Chapter 1. The UNIX Operating System

Introduction

This chapter introduces you to the UNIX operating system. We first look at what is an operating system and then proceed to discuss the different features of UNIX that have made it a popular operating system.

Objectives

- What is an operating system (OS)?
- Features of UNIX OS
- A Brief History of UNIX OS, POSIX and Single Unix Specification (SUS)

1. What is an operating system (OS)?

An operating system (OS) is a resource manager. It takes the form of a set of software routines that allow users and application programs to access system resources (e.g. the CPU, memory, disks, modems, printers, network cards etc.) in a **safe, efficient** and **abstract** way.

For example, an OS ensures **safe** access to a printer by allowing only one application program to send data directly to the printer at any one time. An OS encourages **efficient** use of the CPU by suspending programs that are waiting for I/O operations to complete to make way for programs that can use the CPU more productively. An OS also provides convenient **abstractions** (such as files rather than disk locations) which isolate application programmers and users from the details of the underlying hardware.

UNIX Operating system allows complex tasks to be performed with a few keystrokes. It doesn't tell or warn the user about the consequences of the command.

Kernighan and Pike (The UNIX Programming Environment) lamented long ago that "as the UNIX system has spread, the fraction of its users who are skilled in its application has decreased." However, the capabilities of UNIX are limited only by your imagination.

2. Features of UNIX OS

Several features of UNIX have made it popular. Some of them are:

Portable

UNIX can be installed on many hardware platforms. Its widespread use can be traced to the decision to develop it using the C language.

Multiuser

The UNIX design allows multiple users to concurrently share hardware and software

Multitasking

UNIX allows a user to run more than one program at a time. In fact more than one program can be

running in the background while a user is working foreground.

Networking
While UNIX was developed to be an interactive, multiuser, multitasking system, networking is also incorporated into the heart of the operating system. Access to another system uses a standard communications protocol known as Transmission Control Protocol/Internet Protocol (TCP/IP).

Organized File System
UNIX has a very organized file and directory system that allows users to organize and maintain files.

Device Independence
UNIX treats input/output devices like ordinary files. The source or destination for file input and output is easily controlled through a UNIX design feature called redirection.

Utilities
UNIX provides a rich library of utilities that can be use to increase user productivity.

3. A Brief History of UNIX

In the late 1960s, researchers from General Electric, MIT and Bell Labs launched a joint project to develop an ambitious multi-user, multi-tasking OS for mainframe computers known as MULTICS (Multiplexed Information and Computing System). MULTICS failed, but it did inspire Ken Thompson, who was a researcher at Bell Labs, to have a go at writing a simpler operating system himself. He wrote a simpler version of MULTICS on a PDP7 in assembler and called his attempt UNICS (Uniplexed Information and Computing System). Because memory and CPU power were at a premium in those days, UNICS (eventually shortened to UNIX) used short commands to minimize the space needed to store them and the time needed to decode them - hence the tradition of short UNIX commands we use today, e.g. `ls, cp, rm, mv` etc.
Ken Thompson then teamed up with Dennis Ritchie, the author of the first C compiler in 1973. They rewrote the UNIX kernel in C - this was a big step forwards in terms of the system's portability - and released the Fifth Edition of UNIX to universities in 1974. The Seventh Edition, released in 1978, marked a split in UNIX development into two main branches: SYSV (System 5) and BSD (Berkeley Software Distribution). BSD arose from the University of California at Berkeley where Ken Thompson spent a sabbatical year. Its development was continued by students at Berkeley and other research institutions. SYSV was developed by AT&T and other commercial companies. UNIX flavors based on SYSV have traditionally been more conservative, but better supported than BSD-based flavors.
Until recently, UNIX standards were nearly as numerous as its variants. In early days, AT&T published a document called System V Interface Definition (SVID). X/OPEN (now The Open Group), a consortium of vendors and users, had one too, in the X/Open Portability Guide (XPG). In the US, yet another set of standards, named Portable Operating System Interface for Computer Environments (POSIX), were developed at the behest of the Institution of Electrical and Electronics Engineers (IEEE).
In 1998, X/OPEN and IEEE undertook an ambitious program of unifying the two standards. In 2001, this joint initiative resulted in a single specification called the Single UNIX Specification, Version 3 (SUSV3), that is also known as IEEE 1003.1:2001 (POSIX.1). In 2002, the International Organization for Standardization (ISO) approved SUSV3 and IEEE 1003.1:2001.

Some of the commercial UNIX based on system V are:

- IBM's AIX

- Hewlett-Packard's HPUX
- SCO's Open Server Release 5
- Silicon Graphics' IRIS
- DEC's Digital UNIX
- Sun Microsystems' Solaris 2

Some of the commercial UNIX based on BSD are:

- SunOS 4.1.X (now Solaris)
- DEC's Ultris
- BSD/OS, 4.4BSD

Some Free UNIX are:

- Linux, written by Linus Torvalds at University of Helsinki in Finland.
- FreeBSD and NetBSD, a derivative of 4.4BSD

Conclusion

In this chapter we defined an operating system. We also looked at history of UNIX and features of UNIX that make it a popular operating system. We also discussed the convergence of different flavors of UNIX into Single Unix Specification (SUS) and Portable Operating System Interface for Computing Environments (POSIX).

Chapter 2. The UNIX Architecture and Command Usage

Introduction

In order to understand the subsequent chapters, we first need to understand the architecture of UNIX and the concept of division of labor between two agencies viz., the shell and the kernel. This chapter introduces the architecture of UNIX. Next we discuss the rich collection of UNIX command set, with a specific discussion of command structure and usage of UNIX commands. We also look at the man command, used for obtaining online help on any UNIX command. Sometimes the keyboard sequences don't work, in which case, you need to know what to do to fix them. Final topic of this chapter is troubleshooting some terminal problems.

Objectives

2. The UNIX Architecture
3. Locating Commands
4. Internal and External Commands
5. Command Structure and usage
6. Flexibility of Command Usage
7. The man Pages, apropos and whatis
8. Troubleshooting the terminal problems

- ## The UNIX Architecture

Users

UNIX architecture comprises of two major components viz., the shell and the kernel. The kernel interacts with the machine's hardware and the shell with the user.

The kernel is the core of the operating system. It is a collection of routines written in C. It is loaded into memory when the system is booted and communicates directly with the hardware. User programs that need to access the hardware use the services of the kernel via use of system calls and the kernel performs the job on behalf of the user. Kernel is also responsible for managing system's memory, schedules processes, decides their priorities.

The shell performs the role of command interpreter. Even though there's only one kernel running on the system, there could be several shells in action, one for each user who's logged in. The shell is responsible for interpreting the meaning of metacharacters if any, found on the command line before dispatching the command to the kernel for execution.

The File and Proces

A file is an array of bytes that stores information. It is also related to another file in the sense that both belong to a single hierarchical directory structure.

A process is the second abstraction UNIX provides. It can be treated as a time image of an executable file. Like files, processes also belong to a hierarchical structure. We will be discussing

the processes in detain in a subsequent chapter.

2. Locating Files

All UNIX commands are single words like ls, cd, cat, etc. These names are in lowercase. These commands are essentially *files* containing programs, mainly written in C. Files are stored in directories, and so are the binaries associated with these commands. You can find the location of an executable program using type command:

```
$ type ls
ls is /bin/ls
```

This means that when you execute ls command, the shell locates this file in /bin directory and makes arrangements to execute it.

The Path

The sequence of directories that the shell searches to look for a command is specified in its own PATH variable. These directories are colon separated. When you issue a command, the shell searches this list in the sequence specified to locate and execute it.

3. Internal and External Commands

Some commands are implemented as part of the shell itself rather than separate executable files. Such commands that are built-in are called internal commands. If a command exists both as an internal command of the shell as well as an external one (in /bin or /usr/bin), the shell will accord top priority to its own internal command with the same name. Some built-in commands are echo, pwd, etc.

4. Command Structure

UNIX commands take the following general form:

```
verb [options] [arguments]
```

where verb is the command name that can take a set of optional options and one or more optional arguments.

Commands, options and arguments have to be separated by spaces or tabs to enable the shell to interpret them as words. A contiguous string of spaces and tabs together is called a whitespace. The shell compresses multiple occurrences of whitespace into a single whitespace.

Options

An option is preceded by a minus sign (-) to distinguish it from filenames.

Example: $ ls –l

There must not be any whitespaces between – and l. Options are also arguments, but given a special name because they are predetermined. Options can be normally compined with only one – sign. i.e., instead of using

```
$ ls –l –a –t
```

we can as well use,

```
$ ls –lat
```

Because UNIX was developed by people who had their own ideas as to what options should look like, there will be variations in the options. Some commands use + as an option prefix instead of -.

Filename Arguments

Many UNIX commands use a filename as argument so that the command can take input from the file. If a command uses a filename as argument, it will usually be the last argument, after all

options.

Example: cp file1 file2 file3 dest_dir
rm file1 file2 file3

The command with its options and argumens is known as the command line, which is considered as complete after *[Enter]* key is pressed, so that the entire line is fed to the shell as its input for interpretation and execution.

Exceptions

Some commands in UNIX like pwd do not take any options and arguments. Some commands like who may or may not be specified with arguments. The ls command can run without arguments (ls), with only options (ls –l), with only filenames (ls f1 f2), or using a combination of both (ls –l f1 f2). Some commands compulsorily take options (cut). Some commands like grep, sed can take an expression as an argument, or a set of instructions as argument.

5. Flexibility of Command Usage

UNIX provides flexibility in using the commands. The following discussion looks at how permissive the shell can be to the command usage.

Combining Commands

Instead of executing commands on separate lines, where each command is processed and executed before the next could be entered, UNIX allows you to specify more than one command in the single command line. Each command has to be separated from the other by a ; (semicolon).

```
wc sample.txt ; ls –l sample.txt
```

You can even group several commands together so that their combined output is redirected to a file.

```
(wc sample.txt ; ls –l sample.txt) > newfile
```

When a command line contains a semicolon, the shell understands that the command on each side of it needs to be processed separately. Here ; is known as a metacharacter.

Note: When a command overflows into the next line or needs to be split into multiple lines, just press enter, so that the secondary prompt (normally >) is displayed and you can enter the remaining part of the command on the next line.

Entering a Command before previous command has finished

You need not have to wait for the previous command to finish before you can enter the next command. Subsequent commands entered at the keyboard are stored in a buffer (a temporary storage in memory) that is maintained by the kernel for all keyboard input. The next command will be passed on to the shell for interpretation after the previous command has completed its execution.

6. man: Browsing The Manual Pages Online

UNIX commands are rather cryptic. When you don’t remember what options are supported by a command or what its syntax is, you can always view man (short for manual) pages to get online help. The man command displays online documentation of a specified command.

A pager is a program that displays one screenful information and pauses for the user to view the contents. The user can make use of internal commands of the pager to scroll up and scroll down the information. The two popular pagers are more and less. more is the Berkeley’s pager, which is a superior alternative to original pg command. less is the standard pager used on Linux systems. less if modeled after a popular editor called vi and is more powerful than more as it provides vi-like

navigational and search facilities. We can use pagers with commands like ls | more. The man command is configured to work with a pager.

7. Understanding The man Documentation

The man documentation is organized in eight (08) sections. Later enhancements have added subsections like 1C, 1M, 3N etc.) References to other sections are reflected as SEE ALSO section of a man page.

When you use man command, it starts searching the manuals starting from section 1. If it locates a keyword in one section, it won't continue the search, even if the keyword occurs in another section. However, we can provide the section number additionally as argument for man command.

For example, passwd appears in section 1 and section 4. If we want to get documentation of passwd in section 4, we use,

$ man 4 passwd OR $ man –s4 passwd (on Solaris)

Understanding a man Page

A typical man page for wc command is shown below:

A man page is divided into a number of compulsory and optional sections. Every command doesn't need all sections, but the first three (NAME, SYNOPSIS and DESCRIPTION) are generally seen in all man pages. NAME presents a one-line introduction of the command. SYNOPSIS shows the syntax used by the command and DESCRIPTION provides a detailed description.

The SYNOPSIS follows certain conventions and rules:

- If a command argument is enclosed in rectangular brackets, then it is optional; otherwise, the argument is required.
- The ellipsis (a set if three dots) implies that there can be more instances of the preceding word.
- The | means that only one of the options shows on either side of the pipe can be used.

All the options used by the command are listed in OPTIONS section. There is a separate section named EXIT STATUS which lists possible error conditions and their numeric representation.

Note: You can use man command to view its own documentation ($ man man). You can also set the pager to use with man ($ PAGER=less ; export PAGER). To understand which pager is being used by man, use $ echo $PAGER.

The following table shows the organization of man documentation.

Section	Subject (SVR4)	Subject (Linux)
1	User programs	User programs
2	Kernel's system calls	Kernel's system calls
3	Library functions	Library functions
4	Administrative file formats	Special files (in /dev)
5	Miscellaneous	Administrative file formats

6	Games	Games
7	Special files (in /dev)	Macro packages and conventions
8	Administration commands	Administration commands

8. Further Help with man –k, apropos and whatis

man –k: Searches a summary database and prints one-line description of the command.
Example:

```
$ man –k awk
awk  awk(1)    -pattern scanning and processing language
nawk nawk(1)   -pattern scanning and processing language
```

apropos: lists the commands and files associated with a keyword.
Example:

```
$ apropos FTP
ftp  ftp(1)         -file transfer program
ftpd in.ftpd(1m)    -file transfer protocol server
ftpusers  ftpusers(4) -file listing users to be disallowed
                        ftp login privileges
```

whatis: lists one-liners for a command.
Example:

```
$ whatis cp
cp   cp(1)          -copy files
```

9. When Things Go Wrong

Terminals and keyboards have no uniform behavioral pattern. Terminal settings directly impact the keyboard operation. If you observe a different behavior from that expected, when you press certain keystrokes, it means that the terminal settings are different. In such cases, you should know which keys to press to get the required behavior. The following table lists keyboard commands to try when things go wrong.

Keystroke or command	**Function**
[Ctrl-h]	Erases text
[Ctrl-c] or Delete	Interrupts a command
[Ctrl-d]	Terminates login session or a program that expects its input from keyboard
[Ctrl-s]	Stops scrolling of screen output and locks keyboard
[Ctrl-q]	Resumes scrolling of screen output and unlocks keyboard
[Ctrl-u]	Kills command line without executing it
[Ctrl-\]	Kills running program but creates a core file containing the memory image of the program
[Ctrl-z]	Suspends process and returns shell prompt; use **fg** to resume job
[Ctrl-j]	Alternative to *[Enter]*
[Ctrl-m]	Alternative to *[Enter]*

stty sane	Restores terminal to normal status

Conclusion

In this chapter, we looked at the architecture of UNIX and the division of labor between two agencies viz., the shell and the kernel. We also looked at the structure and usage of UNIX commands. The man documentation will be the most valuable source of documentation for UNIX commands. Also, when the keyboard sequences won't sometimes work as expected because of different terminal settings. We listed the possible remedial keyboard sequences when that happens.

Chapter 3. The File System

Introduction

In this chapter we will look at the file system of UNIX. We also look at types of files their significance. We then look at two ways of specifying a file viz., with absolute pathnames and relative pathnames. A discussion on commands used with directory files viz., cd, pwd, mkdir, rmdir and ls will be made. Finally we look at some of the important directories contained under UNIX file system.

Objectives

9. Types of files
10. UNIX Filenames
11. Directories and Files
12. Absolute and Relative Pathnames
13. pwd – print working directory
14. cd – change directory
15. mkdir – make a directory
16. rmdir – remove directory
17. The PATH environmental variable
18. ls – list directory contents
19. The UNIX File System

1. Types of files

A simple description of the UNIX system is this:
"On a UNIX system, everything is a file; if something is not a file, it is a process."
A UNIX system makes no difference between a file and a directory, since a directory is just a file containing names of other files. Programs, services, texts, images, and so forth, are all files. Input and output devices, and generally all devices, are considered to be files, according to the system.

Most files are just files, called *regular* files; they contain normal data, for example text files, executable files or programs, input for or output from a program and so on.
While it is reasonably safe to suppose that everything you encounter on a UNIX system is a file, there are some exceptions.
Directories: files that are lists of other files.
Special files or Device Files: All devices and peripherals are represented by files. To read or write a device, you have to perform these operations on its associated file. Most special files are in /dev.
Links: a system to make a file or directory visible in multiple parts of the system's file tree.
(Domain) sockets: a special file type, similar to TCP/IP sockets, providing inter−process networking protected by the file system's access control.
Named pipes: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

Ordinary (Regular) File

This is the most common file type. An ordinary file can be either a text file or a binary file.
A text file contains only printable characters and you can view and edit them. All C and Java program sources, shell scripts are text files. Every line of a text file is terminated with the *newline* character.
A binary file, on the other hand, contains both printable and nonprintable characters that cover the entire ASCII range. The object code and executables that you produce by compiling C programs are

binary files. Sound and video files are also binary files.

Directory File

A directory contains no data, but keeps details of the files and subdirectories that it contains. A directory file contains one entry for every file and subdirectory that it houses. Each entry has two components namely, the filename and a unique identification number of the file or directory (called the *inode number).*

When you create or remove a file, the kernel automatically updates its corresponding directory by adding or removing the entry (filename and inode number) associated with the file.

Device File

All the operations on the devices are performed by reading or writing the file representing the device. It is advantageous to treat devices as files as some of the commands used to access an ordinary file can be used with device files as well.

Device filenames are found in a single directory structure, /dev. A device file is not really a stream of characters. It is the attributes of the file that entirely govern the operation of the device. The kernel identifies a device from its attributes and uses them to operate the device.

2. Filenames in UNIX

On a UNIX system, a filename can consist of up to 255 characters. Files may or may not have extensions and can consist of practically any ASCII character except the / and the Null character. You are permitted to use control characters or other nonprintable characters in a filename. However, you should avoid using these characters while naming a file. It is recommended that only the following characters be used in filenames:

Alphabets and numerals.

The period (.), hyphen (-) and underscore (_).

UNIX imposes no restrictions on the extension. In all cases, it is the application that imposes that restriction. Eg. A C Compiler expects C program filenames to end with .c, Oracle requires SQL scripts to have .sql extension.

A file can have as many dots embedded in its name. A filename can also begin with or end with a dot.

UNIX is case sensitive; cap01, Chap01 and CHAP01 are three different filenames that can coexist in the same directory.

3. Directories and Files

A file is a set of data that has a name. The information can be an ordinary text, a user-written computer program, results of a computation, a picture, and so on. The file name may consist of ordinary characters, digits and special tokens like the underscore, except the forward slash (/). It is permitted to use special tokens like the ampersand (&) or spaces in a filename.

Unix organizes files in a tree-like hierarchical structure, with the *root directory*, indicated by a forward slash (/), at the top of the tree. See the Figure below, in which part of the hierarchy of files and directories on the computer is shown.

4. Absolute and relative paths

A path, which is the way you need to follow in the tree structure to reach a given file, can be described as starting from the trunk of the tree (the / or root directory). In that case, the path starts with a slash and is called an absolute path, since there can be no mistake: only one file on the

system can comply.
Paths that don't start with a slash are always relative to the current directory. In relative paths we also use the . and .. indications for the current and the parent directory.

The HOME variable
When you log onto the system, UNIX automatically places you in a directory called the *home directory*. The shell variable HOME indicates the home directory of the user.

E.g.,

```
$ echo $HOME
  /home/kumar
```

What you see above is an absolute pathname, which is a sequence of directory names starting from root (/). The subsequent slashes are used to separate the directories.

5. pwd - print working directory

At any time you can determine where you are in the file system hierarchy with the ***pwd***, print working directory, command,

E.g.,:

```
$ pwd
  /home/frank/src
```

6. cd - change directory

You can change to a new directory with the ***cd***, change directory, command. ***cd*** will accept both absolute and relative path names.

Syntax

cd [directory]

Examples

cd changes to user's home directory
cd / changes directory to the system's root
cd .. goes up one directory level
cd ../.. goes up two directory levels
cd /full/path/name/from/root changes directory to absolute path named (note the leading slash)
cd path/from/current/location changes directory to path relative to current location (no leading slash)

7. mkdir - make a directory

You extend your home hierarchy by making sub-directories underneath it. This is done with the ***mkdir***, make directory, command. Again, you specify either the full or relative path of the directory.

Examples

mkdir patch Creates a directory *patch* under current directory
mkdir patch dbs doc Creates three directories under current directory
mkdir pis pis/progs pis/data Creates a directory tree with *pis* as a directory under the current directory and *progs* and *data* as subdirectories under *pis*

Note the order of specifying arguments in example 3. The parent directory should be specified first, followed by the subdirectories to be created under it.

The system may refuse to create a directory due to the following reasons:
1. The directory already exists.

2. There may be an ordinary file by the same name in the current directory.
3. The permissions set for the current directory don't permit the creation of files and directories by the user.

8. rmdir - remove directory

A directory needs to be empty before you can remove it. If it's not, you need to remove the files first. Also, you can't remove a directory if it is your present working directory; you must first change out of that directory. You cannot remove a subdirectory unless you are placed in a directory which is hierarchically *above* the one you have chosen to remove.

E.g.

rmdir patch Directory must be empty
rmdir pis pis/progs pis/data Shows error as *pis* is not empty. However rmdir silently deletes the lower level subdirectories *progs* and *data*.

9. The PATH environment variable

Environmental variables are used to provide information to the programs you use. We have already seen one such variable called HOME.
A command runs in UNIX by executing a disk file. When you specify a command like *date*, the system will locate the associated file from a list of directories specified in the PATH variable and then executes it. The PATH variable normally includes the current directory also.

Whenever you enter any UNIX command, you are actually specifying the name of an executable file located somewhere on the system. The system goes through the following steps in order to determine which program to execute:
1. Built in commands (such as cd and history) are executed within the shell.
2. If an absolute path name (such as /bin/ls) or a relative path name (such as ./myprog), the system executes the program from the specified directory.
3. Otherwise the PATH variable is used.

10. ls - list directory contents

The command to list your directories and files is ***ls***. With options it can provide information about the size, type of file, permissions, dates of file creation, change and access.

Syntax

ls [options] [argument]

Common Options

When no argument is used, the listing will be of the current directory. There are many very useful options for the ls command. A listing of many of them follows. When using the command, string the desired options together preceded by "-".

-a Lists all files, including those beginning with a dot (**.**).
-d Lists only names of directories, not the files in the directory
-F Indicates type of entry with a trailing symbol: executables with *, directories with / and symbolic links with @
-R Recursive list
-u Sorts filenames by last access time
-t Sorts filenames by last modification time
-i Displays inode number
-l Long listing: lists the mode, link information, owner, size, last modification (time). If the file is a symbolic link, an arrow (-->) precedes the pathname of the linked-to file.

The **mode field** is given by the **-l** option and consists of 10 characters. The first character is one of the following:

CHARACTER	IF ENTRY IS A
d	directory
-	plain file
b	block-type special file
c	character-type special file
l	symbolic link
s	socket

The next 9 characters are in 3 sets of 3 characters each. They indicate the **file access permissions**: the first 3 characters refer to the permissions for the **user**, the next three for the users in the Unix **group** assigned to the file, and the last 3 to the permissions for **other** users on the system.

Designations are as follows:

r read permission
w write permission
x execute permission
- no permission

Examples

1. To list the files in a directory:

```
$ ls
```

2. To list all files in a directory, including the hidden (dot) files:

```
$ ls -a
```

3. To get a long listing:

```
$ ls -al
total 24
drwxr-sr-x 5 workshop acs 512 Jun 7 11:12 .
drwxr-xr-x 6 root sys 512 May 29 09:59 ..
-rwxr-xr-x 1 workshop acs 532 May 20 15:31 .cshrc
-rw------- 1 workshop acs 525 May 20 21:29 .emacs
-rw------- 1 workshop acs 622 May 24 12:13 .history
-rwxr-xr-x 1 workshop acs 238 May 14 09:44 .login
-rw-r--r-- 1 workshop acs 273 May 22 23:53 .plan
-rwxr-xr-x 1 workshop acs 413 May 14 09:36 .profile
-rw------- 1 workshop acs 49 May 20 20:23 .rhosts
drwx------ 3 workshop acs 512 May 24 11:18 demofiles
drwx------ 2 workshop acs 512 May 21 10:48 frank
drwx------ 3 workshop acs 512 May 24 10:59 linda
```

11. The UNIX File System

The root directory has many subdirectories. The following table describes some of the subdirectories contained under root.

Directory	Content
/bin	Common programs, shared by the system, the system administrator and the users.
/dev	Contains references to all the CPU peripheral hardware, which are represented as files with special properties.
/etc	Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
/home	Home directories of the common users.
/lib	Library files, includes files for all kinds of programs needed by the system and the users.
/sbin	Programs for use by the system and the system administrator.

/tmp	Temporary space for use by the system, cleaned upon reboot, so don't use this for saving any work!
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, or to keep an image of a CD before burning it.

Conclusion

In this chapter we looked at the UNIX file system and different types of files UNIX understands. We also discussed different commands that are specific to directory files viz., pwd, mkdir, cd, rmdir and ls. These commands have no relevance to ordinary or device files. We also saw filenaming conventions in UNIX. Difference between the absolute and relative pathnames was highlighted next. Finally we described some of the important subdirectories contained under root (/).