UNIX

What is Unix ?

The Unix operating system is a set of programs that act as a link between the computer and the user.

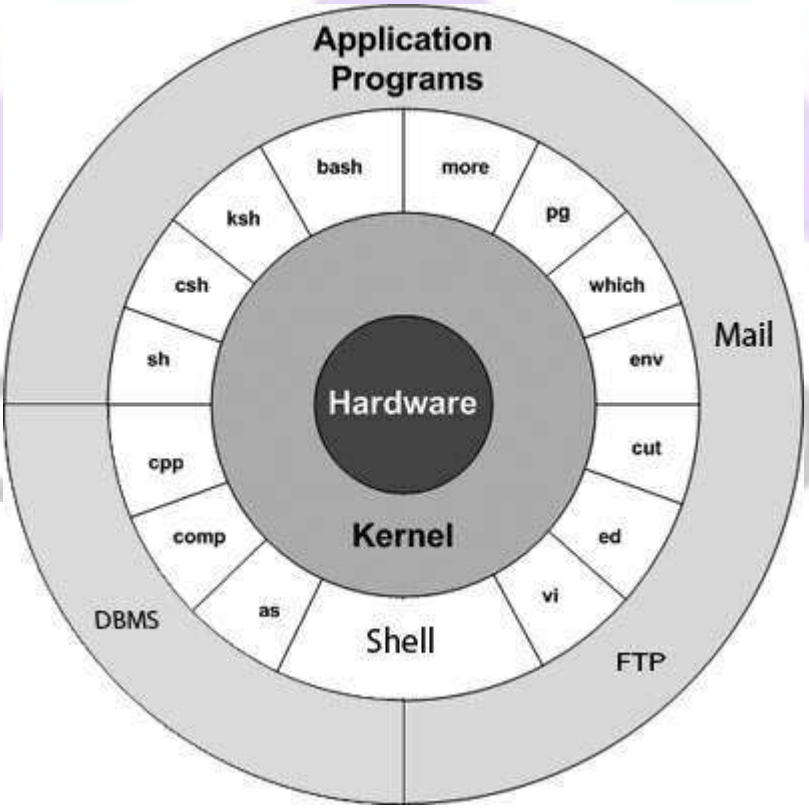
The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples. Linux is also a flavor of Unix which is freely available.
- Several people can use a Unix computer at the same time; hence Unix is called a multiuser system.
- A user can also run multiple programs at the same time; hence Unix is a multitasking environment.

Unix Architecture

Here is a basic block diagram of a Unix system −



The main concept that unites all the versions of Unix is the following four basics −

- **Kernel** − The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.

Shell − The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.

Windows vs. Linux

---

1. Windows is proprietary and Linux is Open Source
2. Windows is NOT case sensitive but Linux is 100% case sensitive
3. In Windows the superuser is Administrator but in Unix and Linux, it is root
4. We do use backslash(\) while defining path of a file, but in Linux it is front slash(/)
5. Normal user/Administrator ==> C:\ , in Linux for root ==> # and for normal users ==> $
6. Linux has single-rooted directory structure but windows has multi-rooted directory structure

Linux Distributions

Linux Distribution ==> Kernel + GNU Utils + Shell
Distributions ==> Slackware / Oracle Linux / Suse Linux / Ubuntu

Redhat ==> Redhat Linux [FREE] --> 4 / 5 / 6 / 7 / 8 / 9

Enterprise Distribution ⇒ RHEL [Redhat Enterprise Linux] ⇒ RHEL8

Community Distribution ⇒ Fedora / CentOS To display the RHEL version ⇒ cat /etc/redhat-release

Basic Linux Commands

---

1. Create a user and assign a password
   useradd student
   passwd student

2. To switch to a particular user ==> su [Switch User]
   su student [It will only switch the UID but not the home directory]su -
   student [Will change the UID and the Home directory]

Note: By default, root user will have home directory as /root and non-root users will have home directory within /home

3. To list the files/directories ==> ls
   ls -l [long listing]

4. To display the current working directory ==> pwd

5. To change to a directory ==> cd <dir_name>
   To go to the home directory => cd OR cd ~ [~ represents the home directory]

6. Symbolic representations ==>
   . ==> Current Directory
   .. ==> Parent Directory
   ../.. ==> Parent to Parent directory
   - ==> Previous directory

7. Shell commands have 3 basic parts - Command to run / Options to adjust the commandbehavior / Arguments
   ls -l /tmp

8. cal ==> to display the calendar
   cal 2021 ⇒ Displays the calendar of the entire yearcal
   12 2021 ⇒ Displays calendar of December

9. To redirect the output of a command to a file ⇒ cal 2021 > mycal2021
   cat mycal2021

10. mkdir => To create a directory
   mkdir database
   cd database
   mkdir dir1
   mkdir dir2

   mkdir -p database/{oracle,mssql}

Path ==> Absolute Path and Relative Path

   /home/student/database/oracle/ora.txt ==> Absolute Path
   database/oracle/ora.txt ==> Relative Path

rmdir ==> To remove an empty directory

rm -rf ==> To remove a directory along with the contentsrm <filename> ⇒ To delete a file

To see the manual page of a Linux command ⇒ man <command_name>

11. Create files ==>
   touch ==> to create empty file
   cat > file1
   hello
   <ctrl> + d

cat file1 ==> To display the content of the file
>> => Append
cat file1 >> file2 ==> Content of file1 would be appended to file2

12. Copy ⇒ cp
Move ⇒ mv

13. who ==> To list the currently logged-in users
who am i / whoami ==> To list the current user only

14. history ⇒ To list all commands which we have executed
!NO ⇒ To recall a command from history
history -d NO ⇒ To delete a specific command from history
history -c ⇒ To clear the history

16. Word Count ⇒ wc
wc /etc/passwd ⇒ Displays no. of lines/words/characters within /etc/passwd filewc -l /etc/passwd ⇒ Lines
wc -w /etc/passwd ⇒ Words
wc -c /etc/passwd ⇒ Characters
wc -lc /etc/passwd ⇒ Lines+Characters

17. head ⇒ displays first 10 lines from a file head
/etc/passwd ⇒ displays first 10 lines head -n 5
/etc/passwd ⇒ displays first 5 lines

18. tail ⇒ displays last 10 lines from a file tail
/etc/passwd ⇒ displays last 10 lines
tail -n 5 /etc/passwd ⇒ displays last 5 lines

19. Linux Editors ⇒ vi [Visual Editor] ⇒ Default editor of Unix vim [Visual Improved] ⇒ Default editor of Linuxnano => simple editor just like notepad
gedit => Graphical Editor
gvim => Graphical version of vimvi / vim commands

- - - - - - - - - - - - - -

i => Insert Mode

a => Append Mode

x => To delete a single characterdd

=> To delete a line

2 yy(yank) p => Copies 2 lines and paste2 dd

p => Deletes 2 lines and paste

o => To insert a line below O

=> To insert a line above u =>

Undo the last operation

/expression => To search for an expression

:nohl => To remove highlights

:wq => To save and Quit

:w! => Save without Quit

:q! => Quit without Save

:se nu => To display the line numbers

:set nonumber => To remove the line numbers

Find and Replace ⇒ :%s/linux/RHEL/g ==> It will replace all occurrences of linux byRHEL

-
- **Commands and Utilities** − There are various commands and utilities which you can make use of in your day-to-day activities. **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.
- **Files and Directories** − All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the **filesystem**.

System Bootup

If you have a computer which has the Unix operating system installed in it, then you simply need to turn on the system to make it live.

As soon as you turn on the system, it starts booting up and finally it prompts you to log into the system, which is an activity to log into the system and use it for your day-to-day activities.

Types of account in UNIX

1. Root/super account
2. System account
3. User account
4.

USER ACCOUNT

1. Add a user account
    a) Using useradd command

Only one user can be added and that username must be unique (different from other usernames already exists on the system).
For example, to add a new user called '**tecmint**', use the following command.

```
[root@tecmint ~]# useradd tecmint
```

it gets created in a locked state and to unlock that user account, we need to set a password for that account with the '**passwd**' command

b) Create user with specific user ID
But, we can create users with custom userid with the '**-u**' option. For example, the following command will create a user '**navin**' with custom userid '**1002**'.

```
[root@tecmint ~]# useradd -u 1002 navin
```

2. Modify user account
It enables you to make a changes to an existing account from the command line using 'usermod' command.

3. Deleting a user account
This can be used to delete the existing user using 'userdel' command.
Example : $userdel -r **tecmint**

Login Unix

When you first connect to a Unix system, you usually see a prompt such as the following −

login:

To log in

- Have your userid (user identification) and password ready. Contact your system administrator if you don't have these yet.
- Type your userid at the login prompt, then press **ENTER**. Your userid is **case-sensitive**, so be sure you type it exactly as your system administrator has instructed.
- Type your password at the password prompt, then press **ENTER**. Your password is also case-sensitive.
- If you provide the correct userid and password, then you will be allowed to enter into the system. Read the information and messages that comes up on the screen, which is as follows.

```
login : amrood
amrood's password:
Last login: Sun Jun 14 09:32:32 2009 from 62.61.164.73
$
```

You will be provided with a command prompt (sometime called the **$** prompt) where you type all your commands. For example, to check calendar, you need to type the **cal** command as follows −

```
$ cal
     June 2009
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

$
```

Change Password

All Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and crackers. Following are the steps to change your password −

Step 1 − To start, type password at the command prompt as shown below.

Step 2 − Enter your old password, the one you're currently using.

Step 3 − Type in your new password. Always keep your password complex enough so that nobody can guess it. But make sure, you remember it.

Step 4 − You must verify the password by typing it again.

```
$ passwd
Changing password for amrood
(current) Unix password:******
New UNIX password:*******
Retype new UNIX password:*******
passwd: all authentication tokens updated  successfully

$
```

Note − We have added asterisk (*) here just to show the location where you need to enter the current and new passwords otherwise at your system. It does not show you any character when you type.

Listing Directories and Files

All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

You can use the **ls** command to list out all the files or directories available in a directory. Following is the example of using **ls** command with **-l** option.

```
$ ls -l
total 19621
drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
```

```
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf

$
```

Here entries starting with **d.....** represent directories. For example, uml, univ and urlspedia are directories and rest of the entries are files.

Who Are You?

While you're logged into the system, you might be willing to know : **Who am I**?

The easiest way to find out "who you are" is to enter the **whoami** command −

```
$ whoami
 amrood

$
```

Try it on your system. This command lists the account name associated with the current login. You can try **who am i** command as well to get information about yourself.

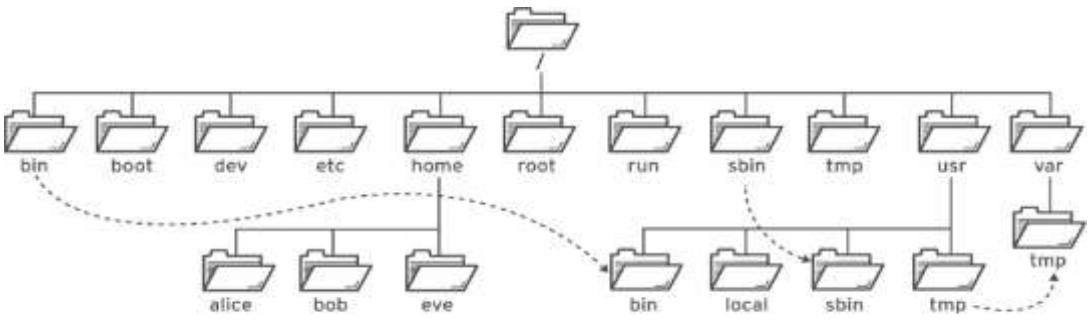
Logging Out

When you finish your session, you need to log out of the system. This is to ensure that nobody else accesses your files.

To log out

- Just type the **logout** command at the command prompt, and the system will clean up everything and break the connection.

Linux Filesystem Hierarchy ⇒ All files in a Linux system are stored on file systems, whichare organized into a single inverted tree of directories known as file-system hierarchy.



Significant file-system directories in Red Hat Enterprise Linux 8

Important Red Hat Enterprise Linux Directories

LOCATION	PURPOSE
/usr	Installed software, shared libraries, include files, and read-only program data. Important subdirectories include: • **/usr/bin**: User commands. • **/usr/sbin**: System administration commands. • **/usr/local**: Locally customized software.
/etc	Configuration files specific to this system.
/var	Variable data specific to this system that should persist between boots. Files that dynamically change, such as databases, cache directories, log files, printer-spooled documents, and website content may be found under **/var**.
/run	Runtime data for processes started since the last boot. This includes process ID files and lock files, among other things. The contents of this directory are recreated on reboot. This directory consolidates **/var/run** and **/var/lock** from earlier versions of Red Hat Enterprise Linux.
/home	*Home directories* are where regular users store their personal data and configuration files.
/root	Home directory for the administrative superuser, `root`.
/tmp	A world-writable space for temporary files. Files which have not been accessed, changed, or modified for 10 days are deleted from this directory automatically. Another temporary directory exists, **/var/tmp**, in which files that have not been accessed, changed, or modified in more than 30 days are deleted automatically.
/boot	Files needed in order to start the boot process.
/dev	Contains special *device files* that are used by the system to access hardware.

which ⇒ is used to display the path of a binary file ⇒ which cal

System Shutdown

The most consistent way to shut down a Unix system properly via the command line is to use one of the following commands −

Sr.No.	Command & Description
1	**halt** Brings the system down immediately
2	**init 0** Powers off the system using predefined scripts to synchronize and clean up the system prior to shutting down
3	**init 6** Reboots the system by shutting it down completely and then restarting it
4	**poweroff** Shuts down the system by powering off
5	**reboot** Reboots the system
6	**shutdown** Shuts down the system

You typically need to be the super user or root (the most privileged account on a Unix system) to shut down the system. However, on some standalone or personally owned Unix boxes, an administrative user and sometimes regular users can do so.

LINUX/UNIX compiler

In this chapter, we discuss about the Linux terminal installation and using online compilers.

- Linux terminal installation
  In this we could teach trainee how to enable Linux terminal in windows (If there is no such permission to enable on dxc pc, could work on citrix machine).
- Online bash compiler
  This is short topic, where they discuss about the available online compilers over the internet.
  This topic will help the trainee to simultaneously run and check the commands that they are being taught.
  Example for online compiler are replit, onlinegdb etc

Unix - File Management

In this chapter, we will discuss in detail about file management in Unix. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

When you work with Unix, one way or another, you spend most of your time working with files. This tutorial will help you understand how to create and remove files, copy and rename them, create links to them, etc.

In Unix, there are three basic types of files −

- **Ordinary Files** − An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
- **Directories** − Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.
- **Special Files** − Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Listing Files

To list the files and directories stored in the current directory, use the following command −

```
$ls
```

Here is the sample output of the above command −

```
$ls

bin       hosts  lib     res.03
ch07      hw1    pub     test_results
ch07.bak  hw2    res.01  users
docs      hw3    res.02  work
```

The command **ls** supports the **-l** option which would help you to get more information about the listed files −

```
$ls -l
total 1962188

drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
```

```
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
drwxr-xr-x  2    200    300      4096 Nov 25  2007 webthumb-1.01
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf
drwxr-xr-x 11 amrood amrood      4096 May 29  2007 zlib-1.2.3
$
```

Here is the information about all the listed columns −

- **First Column** − Represents the file type and the permission given on the file. Below is the description of all type of files.
- **Second Column** − Represents the number of memory blocks taken by the file or directory.
- **Third Column** − Represents the owner of the file. This is the Unix user who created this file.
- **Fourth Column** − Represents the group of the owner. Every Unix user will have an associated group.
- **Fifth Column** − Represents the file size in bytes.
- **Sixth Column** − Represents the date and the time when this file was created or modified for the last time.
- **Seventh Column** − Represents the file or the directory name.

In the **ls -l** listing example, every file line begins with a **d**, **-**, or **l**. These characters indicate the type of the file that's listed.

Sl.No.	Prefix & Description
1	**-** Regular file, such as an ASCII text file, binary executable, or hard link.
2	**b** Block special file. Block input/output device file such as a physical hard drive.
3	**c** Character special file. Raw input/output device file such as a physical hard drive.
4	**d** Directory file that contains a listing of other files and directories.

5	**l** Symbolic link file. Links on any regular file.
6	**p** Named pipe. A mechanism for interprocess communications.
7	**s** Socket used for interprocess communication.

Metacharacters

Metacharacters have a special meaning in Unix. For example, ***** and **?** are metacharacters. We use ***** to match 0 or more characters, a question mark (**?**) matches with a single character.

For Example −

```
$ls ch*.doc
```

Displays all the files, the names of which start with **ch** and end with **.doc** −

```
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc ch02-1.doc c
```

Here, ***** works as meta character which matches with any character. If you want to display all the files ending with just **.doc**, then you can use the following command −

```
$ls *.doc
```

Hidden Files

An invisible file is one, the first character of which is the dot or the period character (.). Unix programs (including the shell) use most of these files to store configuration information.

Some common examples of the hidden files include the files −

- **.profile** − The Bourne shell (sh) initialization script
- **.kshrc** − The Korn shell (ksh) initialization script
- **.cshrc** − The C shell (csh) initialization script
- **.rhosts** − The remote shell configuration file

To list the invisible files, specify the **-a** option to **ls** −

```
$ ls -a

.       .profile      docs     lib     test_results
..      .rhosts       hosts    pub     users
.emacs  bin           hw1      res.01  work
```

```
.exrc     ch07          hw2      res.02
.kshrc    ch07.bak      hw3      res.03
$
```

- **Single dot (.)** − This represents the current directory.
- **Double dot (..)** − This represents the parent directory.

Creating Files

You can use the **vi** editor to create ordinary files on any Unix system. You simply need to give the following command −

```
$ vi filename
```

The above command will open a file with the given filename. Now, press the key **i** to come into the edit mode. Once you are in the edit mode, you can start writing your content in the file as in the following program −

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
```

Once you are done with the program, follow these steps −

- Press the key **esc** to come out of the edit mode.
- Press two keys **Shift + ZZ** together to come out of the file completely.

You will now have a file created with **filename** in the current directory.

```
$ vi filename
$
```

Editing Files

You can edit an existing file using the **vi** editor. We will discuss in short how to open an existing file −

```
$ vi filename
```

Once the file is opened, you can come in the edit mode by pressing the key **i** and then you can proceed by editing the file. If you want to move here and there inside a file, then first you need to come out of the edit mode by pressing the key **Esc**. After this, you can use the following keys to move inside a file −

- **l** key to move to the right side.
- **h** key to move to the left side.
- **k** key to move upside in the file.
- **j** key to move downside in the file.

So using the above keys, you can position your cursor wherever you want to edit. Once you are positioned, then you can use the **i** key to come in the edit mode. Once you are done with the editing in your file, press **Esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

Display Content of a File

You can use the **cat** command to see the content of a file. Following is a simple example to see the content of the above created file −

```
$ cat filename
```

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
$
```

You can display the line numbers by using the **-b** option along with the **cat** command as follows −

```
$ cat -b filename
1   This is unix file....I created it for the first time.....
2   I'm going to save this content in this file.
$
```

Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is a simple example to see the information about the file created above −

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns −

- **First Column** − Represents the total number of lines in the file.
- **Second Column** − Represents the total number of words in the file.
- **Third Column** − Represents the total number of bytes in the file. This is the actual size of the file.
- **Fourth Column** − Represents the file name.

You can give multiple files and get information about those files at a time. Following is simple syntax −

```
$ wc filename1 filename2 filename3
```

Copying Files

To make a copy of a file use the **cp** command. The basic syntax of the command is −

```
$ cp source_file destination_file
```

Following is the example to create a copy of the existing file **filename**.

```
$ cp filename copyfile
$
```

You will now find one more file **copyfile** in your current directory. This file will exactly be the same as the original file **filename**.

Renaming Files

To change the name of a file, use the **mv** command. Following is the basic syntax −

```
$ mv old_file new_file
```

The following program will rename the existing file **filename** to **newfile**.

```
$ mv filename newfile
$
```

The **mv** command will move the existing file completely into the new file. In this case, you will find only **newfile** in your current directory.

Deleting Files

To delete an existing file, use the **rm** command. Following is the basic syntax −

```
$ rm filename
```

Caution − A file may contain useful information. It is always recommended to be careful while using this **Delete** command. It is better to use the **-i** option along with **rm** command.

Following is the example which shows how to completely remove the existing file **filename**.

```
$ rm filename
$
```

You can remove multiple files at a time with the command given below −

```
$ rm filename1 filename2 filename3
$
```

Standard Unix Streams

Under normal circumstances, every Unix program has three streams (files) opened for it when it starts up −

- **stdin** − This is referred to as the standard input and the associated file descriptor is 0. This is also represented as STDIN. The Unix program will read the default input from STDIN.
- **stdout** − This is referred to as the standard output and the associated file descriptor is 1. This is also represented as STDOUT. The Unix program will write the default output at STDOUT
- **stderr** − This is referred to as the standard error and the associated file descriptor is 2. This is also represented as STDERR. The Unix program will write all the error messages at STDERR.

Unix - Directory Management

In this chapter, we will discuss in detail about directory management in Unix.

A directory is a file the solo job of which is to store the file names and the related information. All the files, whether ordinary, special, or directory, are contained in directories.

Unix uses a hierarchical structure for organizing files and directories. This structure is often referred to as a directory tree. The tree has a single root node, the slash character (/), and all other directories are contained below it.

Home Directory

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command −

```
$cd ~
$
```

Here ~ indicates the home directory. Suppose you have to go in any other user's home directory, use the following command −

```
$cd ~username
$
```

To go in your last directory, you can use the following command −

```
$cd -
$
personal/res
```

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory −

```
$pwd
/user0/home/amrood

$
```

Listing Directories

To list the files in a directory, you can use the following syntax −

```
$ls dirname
```

Following is the example to list all the files contained in **/usr/local** directory −

```
$ls /usr/local

X11      bin      gimp     jikes     sbin
ace      doc      include  lib       share
atalk    etc      info     man       ami
```

Creating Directories

We will now understand how to create directories. Directories are created by the following command −

```
$mkdir dirname
```

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command −

```
$mkdir mydir
$
```

Creates the directory **mydir** in the current directory. Here is another example −

```
$mkdir /tmp/test-dir
$
```

This command creates the directory **test-dir** in the **/tmp** directory. The **mkdir** command produces no output if it successfully creates the requested directory.

If you give more than one directory on the command line, **mkdir** creates each of the directories. For example, −

```
$mkdir docs pub
$
```

Creates the directories docs and pub under the current directory.

Creating Parent Directories

We will now understand how to create parent directories. Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, **mkdir** issues an error message as follows −

```
$mkdir /tmp/amrood/test
mkdir: Failed to make directory "/tmp/amrood/test";
No such file or directory
$
```

In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example −

```
$mkdir -p /tmp/amrood/test
$
```

The above command creates all the required parent directories.

Removing Directories

Directories can be deleted using the **rmdir** command as follows −

```
$rmdir dirname
$
```

Note − To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can remove multiple directories at a time as follows −

```
$rmdir dirname1 dirname2 dirname3
$
```

The above command removes the directories dirname1, dirname2, and dirname3, if they are empty. The **rmdir** command produces no output if it is successful.

Changing Directories

You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below −

```
$cd dirname
$
```

Here, **dirname** is the name of the directory that you want to change to. For example, the command −

```
$cd /usr/local/bin
$
```

Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path −

```
$cd ../../home/amrood
$
```

Renaming Directories

The **mv (move)** command can also be used to rename a directory. The syntax is as follows −

```
$mv olddir newdir
$
```

You can rename a directory **mydir** to **yourdir** as follows −

```
$mv mydir yourdir
$
```

The directories . (dot) and .. (dot dot)

The **filename .** (dot) represents the current working directory; and the **filename ..** (dot dot) represents the directory one level above the current working directory, often referred to as the parent directory.

If we enter the command to show a listing of the current working directories/files and use the **-a option** to list all the files and the **-l option** to provide the long listing, we will receive the following result.

```
$ls -la
drwxrwxr-x    4    teacher   class   2048  Jul 16 17.56 .
drwxr-xr-x    60   root              1536  Jul 13 14:18 ..
----------    1    teacher   class   4210  May 1 08:27 .profile
-rwxr-xr-x    1    teacher   class   1948  May 12 13:42 memo
$
```

Unix - File Permission / Access Modes

In this chapter, we will discuss in detail about file permission and access modes in Unix. File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

- **Owner permissions** − The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows −

```
$ls -l /home/amrood
-rwxr-xr--  1 amrood   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 amrood   users 1024  Nov 2 00:10  mydir
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

- The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.

- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

Read

Grants the capability to read, i.e., view the contents of the file.

Write

Grants the capability to modify, or remove the content of the file.

Execute

User with execute permissions can run a file as a program.

Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

Write

Access means that the user can add or delete files from the directory.

Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

Sr.No.	Chmod operator & Description

1	+ Adds the designated permission(s) to a file or directory.
2	- Removes the designated permission(s) from a file or directory.
3	= Sets the designated permission(s).

Here's an example using **testfile**. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Here's how you can combine these commands on a single line −

```
$chmod o+wx,u-x,g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---

1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files −

- **chown** − The **chown** command stands for **"change owner"** and is used to change the owner of a file.
- **chgrp** − The **chgrp** command stands for **"change group"** and is used to change the group of a file.

Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows −

```
$ chown user filelist
```

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept −

```
$ chown amrood testfile
$
```

Changes the owner of the given file to the user **amrood**.

NOTE − The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows −

```
$ chgrp group filelist
```

The value of group can be the **name of a group** on the system or **the group ID (GID)** of a group on the system.

Following example helps you understand the concept −

```
$ chgrp special testfile
$
```

Changes the group of the given file to **special** group.

Unix – Environment

In this chapter, we will discuss in detail about the Unix environment. An important Unix concept is the **environment**, which is defined by environment variables. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

For example, first we set a variable TEST and then we access its value using the **echo** command −

```
$TEST="Unix Programming"
$echo $TEST
```

It produces the following result.

```
Unix Programming
```

Note that the environment variables are set without using the **$** sign but while accessing them we use the $ sign as prefix. These variables retain their values until we come out of the shell.

When you log in to the system, the shell undergoes a phase called **initialization** to set up the environment. This is usually a two-step process that involves the shell reading the following files −

- /etc/profile

- profile

The process is as follows −

- The shell checks to see whether the file **/etc/profile** exists.
- If it exists, the shell reads it. Otherwise, this file is skipped. No error message is displayed.
- The shell checks to see whether the file **.profile** exists in your home directory. Your home directory is the directory that you start out in after you log in.
- If it exists, the shell reads it; otherwise, the shell skips it. No error message is displayed.

As soon as both of these files have been read, the shell displays a prompt −

```
$
```

This is the prompt where you can enter commands in order to have them executed.

Note − The shell initialization process detailed here applies to all **Bourne** type shells, but some additional files are used by **bash** and **ksh**.

The .profile File

The file **/etc/profile** is maintained by the system administrator of your Unix machine and contains shell initialization information required by all users on a system.

The file **.profile** is under your control. You can add as much shell customization information as you want to this file. The minimum set of information that you need to configure includes −

- The type of terminal you are using.
- A list of directories in which to locate the commands.
- A list of variables affecting the look and feel of your terminal.

You can check your **.profile** available in your home directory. Open it using the vi editor and check all the variables set for your environment.

Setting the Terminal Type

Usually, the type of terminal you are using is automatically configured by either the **login** or **getty** programs. Sometimes, the auto configuration process guesses your terminal incorrectly.

If your terminal is set incorrectly, the output of the commands might look strange, or you might not be able to interact with the shell properly.

To make sure that this is not the case, most users set their terminal to the lowest common denominator in the following way −

```
$TERM=vt100
$
```

Setting the PATH

When you type any command on the command prompt, the shell has to locate the command before it can be executed.

The PATH variable specifies the locations in which the shell should look for commands. Usually the Path variable is set as follows −

```
$PATH=/bin:/usr/bin
$
```

Here, each of the individual entries separated by the colon character **(:)** are directories. If you request the shell to execute a command and it cannot find it in any of the directories given in the PATH variable, a message similar to the following appears −

```
$hello
hello: not found
$
```

There are variables like PS1 and PS2 which are discussed in the next section.

There are quite a few **escape sequences** that can be used as value arguments for PS1; try to limit yourself to the most critical so that the prompt does not overwhelm you with information.

Sr.No.	Escape Sequence & Description
1	**\t** Current time, expressed as HH:MM:SS
2	**\d** Current date, expressed as Weekday Month Date
3	**\n** Newline
4	**\s** Current shell environment
5	**\W** Working directory
6	**\w** Full path of the working directory
7	**\u** Current user’s username

8	**\h** Hostname of the current machine
9	**\#** Command number of the current command. Increases when a new command is entered
10	**\$** If the effective UID is 0 (that is, if you are logged in as root), end the prompt with the # character; otherwise, use the $ sign

Environment Variables

Following is the partial list of important environment variables. These variables are set and accessed as mentioned below −

Sr.No.	Variable & Description
1	**DISPLAY** Contains the identifier for the display that **X11** programs should use by default.
2	**HOME** Indicates the home directory of the current user: the default argument for the cd **built-in** command.
3	**IFS** Indicates the **Internal Field Separator** that is used by the parser for word splitting after expansion.
4	**LANG** LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is **pt_BR**, then the language is set to (Brazilian) Portuguese and the locale to Brazil.
5	**LD_LIBRARY_PATH**

	A Unix system with a dynamic linker, contains a colonseparated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.
6	**PATH** Indicates the search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
7	**PWD** Indicates the current working directory as set by the cd command.
8	**RANDOM** Generates a random integer between 0 and 32,767 each time it is referenced.
9	**SHLVL** Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session.
10	**TERM** Refers to the display type.
11	**TZ** Refers to Time zone. It can take values like GMT, AST, etc.
12	**UID** Expands to the numeric user ID of the current user, initialized at the shell startup.

Following is the sample example showing few environment variables −

```
$ echo $HOME
/root
]$ echo $DISPLAY

$ echo $TERM
xterm
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/home/amrood/bin:/usr/local/bin
```

```
$
```

Unix Basic Utilities - Printing, Email

In this chapter, we will discuss in detail about Printing and Email as the basic utilities of Unix. So far, we have tried to understand the Unix OS and the nature of its basic commands. In this chapter, we will learn some important Unix utilities that can be used in our day-to-day life.

Many versions of Unix include two powerful text formatters, **nroff** and **troff**.

The pr Command

The **pr** command does minor formatting of files on the terminal screen or for a printer. For example, if you have a long list of names in a file, you can format it onscreen into two or more columns.

Following is the syntax for the **pr** command −

```
pr option(s) filename(s)
```

The **pr** changes the format of the file only on the screen or on the printed copy; it doesn't modify the original file. Following table lists some **pr** options −

Sr.No.	Option & Description
1	**-k** Produces **k** columns of output
2	**-d** Double-spaces the output (not on all **pr** versions)
3	**-h "header"** Takes the next item as a report header
4	**-t** Eliminates the printing of header and the top/bottom margins
5	**-l PAGE_LENGTH** Sets the page length to PAGE_LENGTH (66) lines. The default number of lines of text is 56
6	**-o MARGIN**

	Offsets each line with MARGIN (zero) spaces
7	**-w PAGE_WIDTH** Sets the page width to PAGE_WIDTH (72) characters for multiple text-column output only

Before using **pr**, here are the contents of a sample file named food.

```
$cat food
Sweet Tooth
Bangkok Wok
Mandalay
Afghani Cuisine
Isle of Java
Big Apple Deli
Sushi and Sashimi
Tio Pepe's Peppers
.....
$
```

Let's use the **pr** command to make a two-column report with the header Restaurants −

```
$pr -2 -h "Restaurants" food
Nov  7  9:58 1997  Restaurants   Page 1

Sweet Tooth            Isle of Java
Bangkok Wok            Big Apple Deli
Mandalay               Sushi and Sashimi
Afghani Cuisine        Tio Pepe's Peppers
.....
$
```

The lp and lpr Commands

The command **lp** or **lpr** prints a file onto paper as opposed to the screen display. Once you are ready with formatting using the **pr** command, you can use any of these commands to print your file on the printer connected to your computer.

Your system administrator has probably set up a default printer at your site. To print a file named **food** on the default printer, use the **lp** or **lpr** command, as in the following example −

```
$lp food
request id is laserp-525  (1 file)
$
```

The **lp** command shows an ID that you can use to cancel the print job or check its status.

- If you are using the **lp** command, you can use the -n**Num** option to print Num number of copies. Along with the command **lpr**, you can use -**Num** for the same.

- If there are multiple printers connected with the shared network, then you can choose a printer using -d**printer** option along with lp command and for the same purpose you can use -P**printer** option along with lpr command. Here printer is the printer name.

The lpstat and lpq Commands

The **lpstat** command shows what's in the printer queue: request IDs, owners, file sizes, when the jobs were sent for printing, and the status of the requests.

Use **lpstat -o** if you want to see all output requests other than just your own. Requests are shown in the order they'll be printed −

```
$lpstat -o
laserp-573  john  128865  Nov 7  11:27  on laserp
laserp-574  grace  82744  Nov 7  11:28
laserp-575  john   23347  Nov 7  11:35
$
```

The **lpq** gives slightly different information than **lpstat -o** −

```
$lpq
laserp is ready and printing
Rank   Owner      Job  Files              Total Size
active john       573  report.ps          128865 bytes
1st    grace      574  ch03.ps ch04.ps      82744 bytes
2nd    john       575  standard input       23347 bytes
$
```

Here the first line displays the printer status. If the printer is disabled or running out of paper, you may see different messages on this first line.

The cancel and lprm Commands

The **cancel** command terminates a printing request from the **lp command**. The **lprm** command terminates all **lpr requests**. You can specify either the ID of the request (displayed by lp or lpq) or the name of the printer.

```
$cancel laserp-575
request "laserp-575" cancelled
$
```

To cancel whatever request is currently printing, regardless of its ID, simply enter cancel and the printer name −

```
$cancel laserp
request "laserp-573" cancelled
$
```

The **lprm** command will cancel the active job if it belongs to you. Otherwise, you can give job numbers as arguments, or use a **dash (-)** to remove all of your jobs −

```
$lprm 575
dfA575diamond dequeued
cfA575diamond dequeued
$
```

The **lprm** command tells you the actual filenames removed from the printer queue.

Sending Email

You use the Unix mail command to send and receive mail. Here is the syntax to send an email −

$mail [-s subject] [-c cc-addr] [-b bcc-addr] to-addr

Here are important options related to mail command −s

Sr.No.	Option & Description
1	**-s** Specifies subject on the command line.
2	**-c** Sends carbon copies to the list of users. List should be a commaseparated list of names.
3	**-b** Sends blind carbon copies to list. List should be a commaseparated list of names.

Following is an example to send a test message to admin@yahoo.com.

$mail -s "Test Message" admin@yahoo.com

You are then expected to type in your message, followed by **"control-D"** at the beginning of a line. To stop, simply type dot **(.)** as follows −

Hi,

This is a test
.
Cc:

You can send a complete file using a **redirect < operator** as follows −

$mail -s "Report 05/06/07" admin@yahoo.com < demo.txt

To check incoming email at your Unix system, you simply type email as follows −

$mail
no email

In this chapter, we will discuss in detail about pipes and filters in Unix. You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, put a vertical bar (|) on the command line between two commands.

When a program takes its input from another program, it performs some operation on that input, and writes the result to the standard output. It is referred to as a **filter**.

The grep Command

The grep command searches a file or files for lines that have a certain pattern. The syntax is −

```
$grep pattern file(s)
```

The name **"grep"** comes from the ed (a Unix line editor) command **g/re/p** which means "globally search for a regular expression and print all lines containing it".

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work −

```
$ls -l | grep "Aug"
-rw-rw-rw-   1 john  doc     11008 Aug  6 14:10 ch02
-rw-rw-rw-   1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-r--   1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

There are various options which you can use along with the **grep** command −

Sr.No.	Option & Description
1	**-v** Prints all lines that do not match pattern.
2	**-n** Prints the matched line and its line number.
3	**-l** Prints only the names of files with matching lines (letter "l")
4	**-c**

	Prints only the count of matching lines.
5	**-i** Matches either upper or lowercase.

Let us now use a regular expression that tells grep to find lines with **"carol"**, followed by zero or other characters abbreviated in a regular expression as ".*"), then followed by "Aug".−

Here, we are using the **-i** option to have case insensitive search −

```
$ls -l | grep -i "carol.*aug"
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

The sort Command

The **sort** command arranges lines of text alphabetically or numerically. The following example sorts the lines in the food file −

```
$sort food
Afghani Cuisine
Bangkok Wok
Big Apple Deli
Isle of Java

Mandalay
Sushi and Sashimi
Sweet Tooth
Tio Pepe's Peppers
$
```

The **sort** command arranges lines of text alphabetically by default. There are many options that control the sorting −

Sr.No.	Description
1	**-n** Sorts numerically (example: 10 will sort after 2), ignores blanks and tabs.
2	**-r** Reverses the order of sort.
3	**-f**

	Sorts upper and lowercase together.
4	**+x** Ignores first **x** fields when sorting.

More than two commands may be linked up into a pipe. Taking a previous pipe example using **grep**, we can further sort the files modified in August by the order of size.

The following pipe consists of the commands **ls**, **grep**, and **sort** −

```
$ls -l | grep "Aug" | sort +4n
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc       8515 Aug  6 15:30 ch07
-rw-rw-rw-  1 john  doc      11008 Aug  6 14:10 ch02
$
```

This pipe sorts all files in your directory modified in August by the order of size, and prints them on the terminal screen. The sort option +4n skips four fields (fields are separated by blanks) then sorts the lines in numeric order.

The pg and more Commands

A long output can normally be zipped by you on the screen, but if you run text through more or use the **pg** command as a filter; the display stops once the screen is full of text.

Let's assume that you have a long directory listing. To make it easier to read the sorted listing, pipe the output through **more** as follows −

```
$ls -l | grep "Aug" | sort +4n | more
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc       8515 Aug  6 15:30 ch07
-rw-rw-r--  1 john  doc     14827 Aug  9 12:40 ch03
            .
            .
            .
-rw-rw-rw-  1 john  doc     16867 Aug  6 15:56 ch05
--More--(74%)
```

The screen will fill up once the screen is full of text consisting of lines sorted by the order of the file size. At the bottom of the screen is the **more** prompt, where you can type a command to move through the sorted text.

Once you're done with this screen, you can use any of the commands listed in the discussion of the more program.

Starting a Process

When you start a process (run a command), there are two ways you can run it −

- Foreground Processes
- Background Processes

Foreground Processes

By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

You can see this happen with the **ls** command. If you wish to list all the files in your current directory, you can use the following command −

The process runs in the foreground, the output is directed to my screen, and if the **ls** command wants any input (which it does not), it waits for it from the keyboard.

While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

Background Processes

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

The simplest way to start a background process is to add an ampersand (**&**) at the end of the command.

```
$ls ch*.doc &
```

This displays all those files the names of which start with **ch** and end with **.doc** −

Listing Running Processes

It is easy to see your own processes by running the **ps** (process status) command as follows −

```
$ps
PID   TTY      TIME     CMD
18358  ttyp3   00:00:00  sh
18361  ttyp3   00:01:31  abiword
18789  ttyp3   00:00:00  ps
```

One of the most commonly used flags for ps is the **-f** (f for full) option, which provides more information as shown in the following example −

```
$ps -f
UID    PID  PPID C STIME    TTY   TIME CMD
amrood  6738 3662 0 10:23:03 pts/6 0:00 first_one
amrood  6739 3662 0 10:22:54 pts/6 0:00 second_one
amrood  3662 3657 0 08:10:53 pts/6 0:00 -ksh
amrood  6892 3662 4 10:51:50 pts/6 0:00 ps -f
```

Here is the description of all the fields displayed by **ps -f** command −

Sr.No.	Column & Description

1	**UID** User ID that this process belongs to (the person running it)
2	**PID** Process ID
3	**PPID** Parent process ID (the ID of the process that started it)
4	**C** CPU utilization of process
5	**STIME** Process start time
6	**TTY** Terminal type associated with the process
7	**TIME** CPU time taken by the process
8	**CMD** The command that started this process

There are other options which can be used along with **ps** command −

Sr.No.	Option & Description
1	**-a** Shows information about all users
2	**-x**

	Shows information about processes without terminals
3	**-u** Shows additional information like -f option
4	**-e** Displays extended information

Stopping Processes

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when the process is running in the foreground mode.

If a process is running in the background, you should get its Job ID using the **ps** command. After that, you can use the **kill** command to kill the process as follows −

```
$ps -f
UID      PID  PPID C STIME    TTY    TIME CMD
amrood   6738 3662 0 10:23:03 pts/6 0:00 first_one
amrood   6739 3662 0 10:22:54 pts/6 0:00 second_one
amrood   3662 3657 0 08:10:53 pts/6 0:00 -ksh
amrood   6892 3662 4 10:51:50 pts/6 0:00 ps -f
$kill 6738
Terminated
```

Here, the **kill** command terminates the **first_one** process. If a process ignores a regular kill command, you can use **kill -9** followed by the process ID as follows −

```
$kill -9 6738
Terminated
```

Parent and Child Processes

Each unix process has two ID numbers assigned to it: The Process ID (pid) and the Parent process ID (ppid). Each user process in the system has a parent process.

Most of the commands that you run have the shell as their parent. Check the **ps -f** example where this command listed both the process ID and the parent process ID.

Zombie and Orphan Processes

Normally, when a child process is killed, the parent process is updated via a **SIGCHLD** signal. Then the parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed before its child is killed. In this case, the "parent of all processes," the **init** process, becomes the new PPID (parent process ID). In some cases, these processes are called orphan processes.

When a process is killed, a **ps** listing may still show the process with a **Z** state. This is a zombie or defunct process. The process is dead and not being used. These processes are different from the orphan processes. They have completed execution but still find an entry in the process table.

Daemon Processes

Daemons are system-related background processes that often run with the permissions of root and services requests from other processes.

A daemon has no controlling terminal. It cannot open **/dev/tty**. If you do a **"ps -ef"** and look at the **tty** field, all daemons will have a **?** for the **tty**.

To be precise, a daemon is a process that runs in the background, usually waiting for something to happen that it is capable of working with. For example, a printer daemon waiting for print commands.

If you have a program that calls for lengthy processing, then it's worth to make it a daemon and run it in the background.

The top Command

The **top** command is a very useful tool for quickly showing processes sorted by various criteria.

It is an interactive diagnostic tool that updates frequently and shows information about physical and virtual memory, CPU usage, load averages, and your busy processes.

Here is the simple syntax to run top command and to see the statistics of CPU utilization by different processes −

```
$top
```

Job ID Versus Process ID

Background and suspended processes are usually manipulated via **job number (job ID)**. This number is different from the process ID and is used because it is shorter.

In addition, a job can consist of multiple processes running in a series or at the same time, in parallel. Using the job ID is easier than tracking individual processes.

Unix - Network Communication Utilities

In this chapter, we will discuss in detail about network communication utilities in Unix. When you work in a distributed environment, you need to communicate with remote users and you also need to access remote Unix machines.

There are several Unix utilities that help users compute in a networked, distributed environment. This chapter lists a few of them.

The ping Utility

The **ping** command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not.

The ping command is useful for the following −

- Tracking and isolating hardware and software problems.
- Determining the status of the network and various foreign hosts.
- Testing, measuring, and managing networks.

Syntax

Following is the simple syntax to use the ftp command −

$ping hostname or ip-address

The above command starts printing a response after every second. To come out of the command, you can terminate it by pressing **CNTRL + C** keys.

Example

Following is an example to check the availability of a host available on the network −

```
$ping google.com
PING google.com (74.125.67.100) 56(84) bytes of data.
64 bytes from 74.125.67.100: icmp_seq = 1 ttl = 54 time = 39.4 ms
64 bytes from 74.125.67.100: icmp_seq = 2 ttl = 54 time = 39.9 ms
64 bytes from 74.125.67.100: icmp_seq = 3 ttl = 54 time = 39.3 ms
64 bytes from 74.125.67.100: icmp_seq = 4 ttl = 54 time = 39.1 ms
64 bytes from 74.125.67.100: icmp_seq = 5 ttl = 54 time = 38.8 ms
--- google.com ping statistics ---
22 packets transmitted, 22 received, 0% packet loss, time 21017ms
rtt min/avg/max/mdev = 38.867/39.334/39.900/0.396 ms
$
```

If a host does not exist, you will receive the following output −

```
$ping giiiiigle.com
ping: unknown host giiiiigle.com
$
```

Managing Server Firewall

Firewall is responsible for protecting a server / network / application from unauthorized access.Types ⇒

1. Software Firewall ⇒ Firewall is built into the Operating System.
2. Hardware Firewall - Cisco PIX

Linux ⇒ **ipchains** / **iptables** / **firewalld** ⇒ Packet Filtering firewall

The linux kernel includes **netfilter**, a framework for network traffic operations such as packet filtering, NAT, Port translati

Firewalld ⇒ is a dynamic firewall manager. It is a front-end to the nftables framework. Until the introduction of nftables, firewalld used the iptables command to configurenetfilter directly.

Network File Sharing Services

Network File Sharing Services ⇒ NFS / FTP / SMB

NFS ⇒ Network File System. It allows remote hosts to mount file systems over a network and interact with those filesystems as though they are mounted locally. This enables us toconsolidate resources onto centralized servers in a network.

NFS shares are generally defined in **/etc/exports** file or **/etc/exports.d/*.exports**

RHEL8 supports NFS version 3(NFSv3) and NFS version 4(**NFSv4**).NFSv2 is not supported in RHEL8.

The default NFS version in RHEL8 is 4.2NFS exports are by default read-only

Package name : **nfs-utils**

Services required by NFS :
nfsd ⇒ The NFS server kernel module that services requests for shared NFS filesystems
rpcbind ⇒ Accepts port reservations from local rpc services.
rpc.nfsd ⇒ This corresponds to the nfs-server service

How can we share a directory / file system using NFS?

vim /etc/exports

<dir_name/filesystem_name> FQDN/DOMAIN/IP/NETWORK(access_mode)
/storage 192.168.1.0/24(ro,sync)
/confidential 192.168.1.100/24(rw,sync)

In this chapter, we will understand how the vi Editor works in Unix. There are many ways to edit files in Unix. Editing files using the screen-oriented text editor **vi** is one of the best ways. This editor enables you to edit lines in context with other lines in the file.

An improved version of the vi editor which is called the **VIM** has also been made available now. Here, VIM stands for **Vi IM**proved.

vi is generally considered the de facto standard in Unix editors because −

- It's usually available on all the flavors of Unix system.
- Its implementations are very similar across the board.
- It requires very few resources.
- It is more user-friendly than other editors such as the **ed** or the **ex**.

You can use the **vi** editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

Starting the vi Editor

The following table lists out the basic commands to use the vi editor −

Sr.No.	Command & Description
1	**vi filename** Creates a new file if it already does not exist, otherwise opens an existing file.
2	**vi -R filename** Opens an existing file in the read-only mode.
3	**view filename** Opens an existing file in the read-only mode.

Following is an example to create a new file **testfile** if it already does not exist in the current working directory −

```
$vi testfile
```

The above command will generate the following output −

```
|
~
~
~
~
~
```

~
~
~
~
~
~
~
"testfile" [New File]

You will notice a **tilde** (~) on each line following the cursor. A tilde represents an unused line. If a line does not begin with a tilde and appears to be blank, there is a space, tab, newline, or some other non-viewable character present.

You now have one open file to start working on. Before proceeding further, let us understand a few important concepts.

Operation Modes

While working with the vi editor, we usually come across the following two modes −

- **Command mode** − This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

vi always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.

Hint − If you are not sure which mode you are in, press the Esc key twice; this will take you to the command mode. You open a file using the vi editor. Start by typing some characters and then come to the command mode to understand the difference.

Getting Out of vi

The command to quit out of vi is **:q**. Once in the command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or use **:wq** and return.

The easiest way to **save your changes and exit vi** is with the ZZ command. When you are in the command mode, type **ZZ**. The **ZZ** command works the same way as the **:wq** command.

If you want to specify/state any particular name for the file, you can do so by specifying it after the **:w**. For example, if you wanted to save the file you were working on as another filename called **filename2**, you would type **:w filename2** and return.

Moving within a File

To move around within a file without affecting your text, you must be in the command mode (press Esc twice). The following table lists out a few commands you can use to move around one character at a time −

Sr.No.	Command & Description
1	**k** Moves the cursor up one line
2	**j** Moves the cursor down one line
3	**h** Moves the cursor to the left one character position
4	**l** Moves the cursor to the right one character position

The following points need to be considered to move within a file −

- vi is case-sensitive. You need to pay attention to capitalization when using the commands.
- Most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2j** moves the cursor two lines down the cursor location.

There are many other ways to move within a file in vi. Remember that you must be in the command mode (**press Esc twice**). The following table lists out a few commands to move around the file −

Given below is the list of commands to move around the file.

Control Commands

The following commands can be used with the Control Key to performs functions as given in the table below −

Given below is the list of control commands.

Editing Files

To edit the file, you need to be in the insert mode. There are many ways to enter the insert mode from the command mode −

Sr.No.	Command & Description
1	**i** Inserts text before the current cursor location

2	**I** Inserts text at the beginning of the current line
3	**a** Inserts text after the current cursor location
4	**A** Inserts text at the end of the current line
5	**o** Creates a new line for text entry below the cursor location
6	**O** Creates a new line for text entry above the cursor location

Deleting Characters

Here is a list of important commands, which can be used to delete characters and lines in an open file −

Sr.No.	Command & Description
1	**x** Deletes the character under the cursor location
2	**X** Deletes the character before the cursor location
3	**dw** Deletes from the current cursor location to the next word
4	**d^** Deletes from the current cursor position to the beginning of the line
5	**d$**

	Deletes from the current cursor position to the end of the line
6	**D** Deletes from the cursor position to the end of the current line
7	**dd** Deletes the line the cursor is on

As mentioned above, most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2x** deletes two characters under the cursor location and **2dd** deletes two lines the cursor is on.

It is recommended that the commands are practiced before we proceed further.

Change Commands

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands −

Sr.No.	Command & Description
1	**cc** Removes the contents of the line, leaving you in insert mode.
2	**cw** Changes the word the cursor is on from the cursor to the lowercase **w** end of the word.
3	**r** Replaces the character under the cursor. vi returns to the command mode after the replacement is entered.
4	**R** Overwrites multiple characters beginning with the character currently under the cursor. You must use **Esc** to stop the overwriting.
5	**s** Replaces the current character with the character you type. Afterward, you are left in the insert mode.

6	**S** Deletes the line the cursor is on and replaces it with the new text. After the new text is entered, vi remains in the insert mode.

Copy and Paste Commands

You can copy lines or words from one place and then you can paste them at another place using the following commands −

Sr.No.	Command & Description
1	**yy** Copies the current line.
2	**yw** Copies the current word from the character the lowercase w cursor is on, until the end of the word.
3	**p** Puts the copied text after the cursor.
4	**P** Puts the yanked text before the cursor.

Advanced Commands

There are some advanced commands that simplify day-to-day editing and allow for more efficient use of vi −

Given below is the list advanced commands.

Word and Character Searching

The vi editor has two kinds of searches: **string** and **character**. For a string search, the **/** and **?** commands are used. When you start these commands, the command just typed will be shown on the last line of the screen, where you type the particular string to look for.

These two commands differ only in the direction where the search takes place −

- The / command searches forwards (downwards) in the file.
- The **?** command searches backwards (upwards) in the file.

The **n** and **N** commands repeat the previous search command in the same or the opposite direction, respectively. Some characters have special meanings. These characters must be preceded by a backslash (\) to be included as part of the search expression.

Sr.No.	Character &Description
1	^ Searches at the beginning of the line (Use at the beginning of a search expression).
2	. Matches a single character.
3	* Matches zero or more of the previous character.
4	$ End of the line (Use at the end of the search expression).
5	[Starts a set of matching or non-matching expressions.
6	< This is put in an expression escaped with the backslash to find the ending or the beginning of a word.
7	> This helps see the '<' character description above.

The character search searches within one line to find a character entered after the command. The **f** and **F** commands search for a character on the current line only. **f** searches forwards and **F** searches backwards and the cursor moves to the position of the found character.

The **t** and **T** commands search for a character on the current line only, but for **t**, the cursor moves to the position before the character, and **T** searches the line backwards to the position after the character.

FTP SERVER:-

FTP ⇒ File Transfer Protocol. Used for transferring files (upload/download) across systemsover the network.

Ports : **20** (Data) **21** (Connection)

Package : **vsftpd** [Very Secure FTP Daemon]Configuration File ⇒
/etc/vsftpd/vsftpd.conf

FTP user ⇒ 1. Registered User 2. Anonymous User [Username ⇒ ftp or anonymous]

Anonymous users can download files but can NOT upload files by default.

Set Commands

You can change the look and feel of your vi screen using the following **:set** commands. Once you are in the command mode, type **:set** followed by any of the following commands.

Sr.No.	Command & Description
1	**:set ic** Ignores the case when searching
2	**:set ai** Sets autoindent
3	**:set noai** Unsets autoindent
4	**:set nu** Displays lines with line numbers on the left side
5	**:set sw** Sets the width of a software tabstop. For example, you would set a shift width of 4 with this command — **:set sw = 4**
6	**:set ws** If wrapscan is set, and the word is not found at the bottom of the file, it will try searching for it at the beginning

7	**:set wm** If this option has a value greater than zero, the editor will automatically "word wrap". For example, to set the wrap margin to two characters, you would type this: **:set wm = 2**
8	**:set ro** Changes file type to "read only"
9	**:set term** Prints terminal type
10	**:set bf** Discards control characters from input

Running Commands

The vi has the capability to run commands from within the editor. To run a command, you only need to go to the command mode and type **:!** command.

For example, if you want to check whether a file exists before you try to save your file with that filename, you can type **:! ls** and you will see the output of **ls** on the screen.

You can press any key (or the command's escape sequence) to return to your vi session.

Replacing Text

The substitution command (**:s/**) enables you to quickly replace words or groups of words within your files. Following is the syntax to replace text −

```
:s/search/replace/g
```

The **g** stands for globally. The result of this command is that all occurrences on the cursor's line are changed.

Important Points to Note

The following points will add to your success with vi −

- You must be in command mode to use the commands. (Press Esc twice at any time to ensure that you are in command mode.)
- You must be careful with the commands. These are case-sensitive.
- You must be in insert mode to enter text.