UNIT-1

Introduction to Unix-Brief History-What is Unix-Unix Components-Using Unix-Commands in Unix-Some Basic Commands-Command Substitution-Giving Multiple Commands.

INTRODUCTION TO UNIX

1.1 BRIEF HISTORY

Unix has a longer history than any other popular operating system. Though many schools have contributed to its development, the initial contributions by The Bell Laboratory of AT&T and the University of California, Berkley (UCB) are notable.

Bell Laboratory's contribution In 1965, Massachusetts Institute of Technology (MIT), General Electric, and The Bell Laboratories of AT&T worked on a joint venture project called Multics (Multiplexed Information & Computing System), which intended to develop a multi-user operating system. As the progress was not satisfactory, AT&T withdrew itself from the Multics project in early 1969.

On the basis of the ideas acquired while working on Multics, Ken Thompson, a researcher started working on a different project. He and Dennis Ritchie developed an operating system (OS), called UNICS (Uniplexed Information and Computing Systemuring the latter part of 1969. UNICS was developed completely in the assembly language of PDP-7 and so it was not portable.

To achieve portability, Thompson considered implementing the system in a higher level language. Ritchie developed a higher level language called C in 1973. Ritchie completely rewrote the entire UNIX system during the same year using C. Actually around 95% of this Unix system was written in C and the remaining was written in the assembly language. At the same time many researchers in AT&T showed interest in the Unix project (around 1970 UNICS became Unix). During those days many text-processing utilities along with a text editor called the *ed* editor and a simple command interpreter called the *shell* were developed. The ed editor was a *line* editor and the then developed shell became the Bourne shell (sh), the grandfather of almost all the currently available *shells.*

A system called Unix System V was announced in 1983. System V has since then undergone many revisions and releases. The most important of the releases is the System V release 4 (SVR4) in 1991. SVR4 brought all the important features of various operating systems like BSD, XENIX and SUN operating systems together that were available by then. During the early days of the development of UNIX. However, AT&T made the Unix system available to universities, commercial firms and defence laboratories either free of cost or at a nominal price.

UCB's contribution University of California at Berkeley (UCB) was one of the early universities that was interested in the Unix operating system and its development. The team at Berkeley was responsible for many important technical contributions as well as the development of useful utilities. For example, an editor called the ex editor and a Pascal compiler were developed during 1974 by Bill Joy and Chuck Haley, then graduate students at UCB. Later the ex editor, which was also a line editor, was provided with the screen-editing facilities and was called the vi editor. Another important contribution of Bill Joy was the C-Shell (csh). In general, researchers at Berkeley filled the gaps that existed in AT&T's Unix at that time with their contributions and released their own version of Unix, called BSD-Unix (Berkeley Software Distribution), during the spring of 1978. Since then UCB has had several of BSD releases. These BSD releases are referred to as 4.0BSD(1980), 4.1BSD(1981), 4.2BSD(1983), 4.3BSD(1986) and 4.4BSD(1993). In fact, UCB made many important technical contributions like Virtual Memory System (VMS), Fast File Systems (FFS), socket facility,

Other’s contribution During the same period, many computer vendors had developed their own Unix systems. For example, Sun Microsystems (a company that was promoted by Bill Joy) developed Sun operating system, which was revised and renamed Solaries. Solaries 7 is one of the widely used OS even today. Digital Equipment Corporation (DEC) developed a system called Ultrix, which was revised and renamed Digital Unix. Microsoft developed a system called XENIX, the first Unix variant to be run on a PC. This OS was based on both AT&T and BSD systems. XENIX was finally sold to SCO (Santa Cruz Operations). Later, SCO developed its own version of these systems—named SCO Unixware-7 and the SCO open server. Other important systems developed are AIX (by IBM), HP–UX (by HP) and IRIX (by Silicon Graphics).

From the mid-1970s there have been many variants of the Unix system. One of the reasons for this is that being a telephone company, AT&T was not permitted to sell computer-based products. However, it could do so free of cost or for a nominal fee. Because BSD was also giving its products free of cost, many obtained the copies of Unix and worked on them. This resulted in a number of Unix variants. One of the important points that worked against the popularity of any Unix variant for a long time was its user-unfriendliness.

Attempts were made to standardize the Unix system. The first attempt was made by the IEEE standards board. This group came out with a set of rules that should be complied with for an OS to be called standard Unix. These set of rules are widely known as POSIX (Portable Operating System Unix). Now POSIX has also undergone many revisions. The latest one is IEEE 1003.10. In fact, AT&T also has its own standard called Unix international (UI). IBM, HP and DEC also formed a consortium called Open Software Foundation for the same purpose.

In August 1991, a system called Linux was announced by Linus Torvalds (who was only 21 years then) in Finland. Actually it was based on a system called Minix (chiefly developed by Andrew S Tanenbaum) which again was based on Unix. It brought in the speed, efficiency and flexibility of Unix to a PC environment, thereby using the advantages of all the capabilities of Unix. In March 1994, Torvalds released the 1.0 kernel of the Linux. Actually Linux is an open source program—its source code is freely available. Anyone can work on it and make enhancements to it. As a result, it is under constant development. Like other Unix variants it was also initially popular only among the researchers and programmers at universities and research environments. However, at present, Linux has become widely popular among commercial and industrial circles along with the universities and research organizations around the world. Today, Linux has many flavors and can be found on computers ranging from desktops to corporate servers. Red Hat Linux is one of the most popular flavors of Linux. All versions of Linux may be downloaded free of cost from the Web.

1.2 WHAT IS UNIX?

Unix is an operating system. An operating system is a software that acts as an interface between the user and the computer hardware. An operating system acts as a resources manager. Here resources mean hardware resources like the processor, the main memory, the hard disk, I/O devices and other peripherals. In addition to being a multi-user operating system, Unix gives its users, the feeling of working on an independent computer system. Unix also provides communication facility with other users who are connected to the system either directly or indirectly, using networking. It is highly portable and has a large number of utilities and can work both on desktops as well as network environments with equal ease.

1.2.1 Salient Features of Unix: Unix is a multi-tasking operating system—has the ability to support concurrent execution of two or more active processes. Here it may be noted that an instance of a program in execution is known as a process.

Unix is a multi-user operating system—has the ability to support more than one user to login into the system simultaneously and execute programs. For this, the Unix presents a virtual computer to every user by creating simulated processors, multiple address spaces and the like.

The difference between multi-tasking and multi-user system is subtle. In multitasking, different tasks like processes running concurrently belong to one user whereas in a multi-user environment, different tasks belong to different users. However, from the system point of view, the concurrently running tasks are just different processes—them belonging to the same user or to different users is immaterial.

Unix operating system is highly portable. Compared to other OS, it is very easy to port Unix on to different hardware platforms with minimal or no modifications at all. This is because a larger chunk of Unix is built on the language C, which itself is highly portable.

As already mentioned, Unix operating system supports multi-users. These users might be directly connected to the same machine through different terminals or may be connected to different machines that are interconnected. Though initially Unix had no interconnection networking with different computers, the development of communication protocols like TCP/IP have made this possible. This has enabled different users connected to the computer networks to exchange information in the form of e-mail and shared data.

As Unix is a multi-user system, there is every chance that a user may intrude into another user's area either intentionally or unintentionally. Because the security of every user as well as the system is very important, Unix offers solid security at various levels, beginning from the system startup level to accessing files as well as saving data in an encrypted form.

One of the very important key features of any Unix system is that it treates everything, including memory and I/O devices. as files. Thus, there are a large number of files under any Unix environment. Unix has a very well-organized file and directory system that allows users to organize and maintain these files/directories easily and efficiently.

1.3 UNIX COMPONENTS

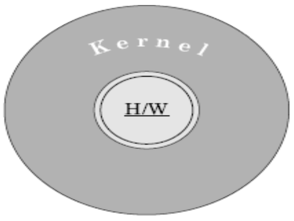
The unix system is said to consist of the following three major components.

1. The kernel
2. The shell
3. The file system

In addition to the above components all commercial Unix systems also include other general utility programs.

1.3.1 The Kernel

The kernel is the heart of any Unix operating system. This kernel is relatively a small piece of code that is embedded on the hardware. Actually, it is a collection of programs that are mostly written in C. Every Unix system has a kernel (just one) that gets automatically loaded on to the memory as soon as the system is booted. As the kernel sits on the hardware it can directly communicate with the hardware.



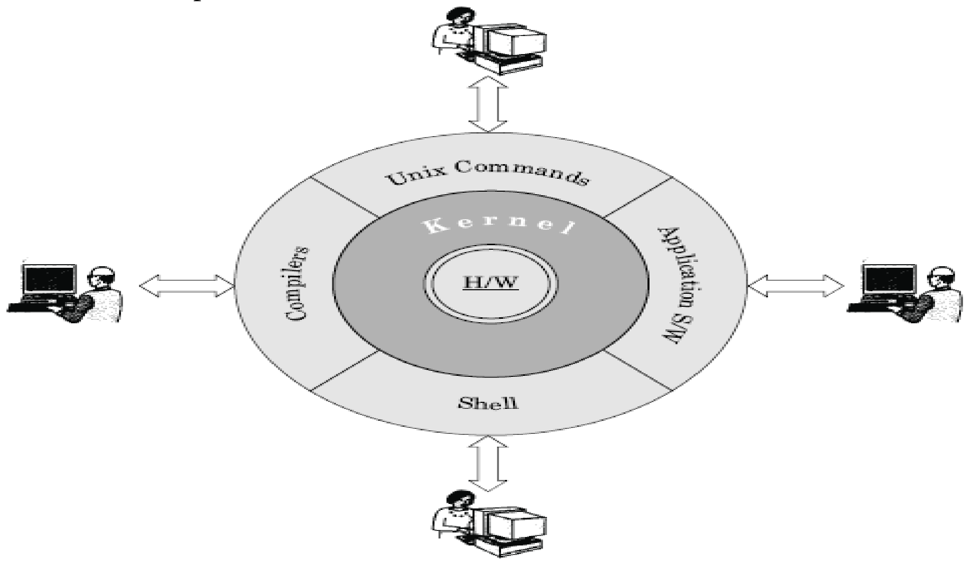
The kernel

In fact, the kernel is the only component that can communicate with the hardware directly. It is the kernel that manages all the system resources like memory and I/O devices, allocates time between users and processes in the case of multi-user environment, decides process priorities, manages interprocess communication (IPC) and performs many other such tasks.

Earlier, all the programs that were a part of a kernel, were integrated together and moved onto the memory during booting. Such integrated kernels are referred to as monolithic kernels. However, only the just-necessary module is moved onto the memory during booting, is called a microkernel. Other modules are moved in and out of the memory depending on the requirement.

1.3.2 The Shell

Every Unix system has, at least, one shell. A shell is a program that sits on the kernel and acts as an agent or interface between the users and the kernel and hence the hardware. It is similar to the command prompt in the MS-DOS environment. The shell has certain programming capability of its own. Using this capability, programs called shell programs can be written. Generally shell programs are called shell scripts.



Unix system components

Types of shells There are different types of shells available. Some of them are discussed here.

The Bourne shell (sh) This is the most common shell available on Unix systems and the first major shell to be developed. This shell is widely used. It has been named after its author, Stephen Bourne at AT&T Bell Labs. This shell is distributed as the standard shell on almost all Unix systems.

The C shell (csh) Bill Joy developed this shell at UCB as a part of the BSD release. It is called the C shell because its syntax and usage is very similar to the C programming language. Unfortunately this shell is not available on all machines. Shell scripts written in the C shell are not compatible with the Bourne shell. One of the major advantages of the C shell over the Bourne shell is its capability to execute processes in the background. A version of this shell called tcsh is available free of cost under Linux.

The Korn shell (ksh) This shell was developed by David Korn at AT&T Bell labs. Basically it is built on the Bourne shell. It also incorporates certain features of the C shell. At present it is one of the widely used shells. It can run Bourne shell scripts without any modifications. One of its versions, the public-domain Korn shell (pdksh), comes with Linux free of cost.

The Bourne-Again shell (bash) This shell was developed by B Fox and C Ramey at Free Software Foundation. Certain Linux operating system variants come with this shell as its default shell. This is clearly a free ware shell.

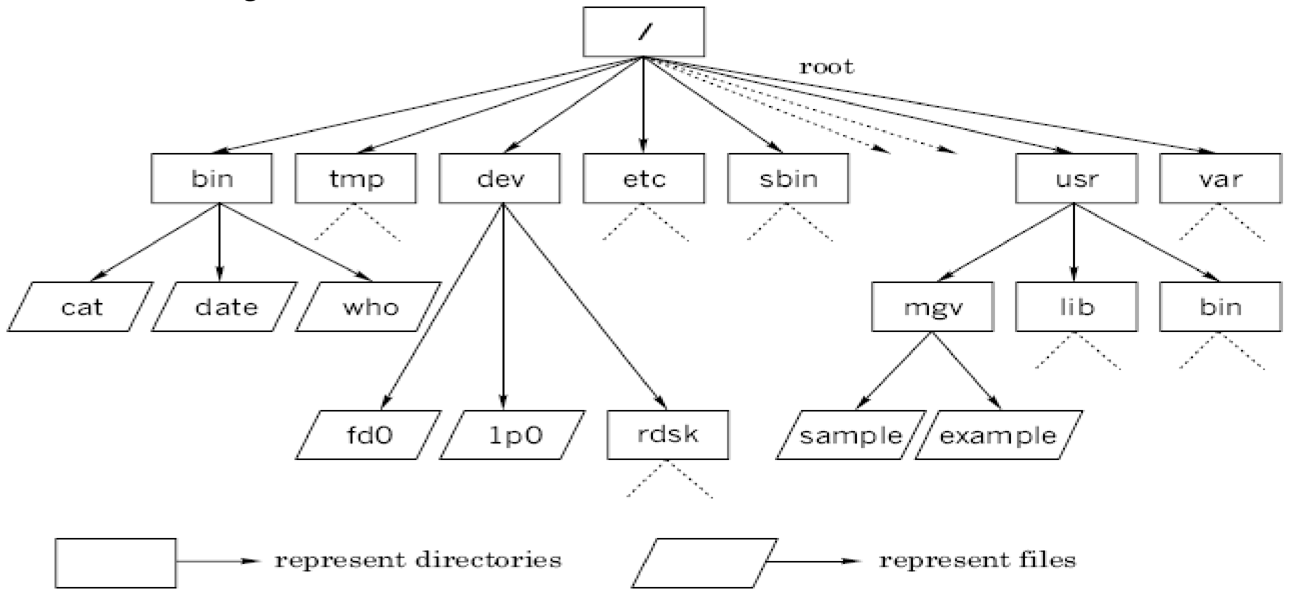
Shell as a Command Processor As already discussed, the shell acts both as a command processor and a small programming language. Given here is a brief account of the behaviour of the shell as a command processor. When interpreting a command line given at its prompt, the shell follows one or more or all of the following steps, depending on the contents of the command line given to it.

1. It parses the command line and identifies each and every word in it and removes additional spaces or tabs present, if any.
2. Evaluates all the variables present that might be prefixed with a $.
3. If commands are present within back quotes, they are executed and their output is substituted into the command line. In other words, command substitution takes place.
4. It then checks for any redirection of the input and/or output and establishes the connectivity between the concerned files accordingly.
5. It then checks for the presence of wildcard characters like *, ? and [,]. If any of these characters are present, file name generation and substitution take place.

It then looks out for the required commands as well as files, retrieves them and hands them to the kernel for execution. The route or the path taken for looking out for the required commands will be in the PATH shell variable. Also the semicolon that allows multiple commands, and logical operators are taken care of by the shell.

1.3.3 The File System

A file system is another major component of a Unix system. Unix treats everything—including hardware devices—as a file. All the files in a Unix system are organized in an inverted tree-like hierarchical structure. This structured arrangement in which all the files are stored is referred to as a file system. A file system could be local to a system or it could be distributed. Local file systems store and manage their data on devices directly connected to the system. Distributed file systems allow a user to access files residing on remote machines.



1.4 USING UNIX

To use Unix, one has to get into the Unix environment. The process of getting into the Unix environment is known as *logging in* into the system. As soon as the system is booted a daemon called init gets started. This init daemon spawns a process called getty for every terminal. Each one of these gettys print the login prompt on the respective terminal. When a user attempts to enter into the Unix environment, that is, tries to login, the login program is executed in order to verify the user name and the password. A file called password file under the /etc directory contains a line for every user, containing the user’s login name, numerical user id, encrypted password, home directory, and other such information. When the user logs in, the login program encrypts the password just read from the terminal and compares it with the password in the password file. If they agree, the login is permitted; if not, it is disallowed. Every user has a user id as well as a password allocated to them by the system administrator. This is true even in the case of single-user systems. However, it may be noted that the user will be the system administrator in the case of single-user systems. The sequence of events in a complete login process can be listed as follows.

- The user enters a login name at the getty's login prompt on the terminal.
- getty executes the login program with the login name as the argument.
- login requests for a password and validates it against /etc/passwd.
- login sets up the TERM environment variable and runs a shell.
- The shell executes the appropriate startup files like .profile.
- The shell then prints a prompt, usually a $ or a % symbol and waits for further input. This indicates the successful entry made into a Unix environment with a proper shell.

The above-listed sequence of events that take place during a login process is schematically shown

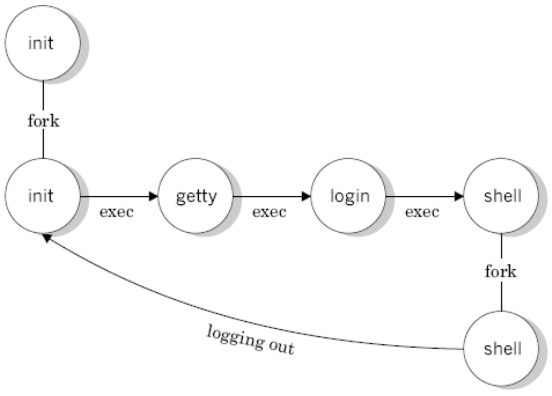


Fig: The log process

When the user completes the session with the system he comes out of the Unix environment. The process of coming out of the Unix environment is known as *logging out.* As soon as the user logs out, the control returns to the init daemon, which in turn spawns a new getty on the corresponding terminal. This facilitates a new user to login to the system.

1.4.1 The Shell Prompt

Successful login into a Unix system is indicated by the appearance of a prompt called the shell prompt or system prompt on the terminal. The character that appears as a prompt depends on the shell used.

List of Default Prompts

Prompt	***Shell***
$ (dollar)	Bourne and Korn shells (sh, bash and ksh)
% (percent)	C shells (csh and tcsh)
# (hash)	Any shell as root

1.5 COMMANDS IN UNIX

Unix has a large number of commands. A list of some general features of a Unix command is given below.

1. A Unix command is a program written to perform certain specific action. All such programs have a name. For example, the program that is used to print today's date in a specific manner has the name date and the program that is used to create a small file or display the contents of a file has the name cat and so on.

2. All Unix commands are written using lower case letters. For example, cat, ls, who, date, and so on.
3. Almost all the Unix commands are cryptic. For example, cat stands for concatenation, ls stands for listing and so on. Unix commands were developed to be cryptic because it was developed by researchers for researchers and the early computer systems were very slow which demanded more time for typing, editing and executing long commands.
4. Unix commands can have zero, one or more number of arguments associated with them.
5. Unix commands can also have format specifiers as well as options associated with them. Format specifiers, whenever present, are indicated by the + character. Options, whenever present, are indicated by hyphen (–). There could be many number of options associated with a command. It is interesting to note that the listing command (ls) has nearly two dozens options that could be used with it.
6. In certain situations, a Unix command with its arguments or a series of commands may not fit in a single line (80 characters). In such cases it may overflow. This is permitted in Unix. Whenever there is an overflow, it is indicated by the appearance of a special prompt in the form of a > symbol in the beginning of the next line. Such a special prompt is known as the secondary prompt.
7. A current Unix command can be killed by using either <delete> or <ctrl-u> command.
8. Commands can be given to the system even when a command given earlier is being executed in the background. This is not possible with the Bourne shell, sh.

1.5.1 Types of Unix Commands

Basically there are two types of Unix commands. They are—external commands and internal commands.

External Commands A command with an independent existence in the form of a separate file is called an external command. For example, programs for the commands such as cat and ls, exist independently in a directory called the /bin directory. When such commands are given, the shell reaches these command files with the help of a system variable called the PATH variable and executes them. Most of the Unix commands are external commands.

Internal Commands A command that does not have an independent existence is called an internal command. Actually the routines for internal commands will be a part of another program or routine. For example, the echo command is an internal command as its routine will be a part of the shell's routine, sh. In other words the echo command is built into the shell. As such, internal commands are also called the built-in commands. cd and mkdir, are two examples of internal commands.

1.6 SOME BASIC COMMANDS

Unix has several hundreds of commands within it. Most of them are simple and are powerful. Some of the commands are general in nature from the user's point of view. A few of such commands are introduced in the following sections.

1.6.1 The echo Command

The echo command is used to display messages. It is quite useful in developing interactive shell programs. It takes zero, one or more number of arguments. Arguments may be given either as a series of individual symbols or as a string within a pair of double quotes (" "). Some examples are given below.

```
 $ echo
                                   #A Blank line is displayed
$
```

1.
```
$ echo I am studying computer science.
I am studying computer science.
$
```
2.
```
$ echo I am studying computer science.
I am studying computer science.
$
```
3.
```
$ echo "I am studying computer science."
I am studying computer science.
$
```
4.
```
$ echo The home directory is $HOME
The home directory is /usr/mgv
$
```

1.6.2 The tput Command

This command is used to control the movement of the cursor on the screen as well as to add certain features like blinking, boldface and underlining to the displayed messages on the screen. Such facilities might be used to add aesthetic value to the shell programs. For example, this command along with the clear argument clears the screen and puts the cursor at the left– top of the screen. However, it may be noted that clear itself is a command which alone could be used to clear the screen.

```
$tput clear
```

This command along with the cup argument and certain co-ordinate values is used to position the cursor at any required position on the screen. An example is given below.

```
$tput cup 10 20
```

When the above command line is executed, the cursor will be placed at the tenth row and the twentieth column on the screen. Now, if an echo command is given, the message will be displayed starting from the new position.

The number of rows and columns on the current terminal is known by using the lines and cols as arguments to the tput command as shown in the following examples.

```
$tput   lines
48
$
$tput   cols
142
$
```

From the above examples, it is seen that there are 142 columns and 48 lines on the current terminal.

1.6.3 The tty Command

In Unix, every terminal is associated with a special file, called the device file. All the device files will be present in the /dev directory. A user can know the name of his device file on which he is working by using the tty command, as shown in the example below.

```
$tty
/dev/tty01
$
```

Here, tty01 is the device file name and will be available in the directory /dev. Under Linux, the output of this command will be as shown below.

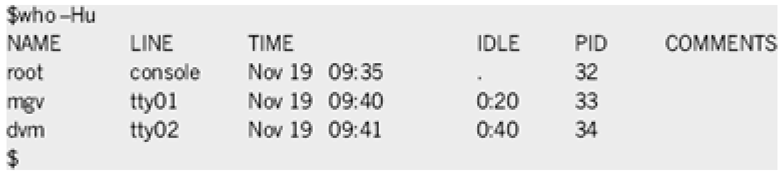
```
$tty
/dev/pts/0
$
```

1.6.4 The who Command

Unix maintains an account for all the current users of the system. Because it is a multi-user system it is prudent for the user to be aware of other current users so that s/he can communicate with them, if required.
The user can know login details of all the current users by using the who command. The use of who command provides a list of all the current users in the three-column format by default, as shown follows.

```
 $who
root        console        Nov  19   09:35
mgv         tty01          Nov  19   09:40
dvm         tty02          Nov  19   09:41
$
```

The first column shows the name of the users, the second column shows the device names and the third column shows the login time.
Some options like –H, –u and –T can be used with this command. The –H option provides headers for the columns and the –u option provides more details like idle time, PID and comments as shown in the example below.



```
$who –Hu
NAME       LINE       TIME                 IDLE     PID      COMMENTS
root       console    Nov 19  09:35        .        32
mgv        tty01      Nov 19  09:40        0:20     33
dvm        tty02      Nov 19  09:41        0:40     34
$
```

If any terminal is idle (not active) for the last 1 minute, the information, that is, for how long that terminal is idle will be indicated on the IDLE column. Thus, 0:20 indicates that mgv's terminal is IDLE for the last 20 minutes. This information will be useful to the system administrator. The PID indicates the process identification number.
The self-login details of a user can be obtained as a single line output using am and i arguments along with the who command as follows.

```
 $who am i
mgv       tty01       Nov  19   09:40
$
```

Generally the who command is used by the system administrator for monitoring terminals.

1.6.5 The uname Command

Using this command one can know the details of one's Unix system. We already know that there are many Unix variants. When this command is used, it gives the name of the Unix system being used by the user. Certain options like r, v, m and a, can be used with this command. Following are given some examples.

1.
   ```
   $uname
   Linux
   $
   ```

2.
   ```
   $uname –r
   2.4.18 – 3             #release details
   $
   ```

3.
   ```
   $uname –m
   i686                   #machine details
   $
   ```

The use of the option –v gives the version of the system being used. The use of the –a option gives all the details of the system.

1.6.6 The date Command

Using this command the user can display the current date along with the time nearest to the second. This is done with the help of the Unix system's internal clock that runs with battery back up during shutdowns.

```
 $date
Sat  Jan 10  11:58:00  IST  2004
$
```

This is one of the very few commands that allows the use of format specifiers as arguments. Format specifiers are single characters using which, one can print the date in a specific manner. Each format specifier is preceded by a + symbol followed by the % operator. For example, by using the format specifier m one can display only the month in the numeric form as follows.

```
 $date  +%m
09
$
```

Instead of the numeric form, the name of the month can be displayed using the h format specifier as shown below.

```
 $date   +%h
Sep
$
```

More than one *format specifier* can be specified at a time. In such cases either double quotes (" ") or single quotes (' ') are used.?

```
 $date +"%h %m"
Sep 09
$
```

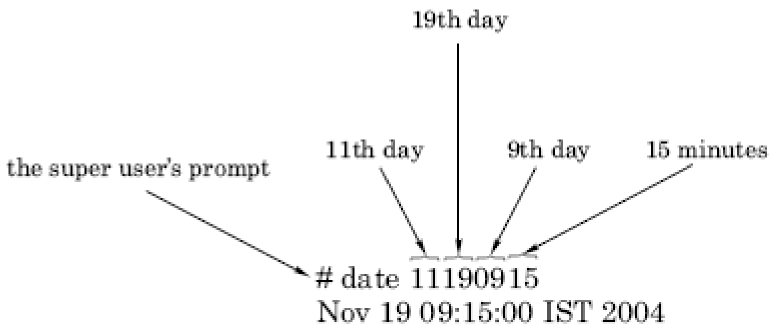
Some of the other codes that could be used as format specifiers with the date command are

1. D and d for the day of the month. (D gives the day in the format mm/dd/yy, where as d gives the day in the format dd).
2. Y and y for the year (Y gives all the four digits of the year, whereas y gives only the *last* two digits).
3. H, M and S stand for hour, minute and second, respectively.

Many number of options like u, r, R, f, can also be used with this command. For example, the use of the u option displays the universal time (Greenwich Mean Time) as shown in the example below, where UTC is Coordinated Universal Time.

```
 $date –u
Sat  Sep 25  05:58:20  UTC  2004
$
$date "Today's date is +%D"
Today's date is 03/16/04
$
```

The System Date The date command is also used by the system administrator to change or reset the system date. This usage has a different syntax. For changing the date, that is, to set the date, a numeric argument is given. This argument is usually an eight characters long string having the form MMDDhhmm (month, day, hour in 24-hour format and minutes) followed by an optional two-digit year. A typical example of this is given below.



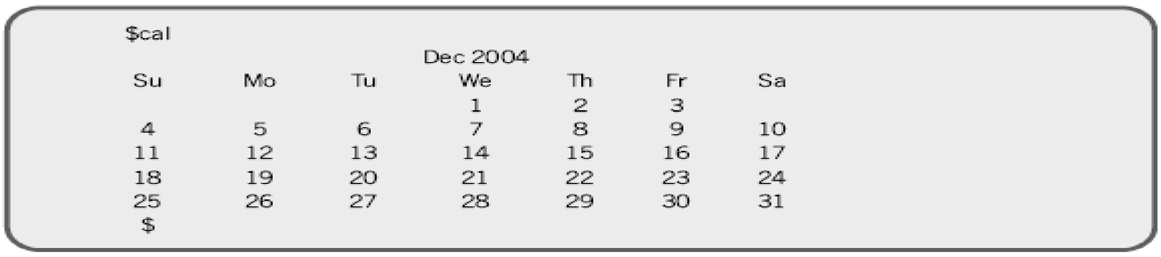
```
# date 11190915
Nov 19 09:15:00 IST 2004
```

Fig. Setting system date

Under Unix, lot of importance is given to the system time. For example, the time at which a file is created, modified or accessed is recorded depending upon the system time. Also, login and logout times are recorded based on the system's time. There are certain commands (like, cron and at) whose action depends upon the system's time. For these and more reasons the system should sufficiently indicate the correct time. However, excessive manipulation of the system time should be avoided.

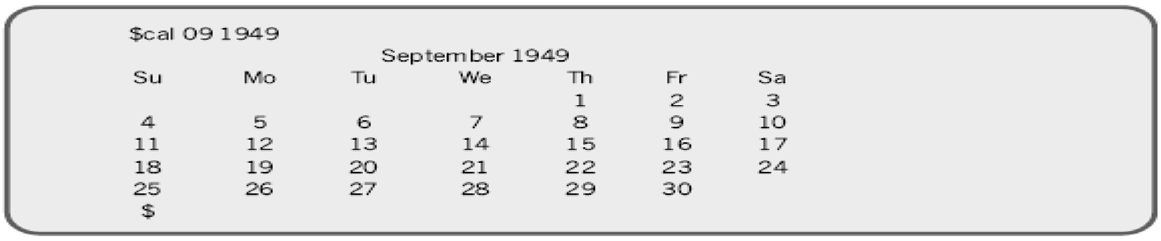
1.6.7 The cal Command

This command is used to print the calendar of a specific month or a specific year. It prints the Gregorian or New Style calendar for any month or year between the years 1 and 9999. When this command is used without any arguments, the calendar of the current month of the current year will be printed, as shown below.



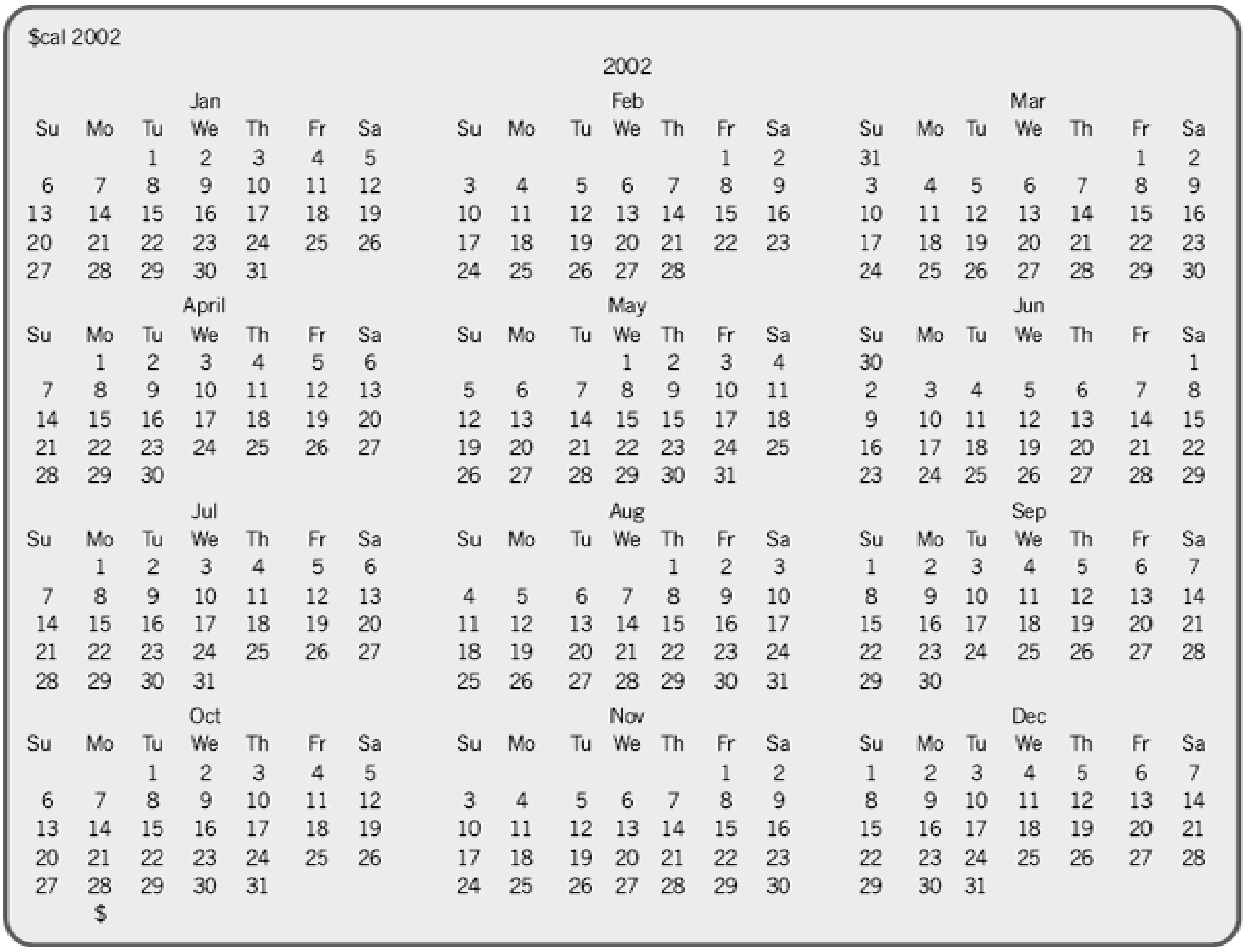
```
$cal
                    Dec 2004
Su    Mo    Tu    We    Th    Fr    Sa
                  1     2     3
4     5     6     7     8     9     10
11    12    13    14    15    16    17
18    19    20    21    22    23    24
25    26    27    28    29    30    31
$
```

When two numeric arguments, are given the first argument will be considered as the month, the second argument will be considered the year and the calendar for that month of that year will be printed as shown in the following example.



```
$cal 09 1949
               September 1949
Su    Mo    Tu    We    Th    Fr    Sa
                        1     2     3
4     5     6     7     8     9     10
11    12    13    14    15    16    17
18    19    20    21    22    23    24
25    26    27    28    29    30
$
```

When given with a single numeric argument, the complete calendar for the entire year represented by the numeric argument will be printed as follows.



```
$cal 2002
                                          2002
                 Jan                                  Feb                                  Mar
 Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa
          1   2   3   4   5                                   1   2          31                   1   2
  6   7   8   9  10  11  12           3   4   5   6   7   8   9           3   4   5   6   7   8   9
 13  14  15  16  17  18  19          10  11  12  13  14  15  16          10  11  12  13  14  15  16
 20  21  22  23  24  25  26          17  18  19  20  21  22  23          17  18  19  20  21  22  23
 27  28  29  30  31                  24  25  26  27  28                  24  25  26  27  28  29  30
                April                                 May                                  Jun
 Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa
      1   2   3   4   5   6                       1   2   3   4          30                       1
  7   8   9  10  11  12  13           5   6   7   8   9  10  11           2   3   4   5   6   7   8
 14  15  16  17  18  19  20          12  13  14  15  15  17  18           9  10  11  12  13  14  15
 21  22  23  24  25  26  27          19  20  21  22  23  24  25          16  17  18  19  20  21  22
 28  29  30                          26  27  28  29  30  31              23  24  25  26  27  28  29
                 Jul                                  Aug                                  Sep
 Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa
      1   2   3   4   5   6                           1   2   3           1   2   3   4   5   6   7
  7   8   9  10  11  12  13           4   5   6   7   8   9  10           8   9  10  11  12  13  14
 14  15  16  17  18  19  20          11  12  13  14  15  16  17          15  16  17  18  19  20  21
 21  22  23  24  25  26  27          18  19  20  21  22  23  24          22  23  24  25  26  27  28
 28  29  30  31                      25  26  27  28  29  30  31          29  30
                 Oct                                  Nov                                  Dec
 Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa          Su  Mo  Tu  We  Th  Fr  Sa
          1   2   3   4   5                                   1   2           1   2   3   4   5   6   7
  6   7   8   9  10  11  12           3   4   5   6   7   8   9           8   9  10  11  12  13  14
 13  14  15  16  17  18  19          10  11  12  13  14  15  16          15  16  17  18  19  20  21
 20  21  22  23  24  25  26          17  18  19  20  21  22  23          22  23  24  25  26  27  28
 27  28  29  30  31                  24  25  26  27  28  29  30          29  30  31
      $
```

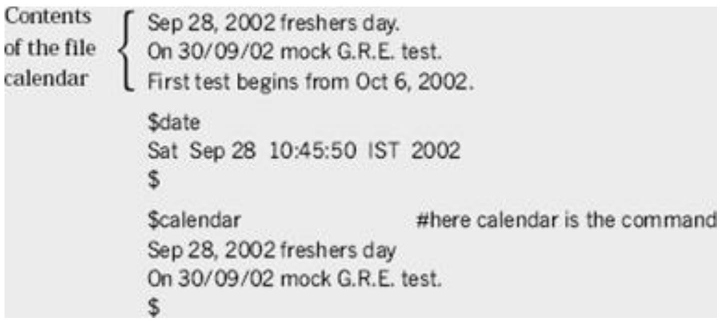
On some systems the month can be given in short—as Dec, Sep, and so on. Care should be taken to give the year in proper numeric format. For example, the year 1949 should be given as 1949. If, by chance, it is given as only 49, the calendar of AD 49 will be printed.

If, for any reason, the calendar-display on the monitor scrolls up (that is, it cannot fit into a single screen and moves up), the scrolling can be paused using the <ctrl-s> command and continued using the <ctrl-q> command. However, the use of the more command is recommended in such cases. The use of the more command displays the output one page at a time.

1.6.8 The calendar Command

It is like an engagement dairy that contains text information and offers a reminder service based on a file called the calendar. This file must be in the present working directory/home directory. This file is created and managed by the user with the help of an editor (like vi editor) on the system. This command works on today and tomorrow dates concept. The present working day's date (picked up from the machine) is taken as today and the days upto and including the next working day are treated as tomorrow. For example, if it is a five-day working week and today is a Friday, then all the days upto and including the next Monday will be considered as tomorrow.

The success of this facility depends upon the date formats permitted to write the calendar text file and the way in which the calendar command searches the file calendar. Typically a calendar file may contain the information as shown below.


```
Contents   ⎧ Sep 28, 2002 freshers day.
of the file⎨ On 30/09/02 mock G.R.E. test.
calendar   ⎩ First test begins from Oct 6, 2002.
             $date
             Sat  Sep 28  10:45:50  IST  2002
             $
             $calendar                    #here calendar is the command
             Sep 28, 2002 freshers day
             On 30/09/02 mock G.R.E. test.
             $
```

1.6.9 The passwd Command

As already mentioned, Unix is a multi-user system due to which there is always a security threat. Many levels of security measures have been included into Unix systems. The simplest and most widely used by all individual users is the use of passwords. During the addition of new users, the system administrator permits or authorizes the new user by assigning a unique password to him or her. A user can change his or her password using the passwd command. In fact, users are advised to change their passwords quite often. The following illustration shows how a user can change the password using the passwd command.

```
$passwd
Old Password: ********
New Password: ********
New Password: ********
$
```

As soon as the passwd command is executed, the system first asks for the old password. When the old password is keyed in correctly the system asks for the new password twice. First time for entering the new password and second time for the confirmation. When everything is keyed in properly, the $ prompt reappears. It may be noted that neither the old password nor the new password is displayed.

1.6.10 The lock Command

For security reasons it is necessary that one should not leave any terminal session unattended as someone else might sneak onto the system and cause problems intentionally or unintentionally. It is advisable either to logout or lock the terminal session before leaving it temporarily. The lock command is used for locking a session for any required amount of time.

Generally this command is used without any argument as shown in the following illustration. By default, the user can lock it for 30 minutes. This locking period can be changed by assigning a different value for the systemvariable DEFLOGOUT. When the lock command is given, the terminal asks for a password twice as shown in the example below.

```
password:********
re-enter password:********
terminal locked by mgv 0 min ago
```

The password used here need not be the actual password that is used to log into the system. It could be any temporary password. One can lock a terminal for a maximum period of 60 minutes. A numeric option may be used to lock the terminal for any period ranging between 1 and 60 minutes as shown in the example below.

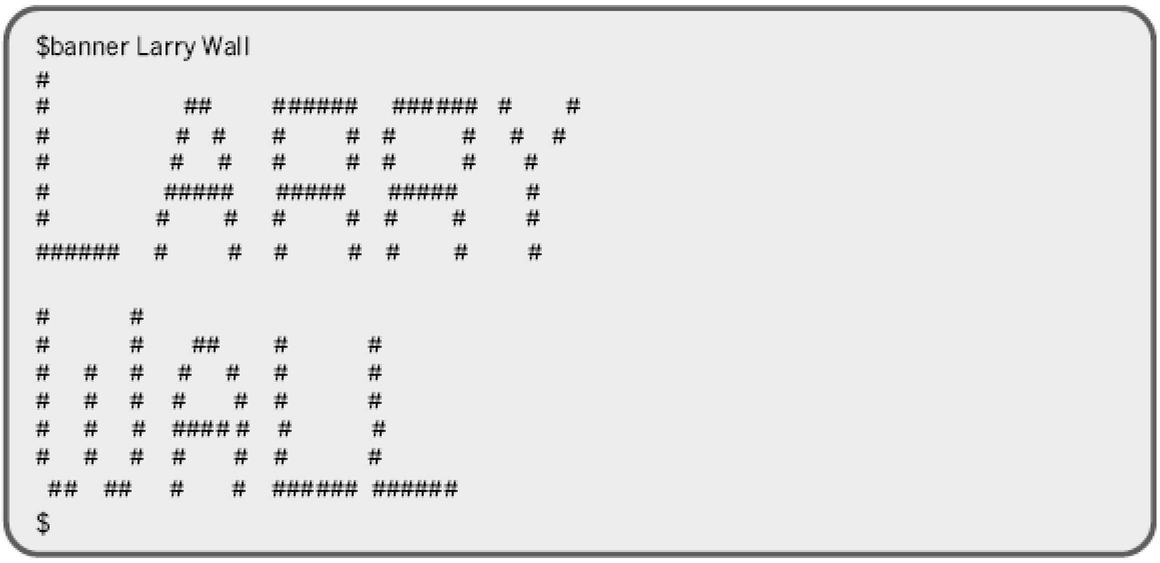
```
 $lock–45              #locks for 45 minutes
```

The locked terminal can be unlocked by re-entering the password with which the terminal was locked earlier. If the terminal is not activated before the lock period expires, the system automatically gets unlocked at the end of the specified time period.

Many Linux distributions include a locking command called vlock. The lock command is used to lock sessions individually, whereas vlock may be used to lock all individual sessions simultaneously. Also a utility called lock screen is available, with many modern OS, using which a session on a terminal can be locked.

1.6.11 The banner Command

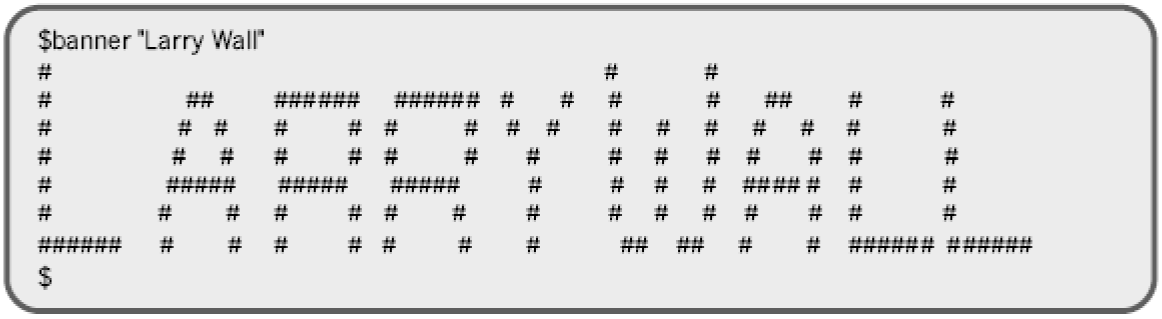
This command is available on SCO Unix (but not on Linux). It is used to display banners or posters. This command simply produces a blown-up version of the characters that are supplied with the command. An example is given below.



```
$banner Larry Wall
#
#           ##     #######  #######  #     #
#          #  #    #     #  #     #   #   #
#         #    #   #     #  #     #    # #
#        ######    ######   ######      #
#       #      #   #     #  #    #      #
####### #      #   #     #  #     #     #

#     #
#     #    ##    #       #
#  #  #   #  #   #       #
#  #  #  #    #  #       #
#  #  #  ###### #       #
#  #  #  #    #  #       #
 ## ##   #    #  ####### #######
$
```

It may be observed that there are two arguments and each argument has been printed on a separate line. It prints a maximum of 10 characters per line. In case an argument consists of more than 10 characters, only first 10 characters will be printed and the remaining will be truncated. As seen from the example above, the output will be made up of the # (hash). A series of arguments may be given as a single argument in the form of a string as shown in the following example.



```
$banner "Larry Wall"
#                                            #     #
#           ##     #######  #######  #     # #     #    ##    #       #
#          #  #    #     #  #     #   #   #  #  #  #   #  #   #       #
#         #    #   #     #  #     #    # #   #  #  #  #    #  #       #
#        ######    ######   ######      #    #  #  #  ######  #       #
#       #      #   #     #  #    #      #    #  #  #  #    #  #       #
####### #      #   #     #  #     #     #     ## ##   #    #  ####### #######
$
```

Further, it may be noted that Larry Wall is the creator of the Perl language.

1.6.12 The cat Command—Creating Small Files

This is one of the most useful and quite frequently used Unix commands. One of the basic purposes of this command is to create small Unix files. A file can be created by writing a command line as shown in the following example where review is the name of the file being created.

```
$cat > review
A > symbol following the command
means that the output
goes to the file name following it.
<ctrl-d>
$
```

In the above example after executing the $cat > review command, the $ prompt vanishes and the system is ready to accept the input from the standard input—the keyboard. At this point the user can type in whatever—s/he wants. The input operation is terminated by using <ctrl-d> on a new line. The input termination command <ctrl-d> does not get into the file. If the file being created already exists, it will be overwritten. One of the drawbacks of this method of creating files is that it lacks editing capabilities. Therefore, it is seldom used for creating files of any considerable size. For creating files of considerable size, editors like vi and emacs are used.

Name:cat

Purpose: It is used

i. To create the files
ii. To display the contents of a file
iii. To Append the files

Syntax:

```
cat   [option(s)]   file(s)
cat   file1  file2>newfile
cat   file1 >> file2
```

Options:

Option	**Description**
-e	Print $ at the end of line
-n	Numbering lines
-s	Silent(no error messages)
-t	Print tabs and form feeds
-v	Print control characters
-b	Ignore blank lines used with –n option.

<u>Examples & Description:</u>

- Simple *cat* command accepts the data from the standard input (key board) and immediately prints the data to standard output(monitor.)
- It accepts and prints the data until it encounters the **eof** character**(ctrl-d)**
- **EX:**

```
$cat
 1234
1234
Abcd
Abcd
Xyzl
Xyzl
[ctrl-d]
$
```

(i)File Creation

- The second use for *cat* is file creation.
- It is accomplished by typing *cat* followed by the output redirection operator and the name of the file to be created, then pressing ENTER and enter data into the file finally simultaneously pressing the CONTROL and *d* keys. So cat command stops to accepts data from key board .For example, a new file named *file1* can be created by typing **cat > file1**
- If a file named *file1* already exists, it will be *overwritten* by the new.

```
$cat>x
Unix is a multitasking os
Unix is a multi user os
In unix the command prompt is $
We can set our own command prompt
[ctrl-d]
$
```

With this a new file **x** is created.

- Using output redirection operator(>) in cat command we can copy the contents of more files into specified file.
- **cat f1 f2 f3 >f4**

  It copies all three files f1,f2,f3 contents into f4.

  Ex:

```
$cat  f1
This is pen
$cat  f2
This is rose
$cat  f3
This is jasmine
```

```
$cat f1 f2 f3>f4
$cat f4
This is pen
This is rose
This is jasmine
```

(ii) File displaying:

Cat command displays the contents of a file on monitor.

```
$cat  x
Unix is a multitasking os
Unix is a multi user os
In unix the command prompt is $
We can set our own command prompt
$
```

```
$cat   y
This is pen
$cat f
This is jasmine
$cat   z
 This is rose
```

- The cat command displays the contents of 2 or more files at the same time.
- the following command will concatenate copies of the contents of the four files *x*,y,z, and *f* and displays all 4 file contents.

```
$cat  x  y  z  f
Unix is a multitasking os
Unix is a multi user os
In unix the command prompt is $
We can set our own command prompt
This is pen
This is rose
This is jasmine
$
```

- While displaying more number of files, if any one file is not existed in the working directory ,then the cat command displays the contents of all available files and generates an error message about the non-existent file.
- **Ex:**

```
$cat    y  z  f   p
        This is pen
        This is rose
        This is jasmine
      Cannot open: p no such file or directory
      $
```

Here **x,y,z,f** files are existed so the contents of all these files are displayed. But **p** is not available so error message was displayed regarding **p**

Options:

-s(suppressing errors): It suppresses the error messages. While displaying the files if the file is not available then no error message is displayed.

```
 $cat  -s  y  z  f   p
This is pen
This is rose
This is jasmine
$
```

- No error message regarding *file p* even though it is not existed. Because of –s option.

-n(numbering to lines):

a. This option gives the number to each line while displaying the output.

```
$cat  x
1:Unix is a multitasking os
2:Unix is a multi user os
3:In unix the command prompt is $
4:We can set our own command prompt
$
```

b. cat command gives line numbers to blank lines also

```
$cat m
 Abc

 Def

 lmn
```

```
$cat  –n  m
1:  Abc
2:
3: Def
4:
5:  lmn
```

-b(excluding Blank lines): This option is used with –n option.
Because of this option blank lines are ignored . so –n option does not give the number to blank lines

```
$cat  –bn  m
1: Abc
2: Def
3: lmn
```

-e (print $ at end of each line): this option prints $ at the end of each line in a file.

```
$cat  x
Unix is a multitasking os$
Unix is a multi user osS
In unix the command prompt is $$
We can set our own command prompt$
$
```

-t(prints tabs and form feeds): It prints tabs and form feeds.

```
$cat>k
this    is       an       example
[ctrl-d]
```

```
$cat  k
this  ^It  is  ^It  an  ^It example
$
```

(iii) File appending:
While creating a new file if the file is not existed *cat* creates new file, if already a file is existed then it is overwritten. to avoid this cat command uses append operator.
It is indicated with >> symbol. It adds one file content to the end of the other file.
Ex:

```
$cat   y
This is pen
```

```
$cat   z
This is rose
```

```
$cat   y>>z
$cat   z
This is rose
This is pen
```

1.6.13 The bc Command

The bc command is both a calculator and a small language for writing numerical programs. Using this command one can perform all the usual arithmetic operations as well as change of bases in the range of base 2–16.
Arithmetic operations are performed using the built-in library functions. The special functions that are available in the library are sin( ), cos( ), arctan( ), ln( ), exp( ), bessel( ). The arguments to the trigonometric functions must be given in radians. Math functions are used by invoking bc with the option –l. Also these math functions are used with the acronyms that follow.

Function	***Acronym***
Cosine	c(n)
Sine	s(n)
Tan	t(n)
Arctan	a(n)
natural log	l(n)
exponential function	e(n)
square root	sqrt(n)
exponent	^

The bc can be used by either entering expressions to be evaluated from the keyboard or running programs stored in files. The user can define one’s own functions and use them.
The syntax used to write numeric programs and to define user-defined functions is similar to the ones used in the C language.

This command uses the infix method of entering data and specifying operations. The required degree of precision is obtained using the scale function available in the library. By default, the scale factor will be 0. It is possible to set a maximum value of 100 to the scale.

This calculator is invoked by just typing in the bc command at the system prompt. When this command is given, a cursor appears on the monitor at which the expression to be evaluated is given. It should be noted that one command line is executed at a time. In other words it works in the interpreter mode. Below are given some examples of such a mode.

```
1.$bc
   sqrt55
   7
   quit
   $
```

```
2.$bc
   scale=4
   sqrt55
   7.4161
   quit
   $
```

```
3.$bc
   ibase=5
   obase=16
   2341
   424
   ibase=16
   obase=5
   424
   2341
   quit
   $
```

From the above examples one can understand the following.

1. The default value of the function scale is 0 (Zero).
2. Precision is set to 4 or any required value using the scale function above.
3. The result is displayed immediately in the next line after the execution of every line.
4. A session with bc is terminated by using the quit command.
5. Base conversion is carried out using ibase and obase functions.
6. ibase stands for the input base and obase stands for the output base. Default values for both ibase and obase is 10.

As already mentioned, numeric programs can be written and used with the bc command. While writing numeric programs only single-character variables are used and only lower case English letters are allowed as variables. There can be a maximum of only 26 variables in a program. However, with Linux, variable names having more than one alphanumeric character with the first character being an alphabet can also be used.

An example that uses the control construct is shown here.

```
$bc
for(i=1;i<=4;i=i+1)          i^2
1
4
9
16
quit
$
```

1.6.14 The spell and ispell Commands: Spell-Check Programs

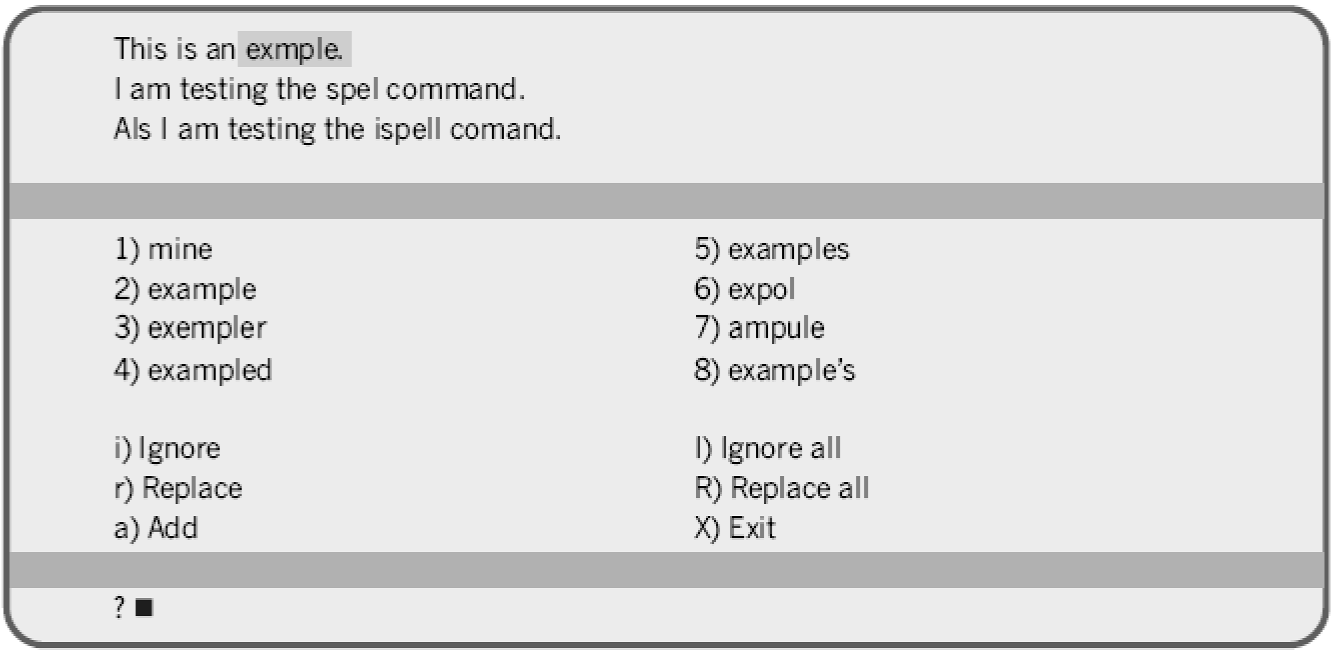
The spell command is the first program that was developed to check for words that are wrongly spelt in a document. This command displays a list of misspelled words in the document used as argument, as shown below.

```
 $cat spell.ux
```

```
This is an exmple.
I am testing the spel command.
Als I am testing the ispell comand.
$
$spell spell.ux
Als
comand
exmple
spel
$
```

As seen from this example all the misspelled words are displayed in alphabetical order. These words are noted down (or saved in a separate file) and necessary corrections are made using an editor. By default, the spell command checks for the spelling based on American usage. Spell checking may be made according to British usage using the –b option. Actually spell check is made by comparing the words in the text with the words on an inbuilt dictionary. A user can provide one's own dictionary also. The ispell command is an interactive spell-check program available in Linux. When used, this command displays a screen full of information in three sections as shown below.

```
$ispell spell.ux
```



```
This is an exmple.
I am testing the spel command.
Als I am testing the ispell comand.

1) mine                         5) examples
2) example                      6) expol
3) exempler                     7) ampule
4) exampled                     8) example's

i) Ignore                       I) Ignore all
r) Replace                      R) Replace all
a) Add                          X) Exit

? ■
```

As seen from the above display, the misspelled words are highlighted. Alternate suggestions (a maximum of 10) for the misspelled words will be given in the middle section. Also the information about alternate actions that one could take will be displayed in this section. A question mark (?) appears in the last section with a blinking cursor along with it. A misspelled word can be substituted with a correct one by using the serial number of the correct word. Suggestions can be ignored. In case none of the suggestions are suitable, an external word can be input and substituted by using the replace command. This new word can be added to the dictionary by using the add action.

1.7 COMMAND SUBSTITUTION

In Unix, it is possible to run a command within a command. For example, the date command can be run within the echo command by writing a command line as follows.

```
$echo Today the date is 'date'
```

```
Today the date is Fri Oct 3 16:25:00 IST 2002
```

As shown in the above example, the command to be executed (that is, date, in this example) within another command (that is, echo in this example) has to be written within a pair of backquotes (''). The shell while parsing the parameters list of the echo command treats the words that are backquoted as a command, executes it and substitutes the result of this execution at the corresponding position in the parameters list. This process is known as command substitution.

In Korn shell the command substitution is accomplished by using a $ sign followed by the command within a pair of parenthesis as shown below.

```
 $echo Today the date is $(date)
Today the date is Fri Oct 3 16:25:00 IST 2002
```

1.8 GIVING MULTIPLE COMMANDS

Normally a single command is given to the shell at its prompt. However, there are many situations when more than one command is given in a single command line. One of the ways of giving multiple commands is to use a semicolon (;) between successive commands as shown below.

```
 $echo "Giving multiple commands"; date; who
```

Commands given in this way does not mutually interact with each other in any manner. They are executed independently one after the other, from left to right as they appear in the command line. Giving multiple commands in a single command line has a definite advantage as the entire command line could be executed as a background job and something else could be done in the foreground. Of course, the Bourne shell (sh) does not permit processing of jobs in the background where as the Korn shell (ksh) does.

1.9 ALIASES—GIVING ALTERNATE NAMES TO COMMANDS

With some of the recent and popular shells like korn and bash it is possible to assign short and meaningful names to a long command or a combination of commands and use these short names as alternate names. Such short names are called aliases of the original command names or combination of commands. This is accomplished using a command called the alias command.

Assume that one needs to know the date, time and users information quite frequently. For this one has to use the who and date commands at all the times. In such circumstances one can use an alias command as follows.

```
 [/home/mgv]alias    whoda=`who;date`
[/home/mgv]_
```

Here one should observe that there should not be any spaces either before or after the equal to (=) operator. After aliasing a command or a sequence of commands one can just use the alias name for the required purpose. In other words, henceforth the word whoda can be used as a command to get the date, time and users information.

A list of all the aliases can be obtained by using the alias command without arguments. An alias is removed by using the unalias command, as shown in the following example. Too much usage of aliases may lead to confusion and is highly error prone. So, one should be very selective while using this facility.

```
 [/home/mgv]unalias   whoda
[/home/mgv]_
```