**Operating Sytem**

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

* Memory Management
* Processor Management
* Device Management
* File Management
* Security
* Control over system performance
* Job accounting
* Error detecting aids
* Coordination between other software and users

# Types of Operating Systems

Following are some of the most widely used types of Operating system.

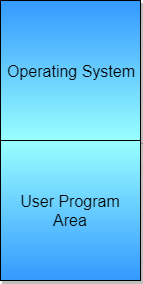
1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Realtime Operating System
8. Handheld System

## Simple Batch Systems

* In this type of system, there is **no direct interaction between user and the computer**.
* The user has to submit a job (written on cards or tape) to a computer operator.
* Then computer operator places a batch of several jobs on an input device.
* Jobs are batched together by type of languages and requirement.
* Then a special program, the monitor, manages the execution of each program in the batch.
* The monitor is always in the main memory and available for execution.

#### Advantages of Simple Batch Systems

1. No interaction between user and computer.
2. No mechanism to prioritise the processes.

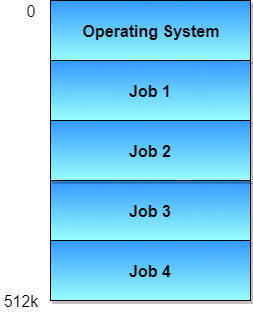


## Multiprogramming Batch Systems

* In this the operating system picks up and begins to execute one of the jobs from memory.
* Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
* Jobs in the memory are always less than the number of jobs on disk(Job Pool).
* If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
* In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
* In Multiprogramming system, CPU will never be idle and keeps on processing.

**Time Sharing Systems** are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on **minimizing the response time**, while in multiprogramming the prime focus is to maximize the CPU usage.



## Multiprocessor Systems

A Multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

#### Advantages of Multiprocessor Systems

1. Enhanced performance
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

## Desktop Systems

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither **multiuser** nor **multitasking**. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called **Desktop Systems** and include PCs running Microsoft Windows and the Apple Macintosh. Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.

**Microcomputers** were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

## Distributed Operating System

The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.

These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.

#### Advantages Distributed Operating System

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

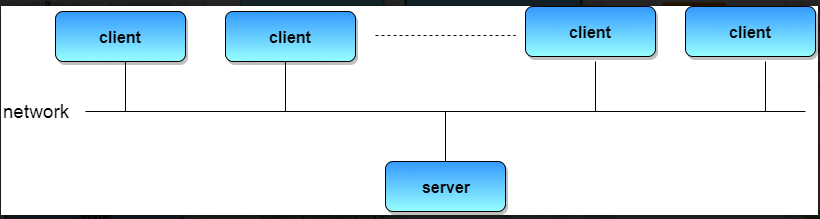
### Types of Distributed Operating Systems

Following are the two types of distributed operating systems used:

1. Client-Server Systems
2. Peer-to-Peer Systems

#### Client-Server Systems

**Centralized systems** today act as **server systems** to satisfy requests generated by **client systems**. The general structure of a client-server system is depicted in the figure below:



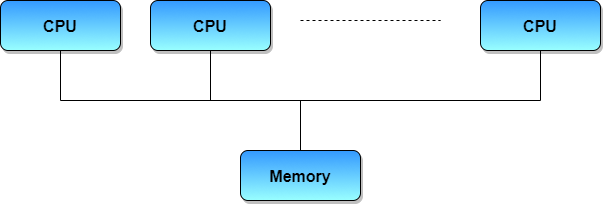
Server Systems can be broadly categorized as: **Compute Servers** and **File Servers**.

* **Compute Server systems**, provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
* **File Server systems**, provide a file-system interface where clients can create, update, read, and delete files.

#### Peer-to-Peer Systems

The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems. When PCs were introduced in the 1970s, they were designed for **personal** use and were generally considered standalone computers. With the beginning of widespread public use of the Internet in the 1990s for electronic mail and FTP, many PCs became connected to computer networks.

In contrast to the **Tightly Coupled** systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled systems ( or distributed systems). The general structure of a client-server system is depicted in the figure below:



## Clustered Systems

* Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
* Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
* The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
* Clustering is usually performed to provide **high availability**.
* A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.
* **Asymmetric Clustering -** In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.
* **Symmetric Clustering -** In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.
* **Parallel Clustering -** Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system's usage and it's features should expand greatly as **Storage Area Networks(SANs)**. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

## Real Time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.**

While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**.

## Handheld Systems

Handheld systems include **Personal Digital Assistants(PDAs)**, such as Palm-Pilots or Cellular Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

* Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
* Currently, many handheld devices do **not use virtual memory** techniques, thus forcing program developers to work within the confines of limited physical memory.
* Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
* The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

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.
* **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.

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.

Who is Logged in?

Sometime you might be interested to know who is logged in to the computer at the same time.

There are three commands available to get you this information, based on how much you wish to know about the other users: **users**, **who**, and **w**.

$ users

amrood bablu qadir

$ who

amrood ttyp0 Oct 8 14:10 (limbo)

bablu ttyp2 Oct 4 09:08 (calliope)

qadir ttyp4 Oct 8 12:09 (dent)

$

Try the **w** command on your system to check the output. This lists down information associated with the users logged in the system.

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.

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.

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.

|  |  |
| --- | --- |
| **Sr.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.

**Directory Management**

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 -

$

Absolute/Relative Pathnames

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.

Following are some examples of absolute filenames.

/etc/passwd

/users/sjones/chem/notes

/dev/rdsk/Os3

A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood's home directory, some pathnames might look like this −

chem/notes

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

$

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.

SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file **/etc/shadow**.

As a regular user, you do not have **read** or **write** access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file **/etc/shadow**.

Additional permissions are given to programs via a mechanism known as the **Set User ID (SUID)** and **Set Group ID (SGID)** bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners’ **execute** permission normally resides.

For example, the command −

$ ls -l /usr/bin/passwd

-r-sr-xr-x 1 root bin 19031 Feb 7 13:47 /usr/bin/passwd\*

$

Shows that the SUID bit is set and that the command is owned by the root. A capital letter **S** in the execute position instead of a lowercase **s** indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users −

* The owner of the sticky directory
* The owner of the file being removed
* The super user, root

To set the SUID and SGID bits for any directory try the following command −

$ chmod ug+s dirname

$ ls -l

drwsr-sr-x 2 root root 4096 Jun 19 06:45 dirname

$

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.