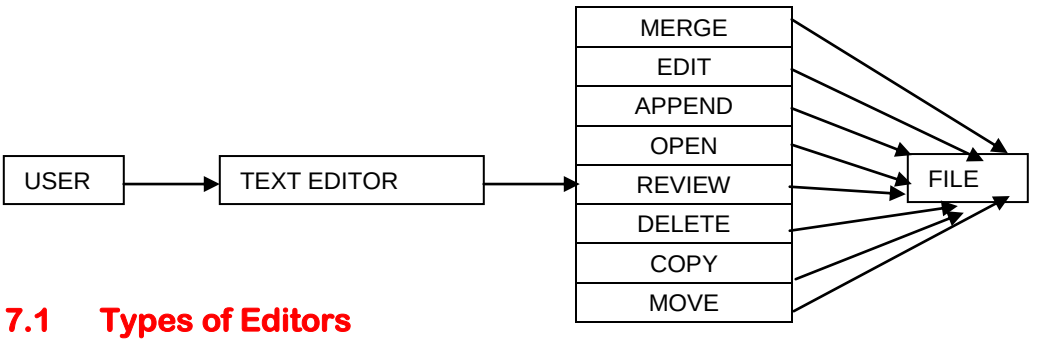
2 8 37 file1.txt

file1.txt has 2 lines, 8 words, and 37 characters in all.

■ 7 ■

# Editors

Text editors are software tools help the user to enter his file from key board in whatever form he wants, store in main memory and secondary memory, to perform the following tasks as shown in the below figure.




## 7.1 Types of Editors

There are several commercial text processing software packages for UNIX, but only the most common ones are below.

| | |
|---|---|
| **nano** | Easy to learn, easy to use. |
| **pico** | editor, full–screen. A simple menu driven text editor. |
| **vi** | editor, full–screen |
| **vim** | Advanced full–screen text editor (“vi–improved”) |
| **gvim** | A graphical version of the vim editor |
| **ed** | editor, line based |
| **gedit** | A simple graphical editor |
| **sed** | batch mode editor |
| **emacs** | editor, programmable full–screen |
| **awk** | pattern–matching processor |
| **joe** | Text–based editor, enables you to edit text on–screen directly. |

## 7.2 vi Visual Editor (The King of all Editors)

The **vi** makes editing easier by displaying a full screen of text at a time and allowing you to see what you’re working on. Because vi displays a full screen of the text you are editing, it is called a screen editor.

**Format:** **vi [option] [filename]**

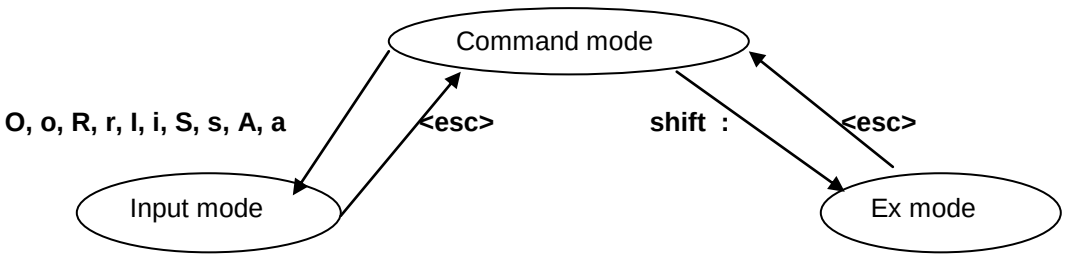
| Option | Purpose |
|---|---|
| **vi filename** | Edits filename starting at line 1. |
| **vi –R filename** | Display filenames in read only mode. |
| **vi –r filename** | **Recover a filename after a system crash.** |

A **vi** session begins by invoking the command vi with(or without) a filename.
*kln@linux:~>* **vi filename**

You are presented a full empty screen, each line beginning with a **~** (tilde). This is **vi**'s way of indicating that they are non existing lines. For text editing **vi** uses 24 of the 25 lines that are normally available in a terminal. The last line is reserved for some commands that you can enter to act on the text. This line is also used by the System to display messages.

## 7.3 Modes of vi Editor

There are three modes in which **vi** works. Before you attempt to enter text into the file, you need to change the default command mode to input mode. There are several methods for entering into this mode, but in every case the mode is terminated by pressing the **<esc>** key.




**Calling vi Editor to create / open a file**

Call vi with the following command line to create a file **hello.txt** in the working directory.
kln@linux:~> **vi hello.txt**

```
~
~
~
"hello.txt" [ New  File ]
```

**The last line of the vi editor gives the status information about the file.**

The terminal Screen will look similar to the one shown above, if the file doesn't exist, the tilde (~) symbols shown at the beginning of each line, to indicate that below that tilde symbols no text is there.

**Then Press character i** (or any character of the Input mode), **to ENTER the text**. When you pressing Input mode character the terminal widow's last line will appears as **— INSERT —**.

```
Hello,
I am KLN welcomes all B.Tech, M.Tech, M.C.A. and M.B.A. students for BPUT.
Thank you for registering under this University.
Wish You have a bright and beautiful feature.
~
-- INSERT --
```

**Then press esc–key followed by :w ( or :w! ) to Save the file.**

When you are pressing Esc–Key followed by :w the last line will appear like this:

**"hello.txt" 4L,230C Written**

Here 4 is the total number of lines, and 230 is the total number of characters in the file.
**Press esc–key followed by :wq ( or :wq! )to Save and Quit from the file.**
**Press esc–key followed by :q ( or :q! )to Quit without saving from the file.**

When you are pressing Esc–Key followed by :wq (or :q) you will go back to shell prompt.

kln@linux:~>

## 7.4 Deleting Text

You can also delete any text in your file with the delete command d. Like the change command the delete command requires a text object.

**Deleting Text in Command Mode**

| Command | Function |
|---|---|
| **dd** | Deletes the entire current line. |
| **6dd** | Deletes the current line and 5 lines below. |

**Deleting text in insert mode**

| Keyboard shortcut | Function |
|---|---|
| **Backspace-key** | Delete the previous character at the time of entering the text without any moment of cursor. |
| **Delete-key** | Delete the current character under the cursor. |

## 7.5 Yank(copy)

"yank" means to extract( or copy). When you yank objects (word, line, group of words or a group of lines) vi copies these objects into buffers without removing them from the file. These objects can then be pasted to another location in the file or even to different files.

| Command | Function |
|---|---|
| **yy or Y(uppercase)** | Yanks(copies) the whole line. |
| **5yy** | Yanks(copies) 5 lines from cursor position. |

## 7.6 Paste or Put commands

The put commands, p and P (upper case p) copy text from the General Purpose Buffer into the Work Buffer.

| Command | Function |
|---|---|
| **p** | Paste below the current line. |
| **P(uppercase)** | Paste above the current line. |

## 7.7 Recovery of Text

There are several ways to recover deleted text.

| Command | Function |
|---|---|
| **u** | Undo the most recent command. |
| **U(uppercase)** | Allow you to discard all the changes before you move away from the line.(Make sure the cursor has not been moved to another line before invoking the command U, otherwise it won't work). |
| **5u(lowercase)** | Undo the last 5 editing actions. |
| **Ctrl r** | To redo the last undone instruction. |
| **3 <ctrl r>** | It will redo the last 3 undone instructions. |

## 7.8 Joining Lines

When you merge two lines into one, position the cursor anywhere on the first line, and press J (uppercase) to join two lines. You could have joined three lines by using the command 3J.

| Command | Function |
|---|---|
| **J(uppercase j)** | It joins the current line, and the one line following it. |
| **4J** | It joins four lines (i.e. the current line, and the three lines following it). |

## 7.9 Setting Parameters (set)

You can adapt vi to your needs and habits by setting vi parameters. To set a parameter while you are using **vi**, enter a colon ( **:** ), the word **set**, a SPACE, and the parameter . The command appears on the status line as you type it and takes effect when you press Enter.

**Set commands**

Toggle options which are either on or off, and that take a numeric or string value.

| | |
|---|---|
| :set option | Activate Option. |
| :set nooption | Deactivate option. |

For example to specify that pattern searches should ignore case, type:

**:set ic**

If you want your **vi** to return to being case–sensitive in searches, give the command:

**:set noic**

**Some set commands which are most commonly used**

| Command | Function |
|---|---|
| **:set nu** | Set display of line numbers on. |
| **:set nonu** | Set display of line numbers off (default). |
| **:set autoindent** | Sets the vi to indent automatically. |
| **:set showmode** | Display mode in which we are working. It will display the type of input mode, such as INSERT,APPEND,REPLACE and so on. |

## 7.10 How to Recover the Un–Saved Files of vi editor

When you are entering a text in the file kln.txt using vi editor , suddenly power failures or any un–even things are happened (like closing the window without saving) so that you are not able to save the document which is typed through vi editor. To recover that matter which is typed open the file in this way.

```
kln@linux:~>     vi    –r   kln.txt
```

**Then you will find the following message:**

Using swap file ".kln.txt.swp"
Original file "~/kln.txt"
Recovery Completed. You should check if everything is OK.
Hit ENTER or typed Command to continue

**After that you press either ENTER–KEY or DOWN–ARROW continuously.**

How the UNIX/Linux system recovers the files which are not saved. Whatever may be you are typed in the file are saved in the hidden file named **.filename.swp** . Here, when you are not saving the file you can recover the contents of the file through the file **.kln.txt.swp** file.

When the next time you are not saving the file it will create the hidden **.kln.txt.swo** file, and it is the latest file. In this way it keeps on creating the files **.kln.txt.swn .kln.txt.swm .kln.txt.swl .kln.txt.swk ….. till .kln.txt.swa** . In this case the most recent file is stored in the file named **.kln.txt.swa** which has a serial number 1.

For the above file kln.txt if you are not saved the file for second time, then it creates the two files, the most recent file with the serial number 1 is stored in the **.kln.txt.swo** file.

**To recover the unsaved file kln.txt :**

```
kln@linux:~>     vi   –r   kln.txt
```

Then you will find the following message:

**Swap files found Using Specified name:**

1. .kln.txt.swo
2. .kln.txt.swp

After that you keep on pressing the DOWN–ARROW till you get the following message:

**Enter number of swap file to use ( 0 to quit )**

Then type 1 followed by ENTER–KEY, then press either ENTER–KEY or DOWN–ARROW continuously to get the latest recovered file. **Hint: Always press the 1 only.**