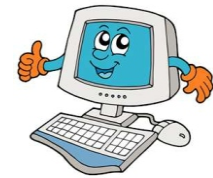


6

Basic Ubuntu Linux Commands



In standard 9 we have studied in depth the working of an operating system called Ubuntu. During the study we learned that once a user logs into the computer system having Ubuntu Linux, he/she can interact with the computer using command line interface or graphical interface. Both these interfaces are important and have their own uses. In this chapter we will learn how to use the command line interface in detail. The command line interface allows us to access the real power of Linux with greater efficiency. It is the most influential parts of any Ubuntu system.

Starting Up the Terminal

To open a command-line console in a graphical interface, a window named terminal window is provided in Linux. To open a terminal window, click on Applications à Accessories à Terminal alternatively you can use CTRL+ ALT + t keys together. A terminal window similar to the one shown in figure 6.1 will be seen on the screen. Note that the look of your actual window may be different from the one shown in figure 6.1 as the user might be different.



Figure 6.1 : Terminal Window

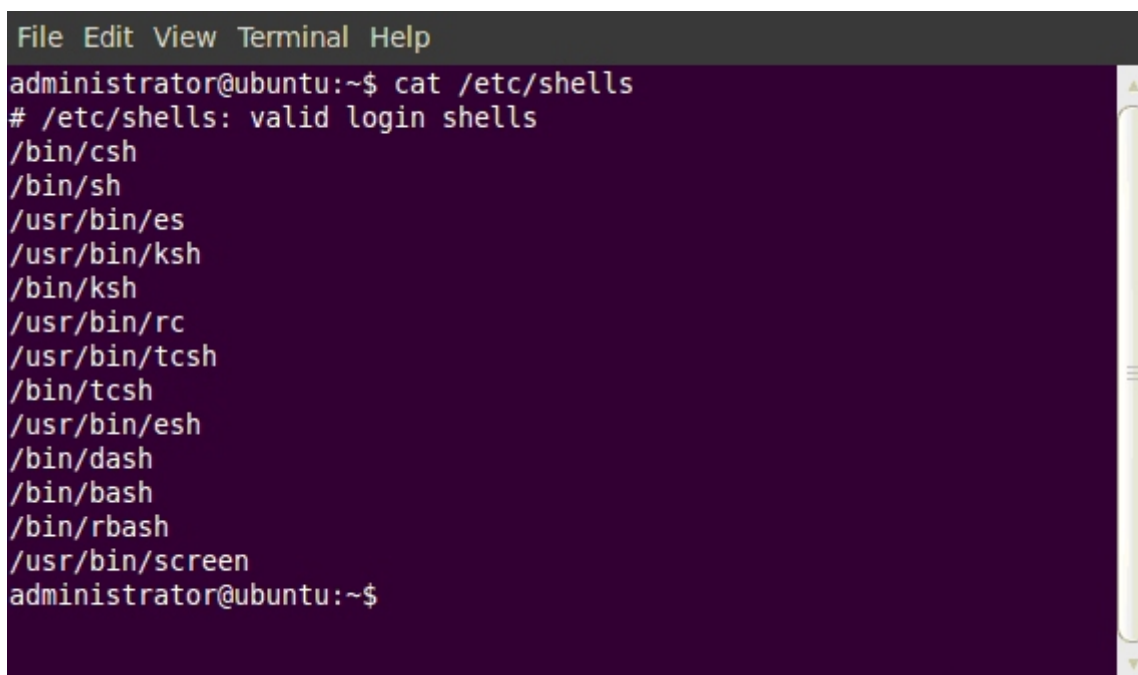
Once the window is clearly visible, you will see a blinking cursor preceded by some letters, and perhaps numbers and symbols, ending with a \$. The first word in that string of characters is username, followed by the symbol @. The symbol @ is followed by the name of the computer that is being used. Finally you will find a colon and the name of the directory you are working in (Generally you start working in your home directory, which is represented by a ~ symbol).

The command prompt indicates that the interface is ready to interact with the user in the form of commands. A command is basically a program that accomplishes certain task. Once the prompt is displayed we can issue commands as described in this chapter. Before discussing various commands, let us revise our knowledge about the term shell. A shell is the command-line interface. Shell is a user program or an environment provided for user interaction. It is a command language interpreter that accepts or issues commands, understands it, interacts with the kernel to execute it and displays results as per given instruction. Numerous shells are available to work on the Ubuntu Linux systems, but the shells available on a particular system may vary.

Some popular shells provided with Linux are Bourne shell (sh), C shell (csh and tcsh), Korn shell (ksh) and bash (sh) shell. Bourne shell with sh as its acronym is the earliest Unix shell used as command line interface. Bourne shell provides basic mechanisms for shell script programming, which allows us to write a program based solely on commands. C shell identified as csh is another shell commonly found on Linux systems. Shell programming can be done using C programming syntax in this shell. The newer version of csh is tcsh. It provides additional shell script programming features to address the limitations of csh. The Korn shell or ksh was developed to combine the features of both sh and csh. Bash shell is a newer version of Bourne shell. Thus it contains same syntax and functions as sh. Nowadays bash is considered standard shell for Linux systems and is thus commonly used and available on all Linux operating systems.

Listing the shells available on the system

To find all available shells in your system you can use the cat command. Type the command as shown in figure 6.2 on the command prompt. You will get list of all the available shells in your computer system. The list of shell that you get as an output may be different from the one shown in figure 6.2 depending on your system configurations. The cat command is discussed later in this chapter.



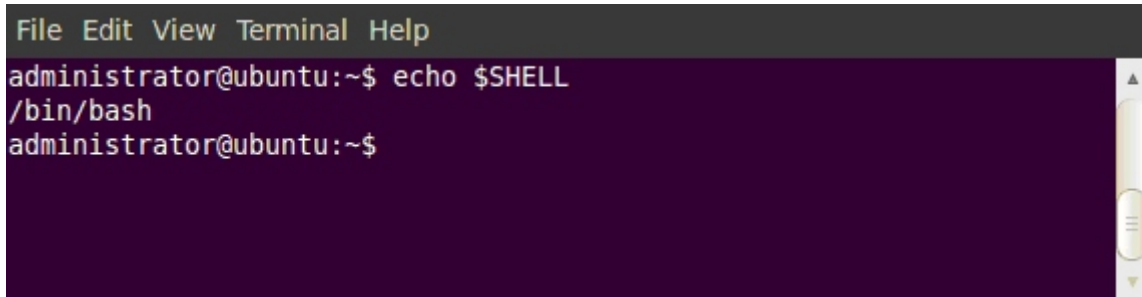
```
File Edit View Terminal Help
administrator@ubuntu:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/esh
/bin/dash
/bin/bash
/bin/rbash
/usr/bin/screen
administrator@ubuntu:~$
```

Figure 6.2 : Different available shells

Determining the default shell

Each Ubuntu Linux account is configured with a certain shell as its default command line interface. Each time you log on, this default shell is utilized for working within the system. Linux operating system comes with certain variables that contain current environment settings as its values and thus are known as environment variables.

The value of default shell is stored in one such environment variable named SHELL. Thus by displaying the value of the variable SHELL, we can come to know which our default shell is. To display the value of any variable, *echo* command can be used. Type `echo $SHELL` on the command prompt and press Enter key. Default shell will be displayed on the screen as shown in figure 6.3. Note that Linux commands are case sensitive hence SHELL, Shell and shell are not same.

A screenshot of a terminal window with a dark background. The window title bar shows 'File Edit View Terminal Help'. The terminal content shows the prompt 'administrator@ubuntu:~\$' followed by the command 'echo \$SHELL'. The output is '/bin/bash', and the prompt returns to 'administrator@ubuntu:~\$'.

```
File Edit View Terminal Help
administrator@ubuntu:~$ echo $SHELL
/bin/bash
administrator@ubuntu:~$
```

Figure 6.3 : Default Shell

Changing the current shell

As discussed earlier we have different shells available with Linux. To change your default shell, type the name of the shell you want to use on the command line. For example, if you want to use the C shell (provided it is available on the system), type `cs` at a command prompt. Then the command prompt will provide a `cs` interface.

Note :

A shell change is temporary and will last only as long as you are logged on that command line.

To return to default shell, type `exit` or press `CTRL + d` at the command prompt of the new shell.

Command Syntax

The syntax of Linux commands is uniform. It consists of three parts, in the order specified below :

- **Name :** It is the name of the command, for example `ls`, `echo` etc.
- **Options :** It is possible to alter the behavior of the commands by specifying additional options. A command may have zero or more options. Options when present starts with a hyphen symbol (-) and are usually a single letter or a digit. Some commands may have options with double hyphen and/or sequence of letters or digits. Depending on the command, the number and meaning of the options will vary.
- **Arguments :** Along with options user can also provide arguments. A command may take zero or more arguments to do its work. The number and expected meaning of the arguments vary from command to command. Some commands may take no arguments; others may take an exact number, while other commands may take any number of arguments.

Linux commands can be classified as internal or external based on whether its binary file exists or not. The commands that have a binary file explicitly stored in either `/sbin`, `/usr/sbin`, `/usr/bin`, `/bin`, or `/usr/local/bin` directories are called external commands. They are generally executed by the kernel and will generate a process id at the time of execution. Most of the commands that we use in

Linux are external commands. On the other hand the commands directly executed by the shell are called internal commands. Internal commands do not generate a new process.

To know whether a command is internal or external we can use the *type* command. The syntax of type command is shown below :

Stype command

For example if we execute a command

Stype info

we will get the output as shown below :

info is /usr/bin/info

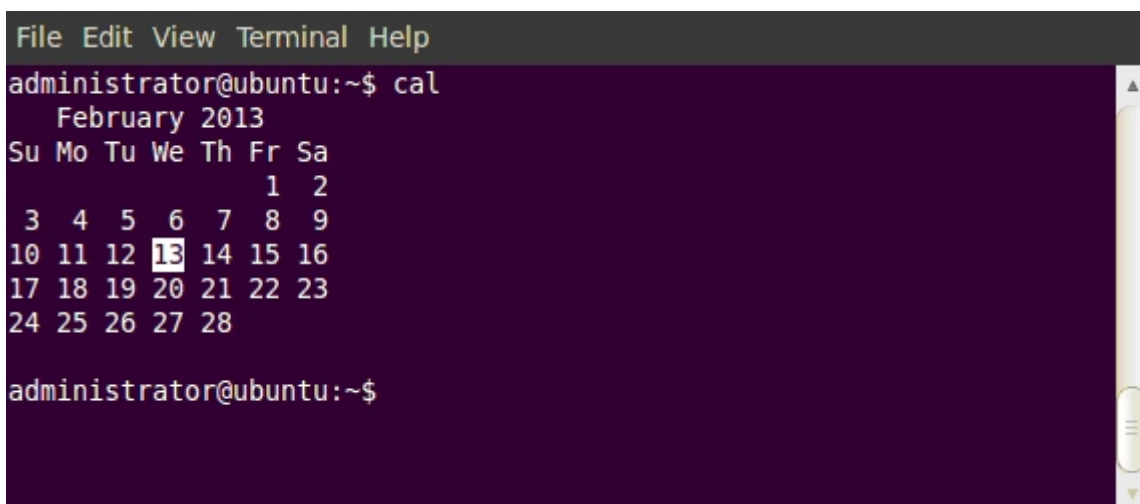
As can be observed this refers to a binary file *info* stored in */usr/bin/*, this indicates that *info* is an external command.

Issuing General Purpose Commands

As you are now familiar with the syntax of commands, let us see how to issue commands through command line interface. The best way to start with learning Linux commands is to try working with some general purpose Linux commands. You can issue a command by typing a command name followed by necessary options and arguments. Other way is you can type in first few letters of a command, press the tab key and the shell automatically provides the remaining information. For example, to display a calendar, type *ca* on the prompt and press tab key. Linux shell will automatically display list of all commands starting with alphabets *ca* including the calendar command *cal* on the screen. If you get more than one command in the list, then type the desired command on the prompt and press Enter key to execute the command.

Calendar (*cal*)

The *cal* command is used to display a calendar of any specific month or entire year. The default output of *cal* command is calendar of the current month. See figure 6.4.



```
File Edit View Terminal Help
administrator@ubuntu:~$ cal
  February 2013
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

administrator@ubuntu:~$
```

Figure 6.4 : Output of cal command

We can change the calendar as per our requirement, for example to display the calendar of January, 2013 type the following command on the prompt and press Enter key.

Scal 01 2013

The output of this command will be similar to the one shown in figure 6.4 except that the month would be January 2013. Similarly if we want to display the calendar of the entire year 2013, we simply have to type `cal` followed by the year as shown below :

Scal 2013

Note that the calendar of entire year may not be displayed on the entire monitor screen; hence we will have to use a pipe operator as shown below :

Scal 2013 | more

In the above command we have concatenated two commands, here *more* is also a command which takes input from the *cal* command. The pipe (|) symbol used in between the two commands is discussed in detail later in this chapter.

Date (date)

Another utility command is *date*; it is used to display the system date.

Sdate

The output of the command is shown in figure 6.5.

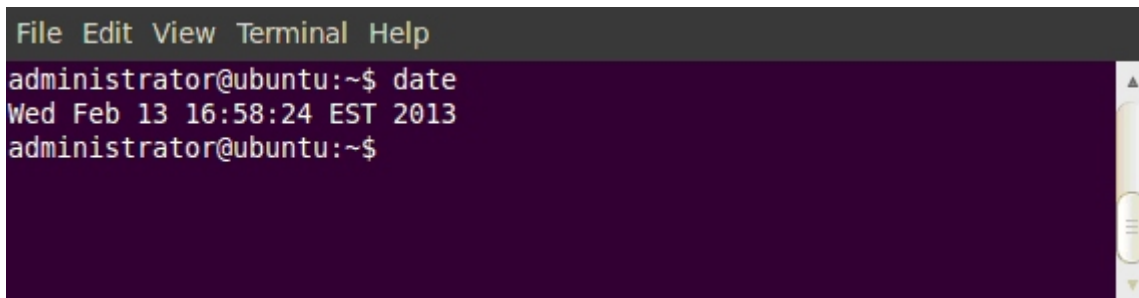
A terminal window with a dark purple background and a grey title bar. The title bar contains the text "File Edit View Terminal Help". The terminal shows the prompt "administrator@ubuntu:~\$" followed by the command "date". The output is "Wed Feb 13 16:58:24 EST 2013". The prompt "administrator@ubuntu:~\$" is shown again on the next line. On the right side of the terminal window, there are three vertical buttons: a triangle pointing up, a hamburger menu icon, and a triangle pointing down.

Figure 6.5 : Output of date command

Observe that the output displays both date as well as time. The date command can also be used with suitable format specification as arguments. Each format is preceded by + symbol, followed by % operator and a single character describing the format. For example, to display only the current date in mm/dd/yy format use the command shown below :

Sdate +%D

Figure 6.6 shows the output of the command.

```
File Edit View Terminal Help
administrator@ubuntu:~$ date +%D
02/13/13
administrator@ubuntu:~$
```

Figure 6.6 : Output of formatted date command

The command line calculator (*bc*)

The *bc* command in Linux is a command line calculator. In addition to performing simple mathematical functions, it can also perform conversions between different number systems, as well as allows us to use some scientific functions. To work with this command use the syntax given below :

Sbc -l

A screen similar to the one shown in figure 6.7 will be displayed. Notice that the dollar prompt is not visible on the screen; this indicates that the *bc* command is now ready to take input from you. The *-l* switch is used to include the standard math library.

```
File Edit View Terminal Help
administrator@ubuntu:~$ bc -l
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Found
ation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
```

Figure 6.7 : Initiation of bc command

Now, just type the formula that you want to evaluate at the blinking cursor and the press Enter key. You can type a simple expression like $5 * 5$, or you may type a complex expression with grouped operators. Let us type the expression $((99.1 / 5.15) * 99.9)$, now press the Enter key. The command will display the output in the next line as shown in figure 6.8.

```
File Edit View Terminal Help
administrator@ubuntu:~$ bc -l
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Found
ation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
((99.1/5/15)*99.9)
132.0011999999999999999966
```

Figure 6.8 : Output of bc command

Observe that we still are not able to see the prompt; this simply means that we can continue working in command line calculator mode.

In addition to the normal mathematical functions like addition, subtraction, multiplication, division, modulus, and exponents, we can also use trigonometric or logarithmic functions like sine, cosine, arctangent, and log. For example, if you need to find the natural logarithm of value 2013, use the command **l(2013)** and you will get output **7.60738142563979148420**.

The *ibase* function allows us to set the numbering system that we want to use for input. Similarly the *obase* function allows us to set what numbering system to use for output. Let us try to convert numbers from decimal number system to hexadecimal number system. First, we need to set *obase* as shown below :

```
obase = 16
```

Now, type the number you want to convert to hexadecimal as shown in the example and press the Enter key.

```
256
```

```
100
```

Here 100 is the hexadecimal equivalent of decimal number 256.

Similarly if you want convert this result to binary number system then just change the *obase* again as shown here.

```
obase=2
```

Now type **100** and the output will be 1100100, observe that this is binary equivalent of decimal 100 and not 256. The reason for this is very simple, we have not changed *ibase*, and hence all entries are considered to be decimal entries.

To convert hexadecimal value 100 to its binary equivalent, set `ibase` and `obase` as shown below :

```
ibase=16
```

```
obase=2
```

Now type a hexadecimal number that you need to convert to binary. For example, type **100** and you will get **100000000** as a result.

To return back to decimal mode set `ibase` to 10. Execute the following to find out the square root of a number using `sqrt` function available in `math` library.

```
sqrt(256)
```

```
16.000000000000000000000000
```

As compared to graphical calculator the command line `bc` calculator is more faster and flexible. To return to the Linux command prompt, press `CTRL+ d`.

Displaying a message (*echo*)

We need to display message very frequently when using command prompt. The *echo* command is used to display a message on the terminal. For example, type the following command and press the Enter key. The string written after `echo` will be displayed on your monitor screen.

```
Secho Hi, I am learning Ubuntu Linux
```

```
Hi, I am learning Ubuntu Linux
```

It is also possible to enclose the string within double quotes. The output will not contain the double quotes. The `echo` command can also be used to display values of variable. For example, define a variable named `cost` and assign it value 10 as shown below :

```
Scost=10
```

Once you press the Enter key you will be returned to prompt. Now type the command given below :

```
Secho The cost of product is Rs. Scost
```

```
The cost of product is Rs. 10
```

To display the value of `cost` on the screen it is passed to the `echo` command. Notice that in the string we have written `cost` twice. The one that is prefixed with `$` symbol represents a variable, while the other is a normal string. When the `echo` command finds any string prefixed with a `$` symbol will consider it to be a variable. It will then try to print the value of variable.

The echo command can be used along with other commands to give meaningful output. For example we may combine the echo and the date command in the manner shown below :

```
Secho Current time is ' date +%T '
```

```
Current time is 14 :55 :04
```

Observe that the command to be executed is placed within back quotes (quotes available on key with ~ sign), thus first the date command will be executed and then its result will be displayed using echo command.

Changing password (*passwd*)

A user needs to change password very often due to various reasons. The *passwd* command helps us perform this operation. It is used to change the password of the current login account by default. The following command allows you to change the password.

Spasswd

Once you press the Enter key a message similar to the one below will be shown, along with the blinking cursor.

Changing password for administrator

```
(current) UNIX password :
```

Type your current password and press the Enter key, Linux will check whether you have entered a valid password, and if you have then it will prompt for the new password. You will be asked to enter new password and retype it again as shown below :

```
Enter new UNIX password :
```

```
Retype new UNIX password :
```

```
passwd : password updated successfully
```

If you have typed the new password correctly and it does not conflict with any guidelines decided for password, your new password will be registered by the system. In case of any problem you may get an error message. It is also possible to change the password of other user of the system by specifying the username after the passwd command. For example, if we have a user named harshal, to change its password we may type the following command.

Spasswd harshal

In case of genuine user you will be allowed to change its password.

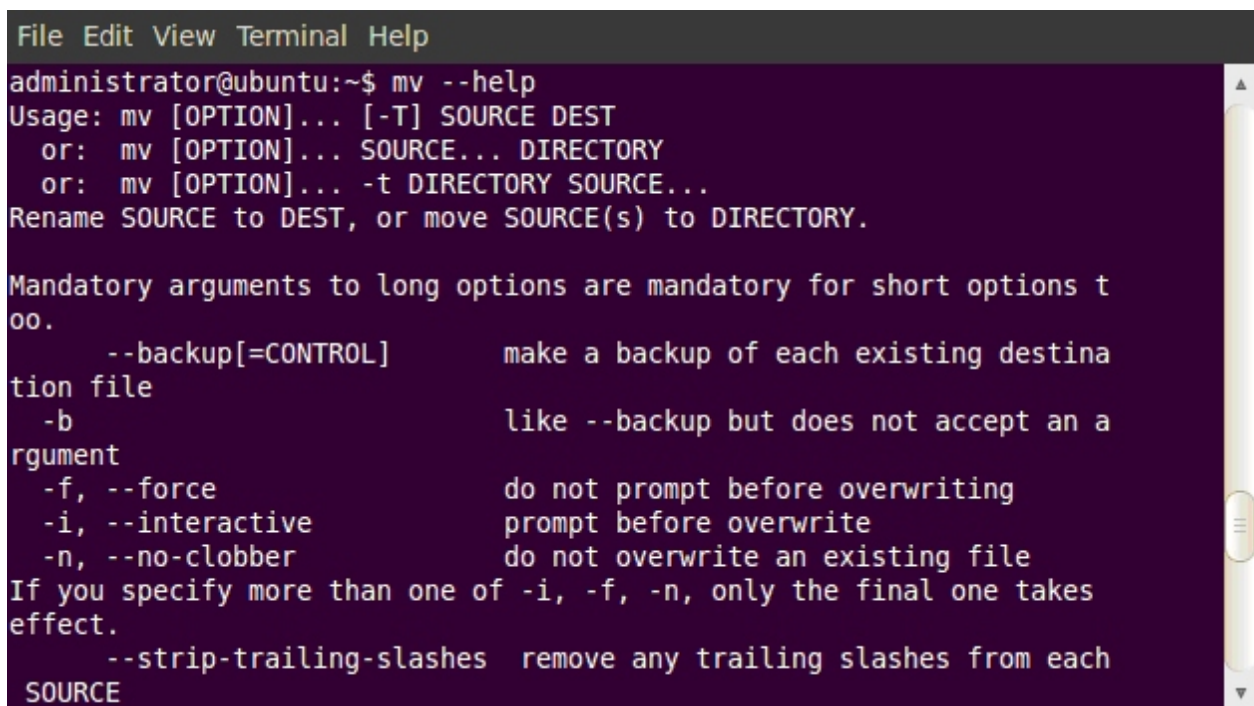
Clearing the Screen (*clear*)

While working on command prompt you must have observed that the screen often gets full. At times it also becomes difficult to see the output clearly, we have one simple solution for this problem, use the *clear* command to remove data on the screen.

Sclear

Getting Help on the Linux Commands

Before looking at any other commands first let us learn how to get help when using commands on Linux platform. Linux provides two inbuilt commands namely *help* and *man* to assist the user while working on the command line interface. All the commands that we use in Linux supports the -h (or -help) option. This option generates a small description of how to use the command. Figure 6.9 shows the use of help command.



```
File Edit View Terminal Help
administrator@ubuntu:~$ mv --help
Usage: mv [OPTION]... [-T] SOURCE DEST
  or: mv [OPTION]... SOURCE... DIRECTORY
  or: mv [OPTION]... -t DIRECTORY SOURCE...
Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

Mandatory arguments to long options are mandatory for short options too.
  --backup[=CONTROL]  make a backup of each existing destination file
  -b                  like --backup but does not accept an argument
  -f, --force         do not prompt before overwriting
  -i, --interactive   prompt before overwrite
  -n, --no-clobber    do not overwrite an existing file
If you specify more than one of -i, -f, -n, only the final one takes effect.
  --strip-trailing-slashes  remove any trailing slashes from each SOURCE
```

Figure 6.9 : Use of help command

Observe that the command used to display the help in figure 6.9 is *mv - - help*. The alternate mechanism to get help on the command is to use Linux online manuals. The *man* command activates a manual corresponding to a specific command that we need to look at. For example, the command *man mv* will show us the manual for the move command. Figure 6.10 shows the output of *man* command.

Observe that *man* command gives us exhaustive information of a command and may run into multiple screens. It generally displays one page at a time, as we press the Enter key the contents scroll down. To come out of the manual screen type alphabet 'q', this will take you back to the command prompt.

```
File Edit View Terminal Help
MV(1)                                User Commands                                MV(1)
NAME
    mv - move (rename) files
SYNOPSIS
    mv [OPTION]... [-T] SOURCE DEST
    mv [OPTION]... SOURCE... DIRECTORY
    mv [OPTION]... -t DIRECTORY SOURCE...
DESCRIPTION
    Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

    Mandatory arguments to long options are mandatory for short options
    too.

    --backup[=CONTROL]
                make a backup of each existing destination file
Manual page mv(1) line 1
```

Figure 6.10 : Use of man command

In case we want only small description of a command then we may use the *whatis* command. It gives us one line explanation of the command, but omits any additional information about options. Figure 6.11 shows the sample output of *whatis* command.

```
File Edit View Terminal Help
administrator@ubuntu:~$ whatis mv
mv (1) - move (rename) files
administrator@ubuntu:~$
```

Figure 6.11 : Sample output of *whatis* command

Many times it may happen that we may not know which command to look for exactly. In such situations we may use the *apropos* command. The syntax of the command is mentioned below :

Sapropos string

When we execute this command we will get a list of all the commands that has the string as the part of the command or command description. For example if we type *apropos copy* on the command prompt and try to execute it, then we may get screen full of commands that have copy as a string within the command or its description. A user must take caution while using this command. Sample output of *apropos* command is shown in figure 6.12.

```
File Edit View Terminal Help
administrator@ubuntu:~$ apropos copy
bcopy (3)          - copy byte sequence
bf_copy (1)       - shell script to copy a bogofilter working...
bf_copy-bdb (1)   - shell script to copy a bogofilter working...
copysign (3)      - copy sign of a number
copysignf (3)    - copy sign of a number
copysignl (3)    - copy sign of a number
cp (1)           - copy files and directories
cpgr (8)         - copy with locking the given file to the p...
cpio (1)         - copy files to and from archives
cppw (8)         - copy with locking the given file to the p...
dd (1)          - convert and copy a file
```

Figure 6.12 : Sample output of apropos command

Working with Directories

In Linux a directory is a special type of file that is used to store files and other directories. Here ‘ / ’ symbol represents the root directory. All other directories come under root directory. Let us learn how to work with directories using terminal window.

Home directory

When a user logs on to the system, Linux automatically places the user in the directory called the home directory. It is created by the system at the time when a user account is created and generally will have a path /home/username. Here the username refers to the login name. For example if the username is harshal, then the home directory will be /home/harshal. It is possible to change this path if needed. The default working directory path is stored in system variable named HOME. We can cross check the directory by using the echo command.

Secho \$HOME

/home/harshal

Note that the path displayed using this command is an absolute path name. Absolute path name is a sequence of directory names separated by / (slashes). An absolute path name shows a location in reference to the root directory. The first slash (/) is synonymous to root directory while the other slashes act as delimiters to other directory names. Thus the directory harshal is located within directory home which further is located in the root directory. Similarly the directory **/home/administrator** refers to home directory of username administrator.

It is possible to change the default path of the home directory. Suppose that you are able to see the output as shown below :

/home/its/ug1/svics

Here *svics* (home directory of user *svics*) is within the sub directory *ug1* (a directory that represents a sub group), which further is within a directory *its* (a directory that represents a group). Directory home and ‘/’ have their default meaning.

Present Working Directory (*pwd*)

After we log into a system we can move around from one directory to another. But at any given point of time we will be located only in one directory. The directory where we are located at that moment is known as *current directory* or *present working directory*. To know the current directory that we are working in we can use the *pwd* command.

\$pwd

Creating a Directory (*mkdir*)

A directory in Linux can be created using the *mkdir* command. The command takes the name of the directory to be created as its argument. Let us create a directory named *subject*.

\$mkdir subject

The power of command line over GUI lies in its flexibility. If we create a directory using GUI we will be able to create one directory at a time, while it is possible to create multiple directories using a single *mkdir* command. The following command syntax illustrates the same.

\$mkdir animals birds vehicles plants

The command when executed will create four directories named *animals*, *birds*, *vehicles* and *plants* in the current directory.

Change Directory (*cd*)

In the case when we need to store any data within a directory, first we need to make it our current directory. We can change (go within) a specific directory using the *cd* command. Let us try to create a directory named *math*, *science* and *economics* within a directory *subject*. To create these directories first we need to be in the *subject* directory. The command sequence shown below allows us to perform the said operation.

\$cd subject

\$pwd

/home/administrator/subject

\$mkdir math science economics

Observe that the user name here is *administrator*. To again come back to the *administrator* directory, simply type the command below :

\$cd ..

In the above command double dots (..) refer to the parent directory. Note that there should be one space between the cd command and the double dot.

Assume that you are in some internal directory that has path /home/administrator/subject/economics and you need to come back to the users home directory then again the cd command comes in handy. To come back to home directory we can issue the cd command twice as shown below :

Scd ..

Scd ..

The command sequence here will take us back one level at time. If we are say *M* levels down within the home directory, then we will have to execute the cd command *M* times. An alternative approach is to use a single cd command as shown below :

Scd ../../

Some example usage of the cd command along with its description is given in table 6.1.

Command Issued	Action Performed
cd ~/Desktop	Changes directory to /home/username/Desktop, from any current path. Here the symbol ~ refers to home directory of the user.
cd /	Changes directory to the root directory from any current path.
cd	Changes directory to the home directory from any current path.
cd -	Changes directory to the previously changed directory.
cd /var/www	Changes directory directly to the www sub-directory with directory var. It is useful when we know the path explicitly.

Table 6.1 : Sample cd commands

Remove Directory (*rmdir*)

An empty directory can be deleted by using the command *rmdir*.

Scrmdir science

Here science is a directory name, and it will be removed using the above command only if it is empty. In case it is not empty we will get a message ‘rmdir : failed to remove ‘science’ : Directory not empty’. In this case we have to first delete all the contents within the directory and then reissue this command. Note that it is also possible to delete multiple empty directories in the same way we created them.

To delete a non-empty directory with all its contents we can use the *rm* command as shown below :

```
Srm -r science
```

The *rm* command is discussed in detail in the later part of chapter.

Naming Conventions in Linux

We have created directories with different names using the *mkdir* command. While creating any object like a directory or a file in Linux we have to follow certain rules. On most of the Linux systems today, a name (of directory or file) can consist of up to 255 characters. Unlike Windows OS, names in Linux can practically consist of any ASCII character except for the slash (/) and the NULL character. Any other control characters or nonprintable characters are permitted. Examples of some valid names are *.name*, *^myname^-++*, *-{ }()*, *test\$#*, *xy.ab.ef* etc.

However, it is recommended that a name should be relevant and contain only alphabetic characters, numerals, period (.), hyphen (-) and underscore (_). Linux strictly adheres to case sensitivity, thus *math*, *Math* and *MATH* are three different names. It is possible for the directories with these names to coexist at same level. If these names refer to a file then they can again coexist in the same directory.

Working with Files

Directory generally works as a container. The data is normally stored in files, which further may be stored in directory for proper arrangement and easy access. Text editors like *nano*, *pico*, *vi* or *vim*, *ed* and others are generally best suited for creating text file. However, many times the user needs to create a file quickly. The *cat* command comes in handy in such scenario. It is mainly used to display the contents of a small file on the terminal. But it can also be used to create a file, concatenate two files and append contents into a file.

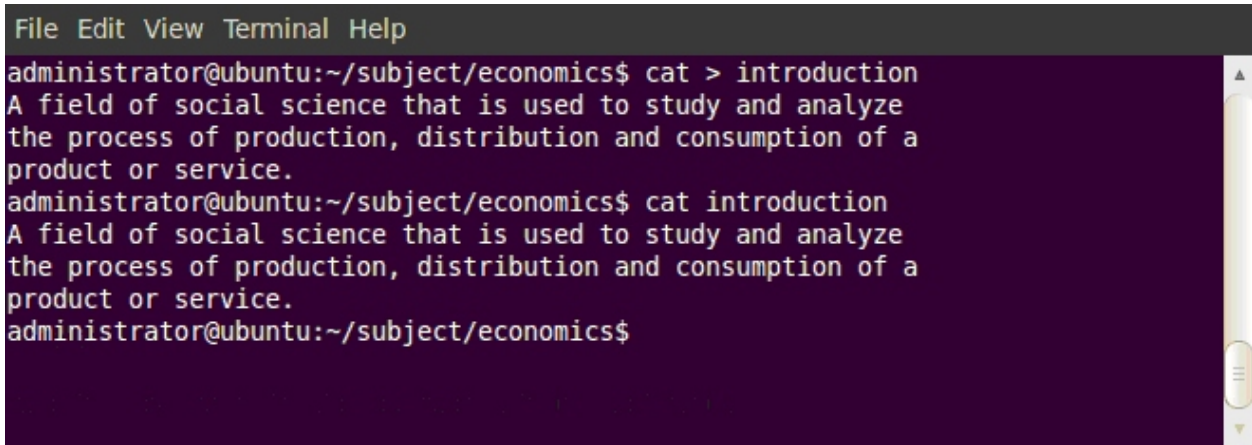
Create a file using *cat* command

Let us create a file named *introduction* within the directory *economics*. To create this file you will have to first make the directory *economics* as your current directory. You can use the *cd* command here. Now type the command *cat* followed by a greater than (>) symbol and name of the file as shown below :

```
Scat > introduction
```

When we execute this command the cursor will be positioned in the next line, waiting for us to type the contents of the file. Type the text that you want to store in file and press CTRL + d. This will take you back to the command prompt. The combination CTRL + d in Linux indicate the end of file character. The greater than (>) symbol used in the above command is known as

redirection operator. It is used to instruct the shell that a redirection is required i.e., input should go to the specified file. The cat command when used without the greater than (>) symbol, displays the contents of the filename specified in argument. Figure 6.13 shows the process of creating and then displaying a file.

A screenshot of a terminal window with a dark purple background. The terminal shows a user at the prompt 'administrator@ubuntu:~/subject/economics\$' typing 'cat > introduction'. The output shows the text 'A field of social science that is used to study and analyze the process of production, distribution and consumption of a product or service.' followed by a prompt. The user then types 'cat introduction' and the same text is displayed again. The terminal window has a menu bar at the top with 'File Edit View Terminal Help' and a scrollbar on the right side.

```
File Edit View Terminal Help
administrator@ubuntu:~/subject/economics$ cat > introduction
A field of social science that is used to study and analyze
the process of production, distribution and consumption of a
product or service.
administrator@ubuntu:~/subject/economics$ cat introduction
A field of social science that is used to study and analyze
the process of production, distribution and consumption of a
product or service.
administrator@ubuntu:~/subject/economics$
```

Figure 6.13 : Creating and displaying file using cat command

Appending contents using cat command

Assume that you have an existing file and you want to add some more content in the file. The cat command can be used again here with one simple change. The redirection operator used previously is to be replaced by append output (>>) redirection operator. The command to append data in the *introduction* file is shown below :

\$cat >> introduction

An alternate definition states that Economics is a science which studies human behaviour as a relationship between ends and scarce means which have alternative uses.

[CTRL+d]

Note that if the file already exists and we use the command *cat > filename* then, the existing contents will be overwritten with the new one. So it is necessary to be careful while opening a file that already has some contents.

Concatenating multiple files using cat command

The cat command can also be used to concatenate the contents of multiple files and store it in another file. The syntax of using the concatenation is shown below :

\$cat file1 file2 > file3

The above command will create *file3* that contains the text of both the files, namely file1 and file2. The new file created will have contents based on the sequence of filenames. Here the initial contents will be of file1, after which the contents of file2 will be appended.

Deleting a File (*rm*)

The *rm* command is used to delete/remove one or more files. For example to delete the file *introduction*, execute the following command :

```
$rm introduction
```

We can also delete multiple files using a single *rm* command. For instance the command

```
$rm file1 file2 file3
```

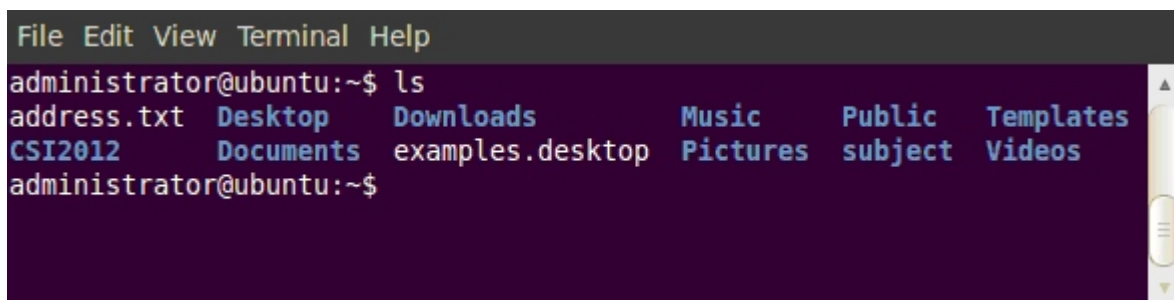
will delete all the three files supplied to it as argument. Table 6.2 gives some options that can be used along with the *rm* command.

Option	Usage
<code>rm -i Filename</code>	Deletes file using interactive mode. The user will be asked to verify the delete operation.
<code>rm -r Directoryname</code>	Deletes directory and along with all its contents.
<code>rm -r *</code>	Deletes all contents (file and/or directory) within the current directory. Here the symbol <code>*</code> is known as wildcard character. This is a very dangerous command as it will delete all the files and directories within the current directory, hence such command should be used only if you certain about the action and the result.
<code>rm -rf *</code>	Same as <code>rm -r</code> , but also deletes the contents even if it is write protected.

Table 6.2: Some options of *rm* command

Moving around the File system

So far we have learned how to create and delete a directory or file. Now let us see how to view the contents that are part of our file system. The *ls* command gives us the list of the contents in the current or a specified directory. The *ls* command can be used with different options to change the output. Let us begin with the plain and simple *ls* command without any options. Figure 6.14 shows the output of simple *ls* command. Note that the output on your computer may vary.



```
File Edit View Terminal Help
administrator@ubuntu:~$ ls
address.txt Desktop Downloads Music Public Templates
CSI2012 Documents examples.desktop Pictures subject Videos
administrator@ubuntu:~$
```

Figure 6.14 : Output of the *ls* command

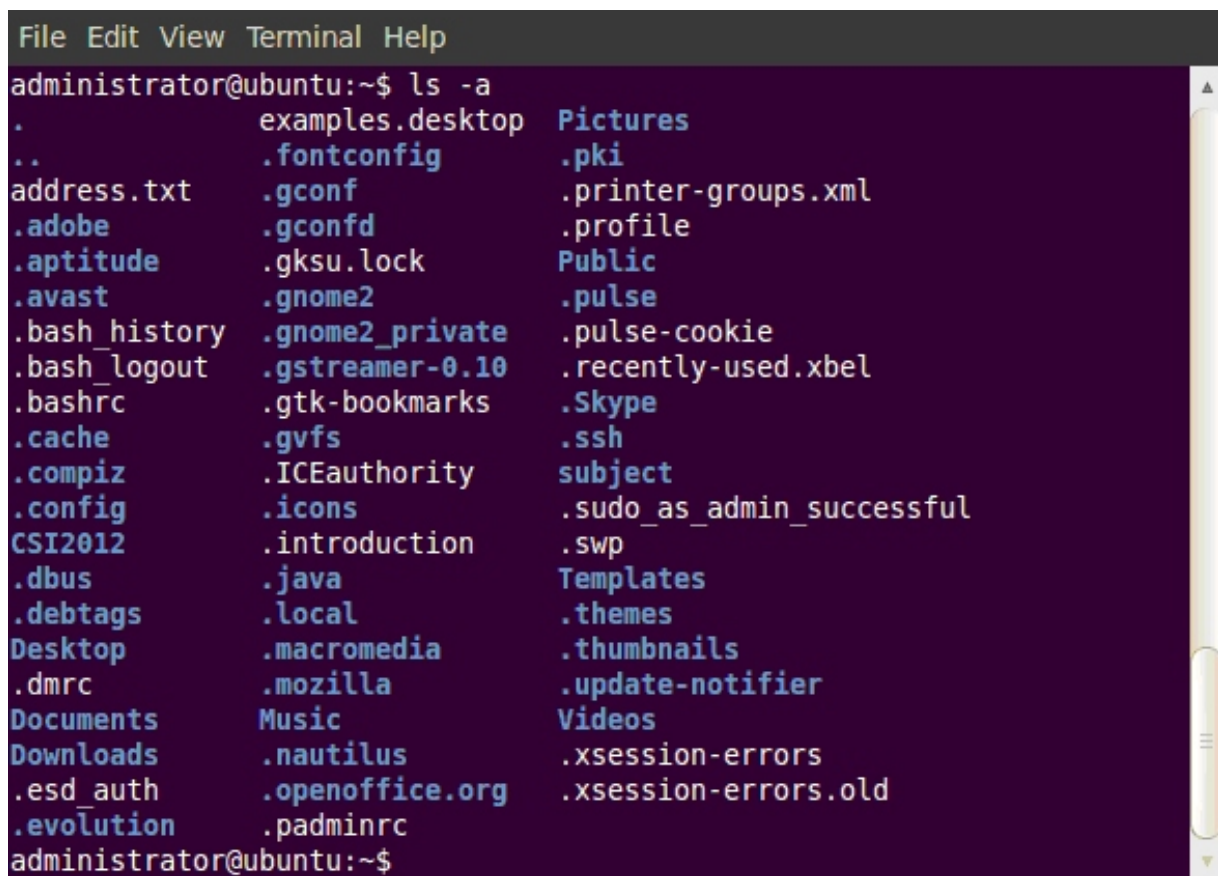
Let us now create a file named *.introduction* and then check whether we are able to list it or not. Type the command shown below to create the file.

Scat > .introduction

Learning Ubuntu Linux is fun....

[CTRL + d]

Now use the `ls` command again to list the contents of a file. You must have observed that the file recently created is not visible on the screen. Note that in Linux any filename that is preceded by a `.'` is treated as a hidden file. To list hidden files in the current directory we need to use `-a` option of the `ls` command. Figure 6.15 shows how to list hidden files.



```
File Edit View Terminal Help
administrator@ubuntu:~$ ls -a
.          examples.desktop  Pictures
..         .fontconfig       .pki
address.txt .gconf            .printer-groups.xml
.adobe     .gconfd           .profile
.apptitude .gksu.lock        Public
.avast     .gnome2           .pulse
.bash_history .gnome2_private  .pulse-cookie
.bash_logout .gstreamer-0.10  .recently-used.xbel
.bashrc     .gtk-bookmarks   .Skype
.cache     .gvfs            .ssh
.compiz    .ICEauthority    subject
.config    .icons           .sudo_as_admin_successful
CSI2012    .introduction    .swp
.dbus      .java            Templates
.debtags   .local           .themes
Desktop    .macromedia      .thumbnails
.dmrc      .mozilla         .update-notifier
Documents  Music            Videos
Downloads  .nautilus        .xsession-errors
.esd_auth  .openoffice.org  .xsession-errors.old
.evolution .padminrc
administrator@ubuntu:~$
```

Figure 6.15: Listing hidden files

Observe the difference in the output of figure 6.14 and figure 6.15. Note that figure 6.15 shows more number of files than shown in figure 6.14. The two entries `.'` and `..` visible in figure 6.15 are of importance, these two entries are automatically created in the directory whenever the directory is created. Table 6.3 gives some options that can be used along with the `ls` command.

Option	Usage
ls ~	Lists the files that are in user's home directory.
ls [svics]*	Lists all the files in which the first character of the filename matches with any of the given alphabets within the square brackets. The remaining part of filename can contain any valid ASCII character.
ls [n-s][5-7]??	Lists all files with 4 character filename. With the condition that the first character is in the range n to s, second character is in the range of 5 to 7, whereas the third and fourth characters are any valid ASCII character.
ls -r	List the files by sorting them in reverse order.
ls -t	List the files by sorting them based on their modification time.
ls -F	List the files and mark all executable files with * and directories with / symbol.
ls -l	List one file per line.

Table 6.3: Some options of ls command

Pattern Matching – The wildcards

In the above discussion you have already seen the usage of characters asterisk (*) and question mark (?). These characters are known as wildcard characters used for matching a pattern as required by the user. Table 6.4 summarizes the working of wildcards used by shell.

Wildcard	Pattern to be matched
*	Any number of characters including none
?	A single character
[abc]	A single character – either a, b or c (user can use other characters also).
[!abc]	A single character <i>other than</i> a, b or c (user can use other characters also).
[p-s]	A single character within the ASCII range of the characters p to s (user can use other characters also).
[!p-s]	A single character that is not within the ASCII range of the characters p to s (user can use other characters also).

Table 6.4: The wildcard characters

Manipulating Files and Directories

In the previous section we learned how to create and delete a file or a directory. Let us now see how to perform operations like copy, move, and assign permission on them.

Copying a file (*cp*)

Very often we need to create a replica of the data that we have generated, the *cp* command copies a file or group of files specified as an argument to it. It creates an exact replica of a file on the disk at the location specified by the user. The *cp* command needs at least two arguments. The first argument refers to a source file while the second argument refers to a destination file. Let us create a copy of file *introduction* using the following command :

```
Scp introduction new_introduction
```

After execution of the above command, an exact copy of file *introduction* will be created with the name *new_introduction*. If a file with the name *new_introduction* already exists, it will simply be overwritten without any warning from the system. In case no such file exists a new file will be first created and then the contents of the file *introduction* will be copied in it.

The *cp* command can also be used to copy more than one file into a specified directory. For instance, the following command :

```
Scp file1 file2 my_dir
```

will copy two files named *file1* and *file2* in a directory named *my_dir*. It is necessary that the directory *my_dir* already exists, or else we will get an error message. Table 6.5 gives some example usage of the *cp* command.

Command	Description
<code>cp /vol/examples/tutorial/science.txt .</code>	Copies the file <i>science.txt</i> to current directory. The dot (.) at the end refers to the current directory.
<code>cp chap01 progs/unit1</code>	A file named chap01 is copied within the directory progs with name unit1. (no directory with name unit1 should exist in progs directory)
<code>cp chap01 progs</code>	A file named chap01 is copied within the directory progs with same name. (because progs is a directory).
<code>cp -r progs newprogs</code>	The directory named progs along with all its contents is copied and stored as a directory newprogs.

Table 6.5 : Sample cp commands

Renaming files and/or moving files (*mv*)

Changing the name of a file or directory is another operation that user performs regularly. The *mv* command is used for renaming a file or directory. For example, to rename the file *introduction* to *introduction.txt*, execute the following command :

\$mv introduction introduction.txt

The command will rename the file and store it at the same location as that of the file *introduction*. Thus no additional space is consumed on disk during renaming.

The *mv* command can also be used to move a file or group of files to a different directory. For example, the command

\$mv file1 file2 my_dir

will move the files named *file1* and *file2* to the directory named *my_dir*.

This command is also used to rename a directory. For instance, the execution of the following command :

\$mv math mathematics

will rename an existing directory *math* to *mathematics*.

Paging output (*more*)

The *more* command is used to view one page of content on the screen at a time. For instance if the file *introduction.txt* contains text that can not fit in a single screen, then reading its contents would become difficult. The *more* command when used displays the contents of file one page at a time. To view the next page we may press any key. Generally we may press character 'b' to view previous page and character 'f' to view next page. The sample usage of the command is shown below :

\$more introduction.txt

Compare two files (*cmp*)

The *cmp* command compares two files of any type and writes the results to the standard output. If the two files compared differ in contents then the byte and line number at which the first dissimilarity occurred is reported. In case there is no difference between the contents of the files we simply see the command prompt again. The sample usage of the command is shown below :

\$cmp introduction introduction.txt

Difference (*Diff*)

An extension of the *cmp* command is the *diff* command. The *diff* command compares two files and displays the contents of both files indicating where the difference lies. To understand the working of the *diff* command we have created a copy of the file *introduction.txt* and named it *new_intro.txt*. We have also removed some lines from the new file. Figure 6.16 shows the output of the command shown below :

\$diff introduction.txt new_intro.txt

```
File Edit View Terminal Help
administrator@ubuntu:~/subject/economics$ diff introduction.txt new_intro.txt
3d2
< product or service.
administrator@ubuntu:~/subject/economics$
```

Figure 6.16 : Output of diff command

In figure 6.16 the lines beginning with a < indicates that file introduction.txt contains the text shown but file new_intro.txt does not contain the line. Any changes in the file new_intro.txt would be shown with the lines beginning with a > sign.

Counting lines, words and characters in a file (wc)

The *wc* command is used to count the number of lines, words, and characters in the specified file or files. The *wc* command can be used along with three options -l, -w and -c for counting lines, words and characters respectively. For instance, execute the following command to count the number of lines in the file introduction.txt.

```
$wc -l introduction.txt
```

```
4 introduction.txt
```

Similarly the commands **wc -w introduction.txt** and **wc -c introduction.txt** will give us the count of number of words and characters in the file. To get all the information together we can use the command as shown below :

```
$wc -l -w -c introduction.txt
```

```
4 49 307 introduction.txt
```

File permissions

In the earlier section, we have seen options which can be used with ls command. The ls command has several other options also. For example the following command when executed may result in the output similar to the one shown here.

```
$ls -l
```

```
total 6
```

```
-rw-r--r-- 1 administrator administrator 313 2013-02-15 18 :04 about_Gandhiji.txt
```

```
-rw-r--r-- 1 administrator administrator 444 2013-02-15 18 :19 introduction.txt
```

```
-rw-r--r-- 1 administrator administrator 401 2013-02-20 16 :43 address.txt
```

```
drwxr-xr-x 1 administrator administrator 4096 2013-02-21 18 :15 backup
```

```
-rw-r--r-- 1 administrator administrator 144 2013-02-13 18 :49 city.txt
```

```
-rw-r--r-- 1 administrator administrator 226 2013-02-20 14 :11 script10.sh
```

Observe the output; it gives us a clear idea about an object in our file system. An object can be categorized as a regular file, a directory or a process. It shows us the owner of the object, size of the object, date and time on which the object was created along with the name of the object. Let us try to understand the file permission in detail. Figure 6.17 shows the relation of different permissions w.r.t. owner, group and other users.

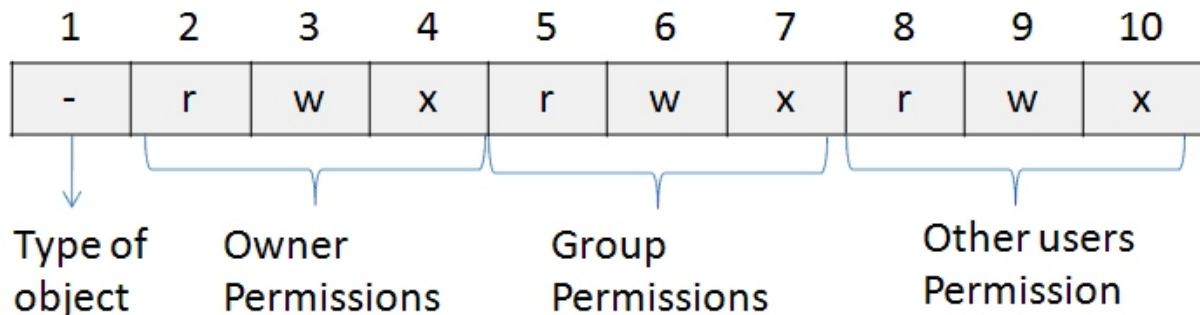


Figure 6.17 : File permissions

As can be observed in figure 6.17 the first column refers to type of object. The character ‘-’ in the first column refers to a file, character ‘d’ refers to a directory and character ‘p’ refers to a process. The next nine characters tell the system what access is permitted for this object; hence the name “permissions”. An object in Linux has three permissions namely, read (r), write (w) and execute (x).

The set of three characters after the file type shown in column 2 to 4 tells which permission the owner of the file has. Owner of the file is the user who created the file (administrator in our case). The character **r** in the first position means you are permitted to read the file. A **w** in the second position means you may write in the file. This includes the ability to delete a file. An **x** in the third position means you may execute the file. A hyphen ‘-’ in any position means that you don’t have that particular permission. As you can in the output of the `ls -l` command, administrator the user who owns the file can read and write files like `about_Gandhiji.txt`, `introduction.txt` and `address.txt`. The characters in column 5 to 7 denotes permission given to the group, to which user belongs. Similarly the characters in column 8 to 10 denote permission given to any other user or group. Generally the term others refer to the users of the system that do not belong to group to which the owner of the file belongs. A user if wishes can change the permission of an object that he/she owns.

The file permissions can also be given as numeric representation. We use octal (base of 8) number system for representing permissions as numeric values. Every octal digit combines read, write and execute permissions together. For example, in permission 644, “6” refers to the rights of the owner, “4” refers to rights of the group and “4” refers to rights of others. The permission 0644 when assigned is interpreted as read and write permission to owner, only read permission to group and others. Table 6.6 shows the interpretation of the octal numbers when used as permission.

Permission in text mode	Permission in octal mode	Meaning
---	0	No permission assigned
--x	1	Only execute access is allowed
-w-	2	Only write access is allowed
-wx	3	Write and execute access are allowed
r--	4	Only read access is allowed
r-x	5	Read and execute access are allowed
rw-	6	Read and write access are allowed
rwX	7	Everything is allowed

Table 6.6 : Octal numbers and permission

Changing Permissions (*chmod*)

To change the permission we use the *chmod* command. The operation of changing the permission is also known as change mode operation. For instance, in the above example we have seen that user (owner) has read and write permissions on the file. To make the file read only file the following command can be used :

Schmod ugo-w introduction.txt

The character ‘u’ in the above chmod command stands for user, ‘g’ for group and ‘o’ for other. After executing the command if we again list the file, then the output will be similar to the one shown below.

Sls -l introduction.txt

-r--r--r-- 1 administrator administrator 307 2013-02-11 14 :19 introduction.txt

Please note that write operation will not be permitted on this file. Additionally it also prevents user from deleting the file intentionally or unintentionally.

To assign write and execute permission to the owner of a file, execute the following command.

Schmod u+wx script10.sh

Sls -l script10.sh

-rwxr--r-- 1 administrator administrator 226 2013-02-20 16 :05 script10.sh

Here the file script10.sh is known as a script file. We are going to learn about shell scripting in the next chapters.

Table 6.7 shows some abbreviations and its meaning when used with the chmod command.

Category	Operation	Permission
u-user	+ assign permission	r- read permission
g-group	- remove permission	w - write permission
o-other	= assign absolute permission	x - execute permission
a-all		

Table 6.7 : Abbreviation used by chmod

I/O Redirection

A user interacts with the Operating System using a standard input device (keyboard). The Operating system displays the output on standard output device (monitor). Thus if any command is executed its input will be taken from the keyboard and output will be displayed on the monitor.

Sometimes it is useful to redirect the input or output to a file or a printer. Linux provides redirection symbols to change the standard input flow. The greater than symbol ‘ > ‘ implies redirection of output. It instructs the OS to put the output in the destination (file) specified by the user instead of displaying on the monitor screen. Similarly the less than symbol ‘ < ‘ implies redirection of input, it instructs the OS to accept the input from the specified source (file) instead of keyboard.

Assume that we issue a command **wc -l < introduction.txt**, here we are instructing OS to accept the input from a file named introduction.txt instead of the keyboard. Similarly the command **ls > list.txt** when executed will transfer the output of ls command to a file named list.txt instead of the monitor. When output redirection is used the output will not be displayed on the monitor hence to see the output we will have to use the command **cat list.txt**.

Piping

Redirection facility discussed above helps in associating the Linux commands to files. Many times we need to use multiple commands to perform a single operation. The piping facility of Linux helps in such cases. The pipe symbol (|) is used to provide the output of one command as an input to another command. The process of converting output of one command into input of another command is known as piping. Let us see an example of piping.

\$ls | wc -l

When we execute the above command, the output of *ls* command becomes the input to the *wc* command. Thus we will get information of total number of files in a current directory. Real power of pipe facility can be availed when we use it along with filters. Filters have been discussed in the next section.

Filters

Filters are commands that accept data from the standard input, process or manipulate it and then write the results to the standard output. Various filters like head, tail, cut, paste, sort and uniq are available in Ubuntu Linux. Let us see the working of these filters.

Displaying lines from top of the file (*head*)

The head command is used to display the required number of lines at the beginning of the file based on user's requirement. When used without any option it displays first 10 lines of the file. To display the lines as per users requirement we need to pass an argument to the head command. For example to display first 2 lines of the file *introduction.txt*, execute the following command :

Head -2 introduction.txt

Displaying lines from bottom of the file (*tail*)

The tail command works exactly opposite of the head command. It displays specified number of lines from the end of the file. To display last 2 lines of the file *introduction.txt*, execute the following command :

Tail -2 introduction.txt

We can use the tail command to display lines from nth line within the files. For example if we execute the following command :

Tail +5 introduction.txt will display lines from 5th line onwards from the file *introduction.txt*.

Slicing a file vertically (*cut*)

The head and tail command discussed in the above sections are used to slice the file horizontally. We can slice the data within the file vertically using the *cut* command. The cut command gives exact and precise outputs if the file has specific delimiters. Let us create one such delimited file to understand the working of the cut command. Create a file named *address.txt* using cat command that stores the data as shown :

Cat address.txt

20013, Vaidehi, Sanjay, Shah, Sector-23, GH-6, Gandhinagar, 382023

20014, Dhruvil, Ajay, Patel, Yesh Enclave, Mota Bazar, Vidyanagar, 388120

20015, Harshit, Amit, Jain, 58, Jaldeep I, Ahmedabad, 380058

20016, Abdul, Shamsheer, Khan, Khan Villa, M G Road, Nadiad, 388011

20017, Nirav, Jose, Mackwan, Jose House, M G Road, Nadiad, 388011

20018, Vidita, Harshal, Arolkar, 17, Jaldeep I, Ahmedabad, 380058

Let us see how to use the cut command along with its various options.

Cutting Characters (-c)

To extract specific characters from each line of the file, cut command with -c option is used. For instance, to extract the roll numbers and first names from the file *address.txt* execute the following command :

```
Scut -c 1-15 address.txt
```

```
20013, Vaidehi,  
20014, Dhrumil,  
20015, Harshit,  
20016, Abdul, S  
20017, Nirav, J  
20018, Vidita,
```

Though the output looks fine to certain extent, it is not exactly the same as we would have expected. Look at the data of Abdul and Nirav it has additional characters. The `-c` option is useful for fixed length fields, we had problem in the output as the first names were not stored using a fixed length.

Cutting fields (-f)

To overcome the problem mentioned in the `-c` option we may use a delimiter. The `cut` command can treat values separated by the delimiter as separate field values. Observe that we have used comma `,` as a delimiter in the `address.txt` file. Thus, to extract roll number and first name only we may execute the following command :

```
Scut -d "," -f 1,2 address.txt
```

```
20013, Vaidehi  
20014, Dhrumil  
20015, Harshit  
20016, Abdul  
20017, Nirav  
20018, Vidita
```

Observe that we have got the desired output now. In this command the `-d` is option used to specify delimiter appearing in a file (comma in our case) and `-f` option is used to specify field numbers to be displayed (roll number (1) and first name (2) in our case).

It is also possible to slice the file vertically from fields in between. For example assume that we want to display the first name, city and pin-code then we need to cut field numbered 2 and 7 onwards. The said operation can be performed by executing the following command :

```
Scut -d "," -f 2,7- address.txt
```

```
Vaidehi, Gandhinagar, 382023  
Dhrumil, Vidyanagar, 388120  
Harshit, Ahmedabad, 380058  
Abdul, Nadiad, 388011  
Nirav, Nadiad, 388011  
Vidita, Ahmedabad, 380058
```

Here 7- in the command imply that we need to display all fields after field number seven (including seven) from the file address.txt. It is also possible to redirect the output to a file. For example if we execute the following command :

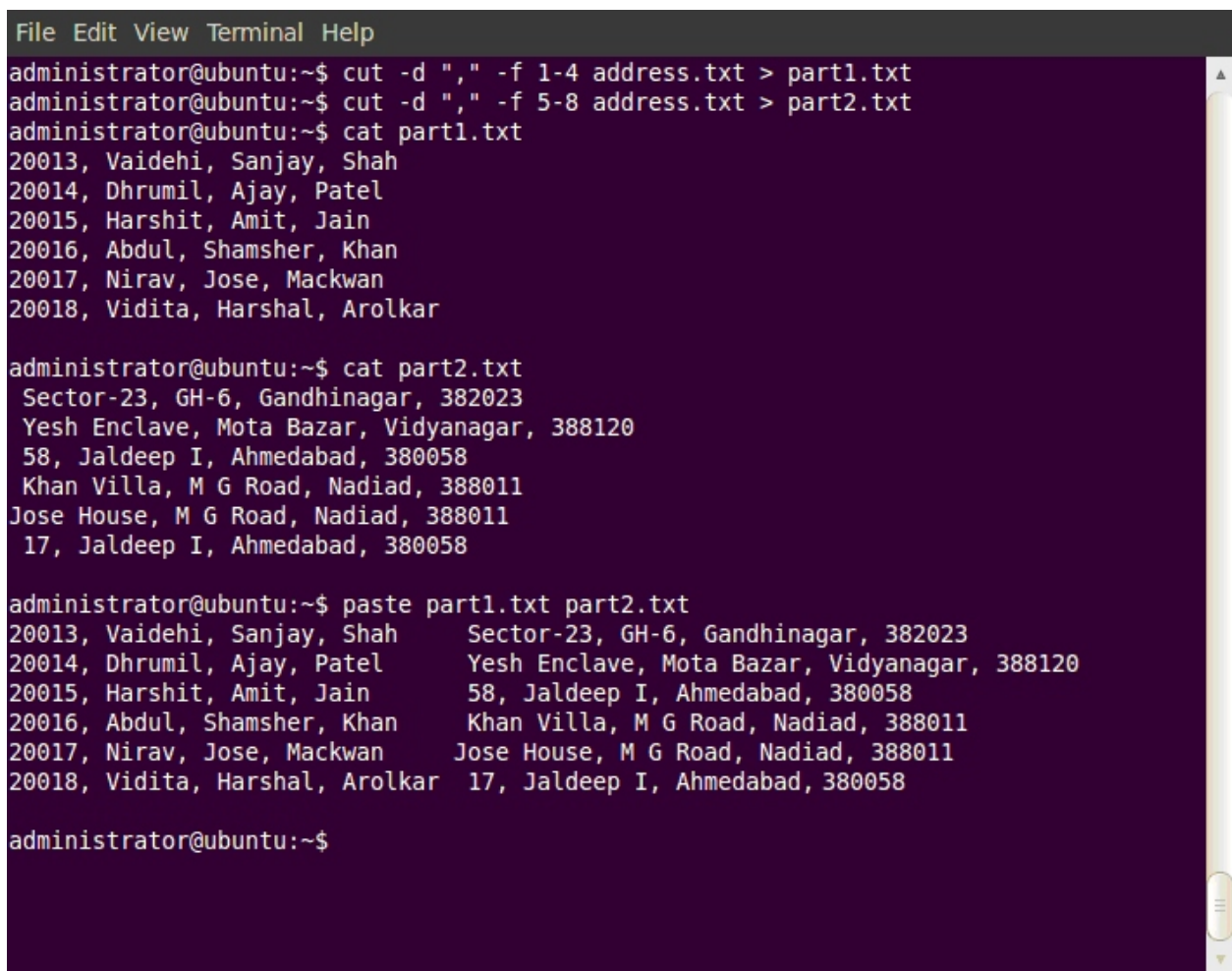
```
Scut -d "," -f 2,7- address.txt > output_cut.txt
```

the output instead of being displayed on monitor will be transferred to the file output_cut.txt.

Joining Contents (*paste*)

Two files can be pasted together using the *paste* command. For the paste command to work properly we need to ensure that both the files have exactly the same number of lines. If the number of lines is not same then the command may not result into expected output as it pastes from top of the files.

We will first create two different files named part1.txt and part2.txt using cut command and then join them using the paste command. Figure 6.18 shows the process of performing this operation.



```
File Edit View Terminal Help
administrator@ubuntu:~$ cut -d "," -f 1-4 address.txt > part1.txt
administrator@ubuntu:~$ cut -d "," -f 5-8 address.txt > part2.txt
administrator@ubuntu:~$ cat part1.txt
20013, Vaidehi, Sanjay, Shah
20014, Dhrumil, Ajay, Patel
20015, Harshit, Amit, Jain
20016, Abdul, Shamsheer, Khan
20017, Nirav, Jose, Mackwan
20018, Vidita, Harshal, Arolkar

administrator@ubuntu:~$ cat part2.txt
Sector-23, GH-6, Gandhinagar, 382023
Yesh Enclave, Mota Bazar, Vidyanagar, 388120
58, Jaldeep I, Ahmedabad, 380058
Khan Villa, M G Road, Nadiad, 388011
Jose House, M G Road, Nadiad, 388011
17, Jaldeep I, Ahmedabad, 380058

administrator@ubuntu:~$ paste part1.txt part2.txt
20013, Vaidehi, Sanjay, Shah      Sector-23, GH-6, Gandhinagar, 382023
20014, Dhrumil, Ajay, Patel      Yesh Enclave, Mota Bazar, Vidyanagar, 388120
20015, Harshit, Amit, Jain       58, Jaldeep I, Ahmedabad, 380058
20016, Abdul, Shamsheer, Khan    Khan Villa, M G Road, Nadiad, 388011
20017, Nirav, Jose, Mackwan      Jose House, M G Road, Nadiad, 388011
20018, Vidita, Harshal, Arolkar   17, Jaldeep I, Ahmedabad, 380058

administrator@ubuntu:~$
```

Figure 6.18 : Example of paste command

Ordering Output (*sort*)

The *sort* command is used to order the data stored within a file in ascending or descending sequence at the time of display. Like the *cut* command, it also identifies fields and can sort on specified fields. When the *sort* command is used without any options, it sorts the file based on entire line. It reorders the lines based on ASCII sequence. The sorting is first applied on white spaces, followed by numerals, uppercase letters and finally lowercase letters.

Let us try to arrange the file *address.txt* in descending order of roll numbers (you must have observed that it is already arranged in ascending order). To display contents of file sorted in reverse order, execute the following command :

```
$sort -r address.txt
```

20018, Vidita, Harshal, Arolkar, 17, Jaldeep I, Ahmedabad, 380058

20017, Nirav, Jose, Mackwan, Jose House, M G Road, Nadiad, 388011

20016, Abdul, Shamsher, Khan, Khan Villa, M G Road, Nadiad, 388011

20015, Harshit, Amit, Jain, 58, Jaldeep I, Ahmedabad, 380058

20014, Dhrumil, Ajay, Patel, Yesh Enclave, Mota Bazar, Vidyanagar, 388120

20013, Vaidehi, Sanjay, Shah, Sector-23, GH-6, Gandhinagar, 382023

Note :

The execution of *sort* command does not modify the actual file. The records in the actual file will remain in the same position. The sort order is applied only at the time of displaying the output.

The *sort* command is mostly used in conjunction with other commands. For example, we can use it with the *cut* command to order the output of the *cut* command. Execute the following command to see the output of both the *cut* and *sort* commands when combined.

```
$cut -d "," -f 2-4 address.txt | sort
```

Abdul, Shamsher, Khan

Dhrumil, Ajay, Patel

Harshit, Amit, Jain

Nirav, Jose, Mackwan

Vaidehi, Sanjay, Shah

Vidita, Harshal, Arolkar

The *cut* command in the above example extracts second, third and fourth column from the file *address.txt*. The extracted output is then given as input to the *sort* command. The *sort* command then sorts the contents and displays it on the screen.

Character Conversion (*tr*)

The command used as filters work with a line or column. The *tr* (translate) command allows us to work with individual characters within a line. It is used to translate (convert) strings or patterns from one set of characters to another.

Working with the `address.txt` file you must have observed that it contains “,” as delimiter. Assume that we do not want to show the delimiter at the time of the display, instead we would like to show a blank space. The *tr* command allows us to perform this operation. Execute the command shown below :

```
Scat address.txt | tr -s '[,]' '[ ]'
```

20013 Vaidehi Sanjay Shah Sector-23 GH-6 Gandhinagar 382023

20014 Dhrumil Ajay Patel Yesh Enclave Mota Bazar Vidyanagar 388120

20015 Harshit Amit Jain 58 Jaldeep I Ahmedabad 380058

20016 Abdul Shamsher Khan Khan Villa M G Road Nadiad 388011

20017 Nirav Jose Mackwan Jose House M G Road Nadiad 388011

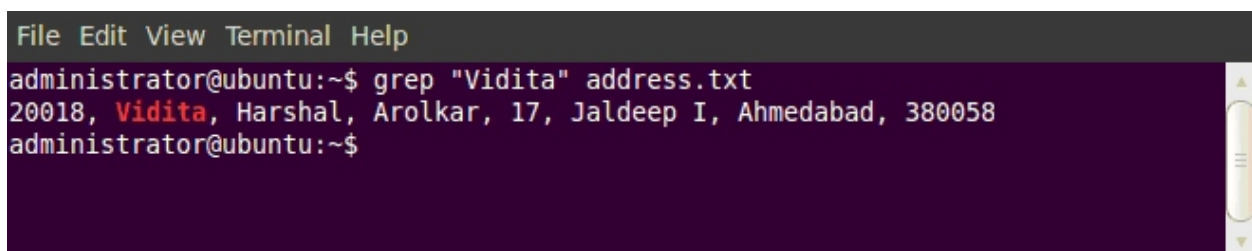
20018 Vidita Harshal Arolkar 17 Jaldeep I Ahmedabad 380058

The translation is only applicable at the display it will not permanently replace the actual delimiter. The `-s` option squeezes the additional space visible in the actual file. In case we need to save the translations visible we can redirect the output to a new file.

Pattern matching (*grep*)

The find operation is one of the most widely used operation in GUI applications. We must have used the CTRL + f keys to find a keyword within files. The *grep* command performs similar operation from the command line interface. The command is based on a fundamental idea of search globally for a regular expression and display lines where instances are found (g/re/p).

Let us make use of the *grep* command to find a name within the file `address.txt` and display its record. Figure 6.19 shows the working of the *grep* command.



```
File Edit View Terminal Help
administrator@ubuntu:~$ grep "Vidita" address.txt
20018, Vidita, Harshal, Arolkar, 17, Jaldeep I, Ahmedabad, 380058
administrator@ubuntu:~$
```

Figure 6.19 : Working of *grep* command

Observe that the string that we are looking for is shown in red colour, also we have enclosed the keyword in double quote. It is not compulsory to enclose the keyword in double quotes hence the command **grep Vidita address.txt** will also give same output. The string used in the grep command is case sensitive hence the strings “Vidita” and “vidita” are different. We can use different options along with grep command that will help us refine our search in a better way. Table 6.8 lists the options and their usage.

Option	Usage
-c	Return only the number of matches, without quoting the text
-i	Ignore case while searching
-l	Return only file names containing a match, without quoting the text.
-n	Return the line number of matched text, as well as the text itself.
-v	Returns all the lines that do not match the text.
-w	Return lines which display only whole words
-o	Shows only the matched string

Table 6.8: Options of grep command

One very powerful feature of grep command is to use a regular expression as a keyword. Say for example we want details about the persons who are staying in society that has ‘Jal’ as its starting and ‘I’ as its end, then regular expression can be used. Let us execute the command shown here :

\$grep "Jal.*I" address.txt

The output of this command is shown in figure 6.20. Here “Jal.*I” is a regular expression.



```

File Edit View Terminal Help
administrator@ubuntu:~$ grep "Jal.*I" address.txt
20015, Harshit, Amit, Jain, 58, Jaldeep I, Ahmedabad, 380058
20018, Vidita, Harshal, Arolkar, 17, Jaldeep I, Ahmedabad, 380058
administrator@ubuntu:~$
  
```

Figure 6.20 : Using regular expressions in grep command

A regular expression is normally followed by one of several repetition operators shown in table 6.9.

Repetition Operator	Meaning
?	The preceding item is optional and matched at most once.
*	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
{n}	The preceding item is matched exactly n number of times.
{n,}	The preceding item is matched n or more number of times.
{,m}	The preceding item is matched at most m number of times.
{n,m}	The preceding item is matched at least n number of times, but not more than m number of times.

Table 6.9 : Repetition operator

Searching a file or Directory (*find*)

Many times we forget the location of the file or a directory that we have created. The *find* command helps us look for such forgotten objects. The *find* command looks for the search criteria (file or directory or both) that you have specified starting from the directory you specify within all its sub directories. We can also search for the object based on its name, owner, group, type, permissions, date, and other criteria. Note that the *find* command when used without any other arguments displays the pathname of all the files and directories in the present directory and all its subdirectories.

Assume that we want to look for the location of file *introduction.txt* that we had created earlier, the command would be

```
$find -name introduction.txt
```

If the file exists then its path will be given as an output. Otherwise we may either get an error or a prompt will be visible if no such file exists. Note that we know the name of the file here, what if we only remembered first few characters of the file name. The wildcard characters can be used in such cases. The example shown below helps in finding all files that start with string “intro”.

```
$find -name intro*
```

```
./subject/economics/my1_dir/intro1
./subject/economics/my1_dir/intro
./subject/economics/introduction.txt
./subject/economics/introduction
./subject/economics/my_dir/intro1
./subject/economics/my_dir/intro
./subject/economics/intro1
```


The output of this command may vary on your screens depending on the number of files that you have which start with string “intro”. Also in Linux we may not assign a file extension, hence we may not be able to know whether intro1, intro and introduction in the above outputs are files or directories. We may refine the search if needed by using the *type* option as shown below :

\$find -name intro* -type f

Table 6.10 shows some example of find and its expected output description.

Command	Description
find / -type d	Search all directory and sub directory available on root only.
find . -mtime -1	Search objects modified within the past 24 hours.
find . -mtime +1	Search objects modified more than 48 hours ago.
find ./dir1 ./dir2 -name script.sh	Search directories “./dir1” and “./dir2” for a file “script.sh”.
find -size 0 -delete	Search for files of zero byte and delete them from the disk.
find -executable	Search for the executable file in current directory.
find /home -user jagat	Search for the object whose owner is jagat within the home directory and its sub directory.
find . -perm 664	Search for object that has read and write authorization for their owner, and group but which other users can only read.

Table 6.10: Example of find command

Running Commands as the Superuser

When you log in to your computer, the account you use is a regular user account. This account has a limited right. The security model of Ubuntu generally allows you to work as a normal user. Not providing administrative rights prevents any accidental changes or installation of malicious programs that may disturb functioning of the system. But many times the user may need to have administrative privileges. The administrative privileges are available to only a user known as superuser. To use the superuser account when using the terminal, we need to add *sudo* as a prefix to the commands that we want to execute. For example, execute the following command to install a new program called skype from command line.

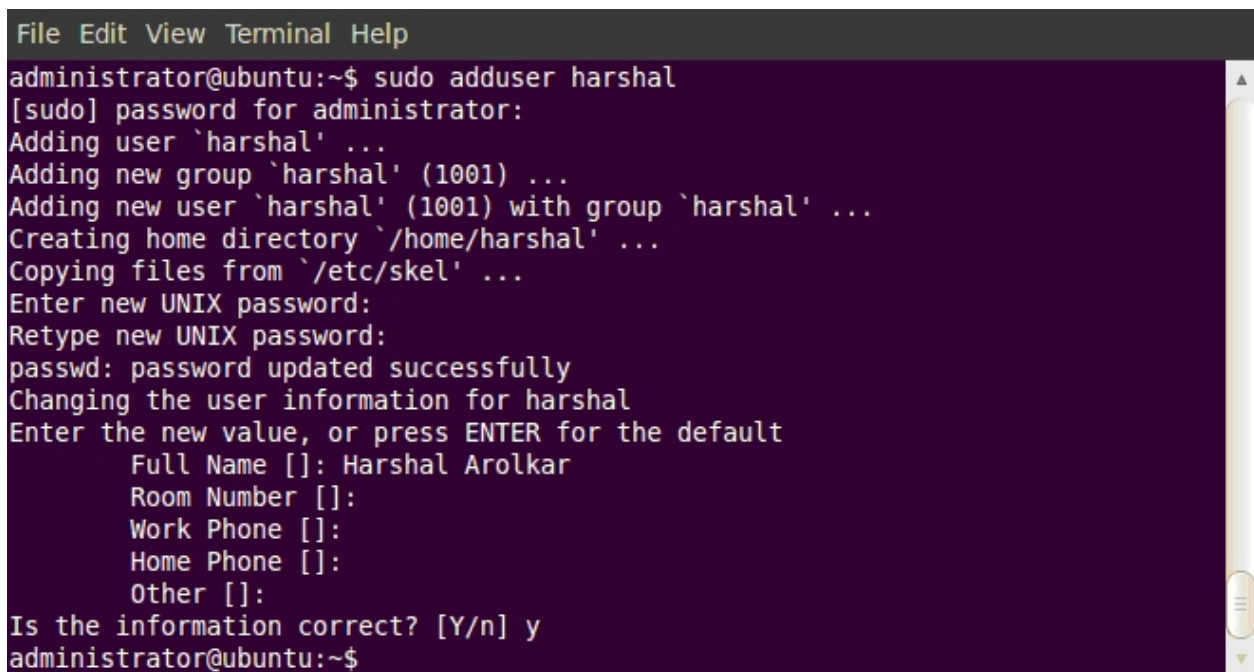
\$sudo apt-get install skype

When you execute the command it will ask for password, provide the password of superuser (generally it will be different from the normal user). This password is the password of the first user that you

added when you installed Ubuntu Linux on the computer. Once the user is authenticated as sudo by means of the terminal, the software will start installing. Once the installation is over we can start using the software.

The super user can also perform the operations of adding, deleting or updating a user, group or object in the system. Some of the commands listed below are used for such purposes.

adduser : The *adduser* command creates a new user on the system. Figure 6.20 shows the process of creating a user. The command when executed will ask for password and some additional details as shown in figure 6.21. Once all the details are provided a new user along with its home directory will be created in the system.

A terminal window with a dark purple background and white text. The window title is "File Edit View Terminal Help". The command "sudo adduser harshal" is entered. The terminal shows the following output: "[sudo] password for administrator:", "Adding user 'harshal' ...", "Adding new group 'harshal' (1001) ...", "Adding new user 'harshal' (1001) with group 'harshal' ...", "Creating home directory '/home/harshal' ...", "Copying files from '/etc/skel' ...", "Enter new UNIX password:", "Retype new UNIX password:", "passwd: password updated successfully", "Changing the user information for harshal", "Enter the new value, or press ENTER for the default", "Full Name []: Harshal Arolkar", "Room Number []:", "Work Phone []:", "Home Phone []:", "Other []:", "Is the information correct? [Y/n] y", and finally "administrator@ubuntu:~\$".

```
File Edit View Terminal Help
administrator@ubuntu:~$ sudo adduser harshal
[sudo] password for administrator:
Adding user `harshal' ...
Adding new group `harshal' (1001) ...
Adding new user `harshal' (1001) with group `harshal' ...
Creating home directory `/home/harshal' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for harshal
Enter the new value, or press ENTER for the default
    Full Name []: Harshal Arolkar
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
administrator@ubuntu:~$
```

Figure 6.21 : Adding a user

passwd : The *passwd* command when executed as a super user do, allows us to change the password of any valid user of the system.

who : The *who* command when executed displays the list of all the users that are presently logged into the machine.

addgroup : The *addgroup* command adds a new group. The users are normally divided into groups so that they can be better controlled.

deluser : The *deluser* command is used to delete a user from the system. Note that we need to explicitly remove the user's files and home directory, by using the *-remove -home* option.

delgroup : The *delgroup* command deletes a group from the system. To perform this operation we must first make sure that no user is associated with the group that we are going to delete.

Summary

In this chapter we learned how to use the Ubuntu Linux command line interface. The CLI (Command Line Interface) when used allows us to perform all operations that we perform using the GUI in efficient and fast manner. We saw how to initiate the CLI using the Linux terminal. We also learned how to create, rename and delete a file or a directory, find out the directory that we are working in, change the directory if required. Later we saw how to create a copy of the file as well as directory. An access right is one of the ways to make sure that our data is not misused; we saw how to assign or change access rights. We saw how to increase the effectiveness of commands by joining multiple commands using the pipes. Further we looked at some features like counting the words or lines within the file, slicing the file horizontally and vertically, joining the file, searching for a file, directory or a string pattern within the files, arranging the display in ascending or descending order. Finally we saw how normal user can perform administrative tasks of installing new software, adding or deleting a user or group, checking who is using the system or change the password of some user.

EXERCISE

1. What is command prompt?
2. Describe shell. Name any three Linux shells.
3. How shells interpret command? Explain with suitable figure.
4. Write the steps used to start a terminal in Ubuntu Linux.
5. Explain following command in detail :
ls, cat, wc, chmod
6. What is the meaning of internal commands in Ubuntu Linux?
7. How can you find help for any command in Ubuntu Linux?
8. Explain different wildcard characters, giving suitable example.
9. Explain the use of pipes, giving suitable example.
10. What is redirection? Explain giving suitable example.
11. List the filter commands used in Linux.
12. **Choose the most appropriate option from those given below :**
 - (1) Which of the following command is used to count the total number of lines, words, and characters contained in a file?
 - (a) countw
 - (b) wcount
 - (c) wc
 - (d) wordcount

- (2) Which of the following command is used to remove files?
- (a) dm (b) rm
(c) delete (d) erase
- (3) Which of the following command is used to remove the directory?
- (a) rdir (b) remove
(c) rd (d) rmdir
- (4) Which of the following command is used to count just the number of lines contained in a file?
- (a) wc - r (b) wc - w
(c) wc - c (d) wc - l
- (5) The command `chmod 761` letter is equivalent to which of the following access rights?
- (a) `chmod u=7, g = 6, o = 1`
(b) `chmod a = 761`
(c) `schmod u = rws, g = rw, o = x`
(d) `chmod 167`
- (6) Which of the following refers to the maximum length of a filename in Linux?
- (a) 8 (b) 10
(c) 200 (d) 255
- (7) The hierarchy of a series of directories branching in a user system starts from which of the following directories?
- (a) `\home` (b) `\root`
(c) `/home` (d) `/root`
- (8) Which of the following command is used to copy a file?
- (a) tar (b) cpio
(c) cp (d) copy
- (9) Which of the following command is used to display your current working directory?
- (a) path (b) pwd
(c) `prompt pg` (d) dir
- (10) Which of the following command is used for searching a pattern in a file?
- (a) grep (b) find
(c) lookup (d) All of the above
- (11) Which of the following is not a redirection symbol?
- (a) > (b) <
(c) * (d) >>

- (12) Which of the following syntax is correct to assign a read permission on a user file?
- (a) chmod r filename (b) chmod u+r filename
(c) chmod filename r (d) chmod filename u+r
- (13) Which of the following refers to the minimum arguments of cp command?
- (a) One (b) Two
(c) Three (d) None
- (14) The mv command in Linux is used for which of the following purpose?
- (a) To rename a directory (b) To move a file
(c) To copy a file (d) All of the above
- (15) Which of the following command is used to view one page content on the screen at a time?
- (a) More (b) more
(c) PAGE (d) page

Laboratory Exercises

- 1. Perform the following using Linux commands :**
- (a) Print the calendar of December 2012.
 - (b) Execute the command which displays login name, the name of your terminal and date and time since user logged in.
 - (c) List all files starting with character 'n' or 'N'.
 - (d) Display the current working directory.
 - (e) Prepare two files named class11_A.txt and class11_B.txt containing details of students of eleventh standard. The file should contain names of the students. Now merge these two files in a single file and name it class11_.txt. (Use cat command).
 - (f) Hide the three files created in question 'e'.
 - (g) List only directories.
 - (h) Get help on the use of the cat command.
 - (i) List all files whose fourth character is 'g' and sixth character is digit.
 - (j) Using terminal calculator perform the following operation :
 - (1) Calculate 2500/7
 - (2) Convert decimal 50 to its binary equivalent
 - (3) Convert decimal 25 to its hexadecimal equivalent
 - (4) Find square root of 36
 - (5) Convert hexadecimal 25 to its decimal equivalent

2. Show the output of following Linux commands :

- (a) `cat f1 >> f2`
- (b) `echo $SHELL`
- (c) `mkdir d1 d2 d3`
- (d) `ls s*t`
- (e) `ls [a-f]*`
- (f) `ls -R`
- (g) `ls -xR`
- (h) `cp f1 f2`
- (i) `ls | sort`
- (j) `ls | tr -s " " | cut -d " " -f 5 | sort`
- (k) `ls -l | grep -c "address.txt"`
- (l) `grep "Harshit Jain" address.txt`
- (m) `chmod u-w address.txt`
- (n) `wc -l address.txt > totalstudents`

