BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA – BANGALORE – 64

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

18CS56-UNIX PROGRAMMING

5TH SEMESTER

LECTURE NOTES

MODULE-1

PREPARED BY
PROF.S.MAHALAKSHMI

UNIX PROGRAMMING (Effective from the academic year 2018 -2019) SEMESTER – V			
Course Code	**18CS56**	**CIE Marks**	40
Number of Contact Hours/Week	3:0:0	**SEE Marks**	60
Total Number of Contact Hours	40	**Exam Hours**	03
CREDITS – 3			

Course Learning Objectives: This course (18CS56) will enable students to

- Interpret the features of UNIX and basic commands.
- Demonstrate different UNIX files and permissions
- Implement shell programs.
- Explain UNIX process, IPC and signals.

Module 1	**Contact Hours**
Introduction: Unix Components/Architecture. Features of Unix. The UNIX Environment and UNIX Structure, Posix and Single Unix specification. General features of Unix commands/ command structure. Command arguments and options. Basic Unix commands such as echo, printf, ls, who, date,passwd, cal, Combining commands. Meaning of Internal and external commands. The type command: knowing the type of a command and locating it. The root login. Becoming the super user: su command. **Unix files:** Naming files. Basic file types/categories. Organization of files. Hidden files. Standard directories. Parent child relationship. The home directory and the HOME variable. Reaching required files- the PATH variable, manipulating the PATH, Relative and absolute pathnames. Directory commands – pwd, cd, mkdir, rmdir commands. The dot (.) and double dots (..) notations to represent present and parent directories and their usage in relative path names. File related commands – cat, mv, rm, cp, wc and od commands. **RBT: L1, L2**	08
Module 2	
File attributes and permissions: The ls command with options. Changing file permissions: the relative and absolute permissions changing methods. Recursively changing file permissions. Directory permissions. **The shells interpretive cycle:** Wild cards. Removing the special meanings of wild cards. Three standard files and redirection. **Connecting commands:** Pipe. Basic and Extended regular expressions. The grep, egrep. Typical examples involving different regular expressions. **Shell programming:** Ordinary and environment variables. The .profile. Read and readonly commands. Command line arguments. exit and exit status of a command. Logical operators for conditional execution. The test command and its shortcut. The if, while, for and case control statements. The set and shift commands and handling positional parameters. The here (<<) document and trap command. Simple shell program examples. **RBT: L1, L2**	08
Module 3	
UNIX File APIs: General File APIs, File and Record Locking, Directory File APIs, Device File APIs, FIFO File APIs, Symbolic Link File APIs. **UNIX Processes and Process Control:** **The Environment of a UNIX Process:** Introduction, main function, Process Termination, Command-Line Arguments, Environment List, Memory Layout of a C Program, Shared Libraries, Memory Allocation, Environment Variables, setjmp and longjmp Functions,	08

getrlimit, setrlimit Functions, UNIX Kernel Support for Processes. **Process Control: Introduction, Process Identifiers, fork, vfork, exit, wait, waitpid, wait3, wait4 Functions, Race Conditions, exec Functions** **RBT: L1, L2, L3**	
Module 4	
Changing User IDs and Group IDs, Interpreter Files, system Function, Process Accounting, User Identification, Process Times, I/O Redirection. **Overview of IPC Methods, Pipes, popen, pclose Functions, Coprocesses, FIFOs, System V IPC, Message Queues, Semaphores.** **Shared Memory, Client-Server Properties, Stream Pipes, Passing File Descriptors, An Open Server-Version 1, Client-Server Connection Functions.** **RBT: L1, L2, L3**	08
Module 5	
Signals and Daemon Processes: Signals: The UNIX Kernel Support for Signals, signal, Signal Mask, sigaction, The SIGCHLD Signal and the waitpid Function, The sigsetjmp and siglongjmp Functions, Kill, Alarm, Interval Timers, POSIX.lb Timers. Daemon Processes: Introduction, Daemon Characteristics, Coding Rules, Error Logging, Client-Server Model. **RBT: L1, L2, L3**	08

Course Outcomes: The student will be able to :

- Explain Unix Architecture, File system and use of Basic Commands
- Illustrate Shell Programming and to write Shell Scripts
- Categorize, compare and make use of Unix System Calls
- Build an application/service over a Unix system.

Question Paper Pattern:

- The question paper will have ten questions.
- Each full Question consisting of 20 marks
- There will be 2 full questions (with a maximum of four sub questions) from each module.
- Each full question will have sub questions covering all the topics under a module.
- The students will have to answer 5 full questions, selecting one full question from each module.

Textbooks:

1. Sumitabha Das., Unix Concepts and Applications., 4th Edition., Tata McGraw Hill (Chapter 1,2 ,3,4,5,6,8,13,14)
2. W. Richard Stevens: Advanced Programming in the UNIX Environment, 2nd Edition, Pearson Education, 2005 (Chapter 3,7,8,10,13,15)
3. Unix System Programming Using C++ - Terrence Chan, PHI, 1999. (Chapter 7,8,9,10)

Reference Books:

1. M.G. Venkatesh Murthy: UNIX & Shell Programming**,** Pearson Education.
2. Richard Blum , Christine Bresnahan : Linux Command Line and Shell Scripting Bible, 2ndEdition, Wiley,2014.

Faculty can utilize open source tools to make teaching and learning more interactive.

MODULE 1: UNIX ARCHITECTURE, BASIC COMMANDS, ADMINISTRATION, UNIX FILES AND ITS RELATED COMMANDS

1.1 Introduction
1.2 Unix Architecture
1.3 Features of Unix
1.4 POSIX and Single Unix Specification
1.5 Command Structure
    1.5.1 Options
    1.5.2 Filename Arguments
    1.5.3 Exceptions
1.6 Understanding of Some Basic Commands
    1.6.1 echo
    1.6.2 printf
    1.6.3 ls
    1.6.4 who
    1.6.5 date
    1.6.6 passwd
    1.6.7 cal
1.7 Flexibility of Command Usage
1.8 Internal and External Commands
1.9 type
1.10 Types of Accounts
    1.10.1 The root Login
    1.10.2 su: Becoming Super user
1.11 UNIX Files
1.12 Naming Files
1.13 Basic File-Types
    1.13.1 Ordinary (Regular) File
    1.13.2 Directory File
    1.13.3 Device File
1.14 Standard Directories
1.15 Parent Child Relationship
1.16 HOME Variable
1.17 PATH Variable
1.18 Relative and Absolute Pathname
    1.18.1 Absolute Pathname
    1.18.2 Relative Pathname
1.19 Directory Commands
    1.19.1 pwd
    1.19.2 cd
    1.19.3 mkdir
    1.19.4 rmdir
1.20 File Related Commands
    1.20.1 cat
    1.20.2 wc
    1.20.3 cp
    1.20.4 mv
    1.20.5 rm
    1.20.6 od

1.1 Introduction

- UNIX is an operating system.
- An operating system is a set of programs that act as a link between the computer and the user.
- The operating system (OS) manages the resources of a computer.
- Examples of computer resources are: CPU, RAM, disk memory, printers, displays, keyboard, etc.

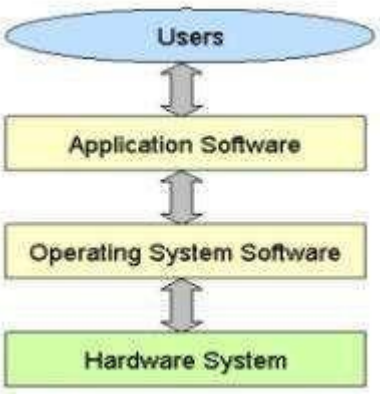

Figure 1.1 : Operasting System

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

1.2 Unix Architecture (Components)

- The UNIX operating system (OS) consists of a kernel layer, a shell layer , an application layer & files.

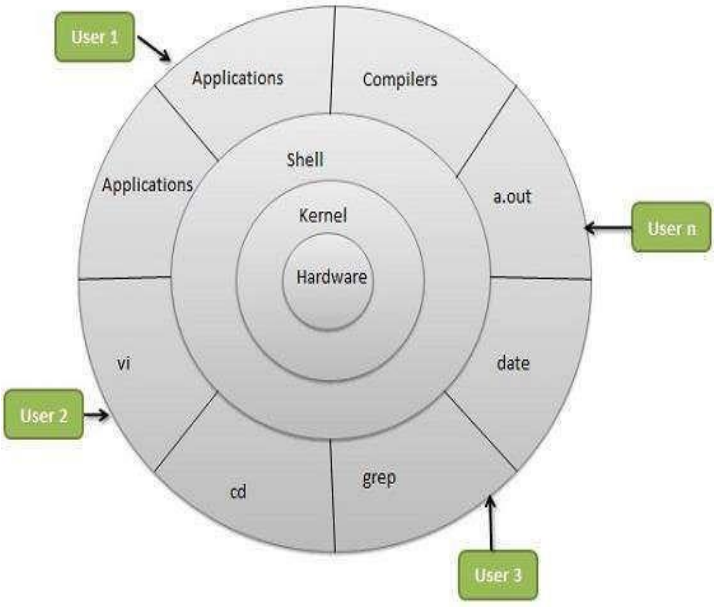

Figure 1.2 : Unix Architecture

Kernel

- The kernel is the heart of the operating system.
- It interacts with the machine's hardware.
- It is a collection of routines written in C.
- It is loaded into memory when the system is booted.

Main responsibilities:

1) Memory management
2) Process management (using commands: kill, ps, nohup)
3) File management (using commands: rm, cat, ls, rmdir, mkdir)

- To access the hardware, user programs use the services of the kernel via system calls.

System Call

- A system call is a request for the operating system to do something on behalf of the user's program.
- The system calls are functions used in the kernel itself.
- To the programmer, the system call appears as a normal C function call.
- UNIX system calls are used to manage the file system and control processes.
- Example: read(), open(), close(), fork(), exec(), exit()

- There can be only one kernel running on the system.

Shell

- The shell interacts with the user.
- The shell is a command line interpreter (CLI).
- Main responsibilities:
  1) interprets the commands the user types in and
  2) dispatches the command to the kernel for execution
- There can be several shells in action, one for each user who's logged in.
- There are multiple shells that are used by the UNIX OS.
- For example: Bourne shell (sh), the C shell (csh), the Korn shell (ksh) and Bourne Again shell (bash).

Application

- This layer includes the commands, word processors, graphic programs and database management programs.

File

- A file is an array of bytes that stores information.
- 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.

1.3 Features of Unix

1) Multiuser

- UNIX is a multiprogramming system.
- Multiple users can access the system by connecting to points known as terminals.
- Several users can run multiple programs simultaneously on one system.

2) Multitasking

- UNIX is a multitasking system.
- A single user can also run multiple tasks concurrently.
- For example:

  At the same time, a user can
  → edit a file
  → print another file one on the printer
  → send email to a friend and
  → browse www

- The kernel is designed to handle a user's multiple needs.
- In a multitasking environment, a user sees one job running in the foreground; the rest run in the background.
- User can switch jobs between background and foreground, suspend, or even terminate them.

3) Pattern Matching
- UNIX has very sophisticated pattern matching features.
- Regular Expressions are a feature of UNIX.
- Regular Expressions describe a pattern to match, a sequence of characters, not words, within a line of text.

4) Portable
- UNIX can be installed on many hardware platforms.
- Unix operating system is written in C language, hence it is more portable than other operating systems.

5) UNIX Toolkit
- UNIX offers facility to add and remove many applications as and when required.
- Tools include
  - → general purpose tools
  - → text manipulation tools
  - → compilers/interpreters
  - → networked applications and
  - → system administration tools

6) Programming Facility
- The UNIX shell is also a programming language; it was designed for programmer, not for end user.
- It has all the necessary ingredients, like control structures, loops and variables, that establish powerful programming language.
- This features are used to design shell scripts – programs that can also invoke UNIX commands.
- Many of the system's functions can be controlled and automated by using these shell scripts.

7) Documentation
- The principal on-line help facility available is the man command, which remains the most important references for commands and their configuration files.
- Apart from the man documentation, there's a vast ocean of UNIX resources available on the Internet.

1.4 POSIX and Single Unix Specification
- The Portable Operating System Interface (POSIX) is a family of standards specified by IEEE for maintaining compatibility between operating systems.
- Two of the most important standards from POSIX are:
  1) POSIX.1 – Specifies the C application program interface – the system calls (Kernel)
  2) POSIX.2 – Deals with the Shell and utilities
- In 2001, a joint initiative of X/Open and IEEE resulted in the unification of two standards.
- This is the Single UNIX Specification, Version 3 (SUSV3).
- The "Write once, adopt everywhere" approach to this development means that once software has been developed on any POSIX machine it can be easily ported to another POSIX compliant machine with minimum or no modification.

1.5 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.

1.5.1 Options

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

  $ ls –l // -l option list all the attributes of the file note
- 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 combined with only one – sign.

  thus $ ls –l –a –t is same as $ 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 -.

1.5.2 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:

```
ls -lat chap01 chap02 chap03 # Multiple filenames as arguments
cp file1 file2 file3 dest_dir
rm file1 file2 file3
```

- The command with its arguments and options is known as the command line.
- This line can be considered complete only after the user has hit [Enter].
- The complete line is then fed to the shell as its input for interpretation and execution.

1.5.3 Exceptions

- There are some commands that don't accept any arguments.
- There are also some commands that may or may not be specified with arguments.
- For Example:

  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).
- There are some commands compulsorily take options (cut).
- There are some commands which can take an expression as an argument, or a set of instructions as argument. Ex: grep, sed

  Thu Jun 25 08:30:19 MST 2017

1.6 Understanding of Some Basic Commands

1) echo

- This command can be used to display a message on the terminal.
- Example:

  $ echo "Welcome to UNIX \n"

  Welcome to UNIX
- This command can be used with following escape characters:

"\a"	Audible Alert (Bell)
"\b“	Back Space
"\f“	Form Feed
"\n“	New Line
"\r“	Carriage Return
"\t“	Horizontal Tab
"\v“	Vertical Tab
"\\“	Backslash

2) printf

- This command can be used to display a message on the terminal.
- This command can be used with following escape characters:

"\a"	Audible Alert (Bell)
"\b“	Back Space
"\f“	Form Feed
"\n“	New Line
"\r“	Carriage Return
"\t“	Horizontal Tab
"\v“	Vertical Tab
"\\“	Backslash

- Example:

```
$ printf "Welcome to UNIX \n"
Welcome to UNIX
```

- Similar to C language, this command can be used with following format specifiers:

%d	Decimal integer
%f	Floating point number
%s	String
%o	Octal integer (base 8)
%x	Hexadecimal integer (base 16)

- Syntax:

```
printf("format-string", variable-list);
        where format-string contains one or more format-specifiers
              variable-list contains names of variables
```

- Example:

```
$ printf "My current shell is %s\n" $SHELL
My current shell is /bin/ksh
```

3) ls

- ls command can be used to obtain a list of all filenames in the current directory.
- -l option can be used to obtain a detailed list of attributes of all files in the current directory.
- Example:

$ ls -l						
Type & Perm	Link	Owner	Group	Size	Date & Time	File Name
-rwxr-xr--	1	kumar	metal	195	may 10 13:45	chap01
drwxr-xr-x	2	kumar	metal	512	may 09 12:55	helpdir

4) who

- This command can be used to display the details of all the users logged-in into the unix system at the same time.
- Example:

$ who		
kumar	tty0	Oct 8 14:10
rama	tty2	Oct 4 09:08

- –H option can be used to display the header information.
- Example:

$ who			
NAME	LINE	TIME	COMMENT
kumar	tty0	Oct 8 14:10	
rama	tty2	Oct 4 09:08	

- -u option can be used to display detailed information of users.

• Example:

$ who -Hu					
NAME	LINE	TIME	IDLE	PID	COMMENT
kumar	tty0	Oct 8 14:10	00:18	185	
rama	tty2	Oct 4 09:08	00:23	123	

5) date

• This command can be used to display the current date and time.
• Example:

```
$ date
Mon Sep 4 16:40:02 IST 2017
```

• This command can also be used with suitable format specifiers as arguments.
• Following are some format specifiers:

d – day of month (1 - 31) m - Month (01-12) y – last two digits of the year.
H– hour (00-24) M – minute (00-59) S – second (00-59)
D – date in the format mm/dd/yy
T – time in the format hh:mm:ss

• Syntax:

$ date +%format_specifier

• Example:

```
$ date +"%d-%m-%y"
04-09-17
$ date +%m                          // displays month number
09
```

6) passwd

• This command can be used to change user password.
• All Unix systems require passwords to help ensure that your files and data remain secure from hackers.
• Following are the steps to change your password –

1) To start, type password at the command prompt as shown below.
2) Enter your old password, the one you're currently using.
3) Type in your new password.
4) You must verify the password by typing it again.

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

7) cal

• This command can be used to display the calendar of the current month.
• Syntax:

cal [[month] year]

• This command can also be used to display the calendar of any specific month or a complete year.

- Example:

```
$ cal
September 2017
```

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					

- You can't hold the calendar of a year in a single screen page; it scrolls off rapidly to end of the year.
- To pause at each screen page, a pipe "|" can be connected to more pager:
- Example:

```
$cal 2017 | more
```

1.7 Flexibility of Command Usage

- UNIX provides flexibility in using the commands.
- A command can often be entered in more than one way.
- Shell allow the following type of command usage:
  1) Combining Commands.
  2) A command line can overflow or Be split into multiple lines.
  3) Entering a command before previous command has finished.

1) Combining Commands

- UNIX allows you to specify more than one command in the single command line.
- Example:

```
$ wc sample.txt ; ls –l sample.txt          //Two commands separated by ; (semicolon).
2 3 16 sample.txt
-rw-rw-r-- 1 kumar group 16 Jan 30 09:35 sample.txt
$ ls | wc                  //Two commands combined here using filter
```

- 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.

2) A Command line can be split into multiple lines

- UNIX terminal width is restricted to maximum 80 characters.
- Shell allows command line to overflow or be split into multiple lines.
- Example:

```
$ echo "This is                  // $ first prompt
> a three-line                  // > is a secondary prompt
> text message"                 // Command line ends here
This is a three-line text message
```

3) 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 can be entered at the keyboard without waiting for prompt.
- The input remains stored in a buffer maintained by kernel for all keyboard input.
- The next command is passed on to the shell for interpretation after the previous program has completed.

1.8 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 the command (file) has an independence existence in the /bin directory, it is called external command.
• Examples:

```
$ type echo                    // echo is an internal command
echo is shell built-in
$ type ls                      // ls is an external command
ls is /bin/ls
```

• If the command exists both as an internal and external one, shell execute internal command only.
• Internal commands will have top priority compare to external command of same name.

1.9 Type

• The UNIX is command based system i.e.,- things happens because the user enters commands in.
• Usually, UNIX commands are less than 5 characters long.
• All UNIX commands are single words like – cat, ls, pwd, date, mkdir, rmdir, cd, grep etc.
• The command names are all in lowercase.
• Example:

```
$ LS bash:
LS: command not found
```

• All UNIX commands are files containing programs, mainly written in C.
• All UNIX commands(files) are stored in directories(folders).
• If you want to know the location of executable program (or command), use type command.
• Example:

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

• When you execute ls command, the shell locates this file in the /bin directory and makes arrangements to execute it.

1.10 Types of Accounts

There are 2 types of accounts on a Unix system:

1) Root Account

• The system administrator is known as superuser or root user.
• The superuser has complete control of the system.

2) User Accounts

• User accounts provide interactive access to the system for users and groups of users.
• General users are typically assigned to these accounts and usually have limited access to critical system files and directories.

1) The root Login

• The system administrator is known as superuser or root user.
• The superuser has complete control of the system.
• The superuser can run any commands without any restriction.
• The job of superuser includes:
→ maintaining user accounts
→ providing security and
→ managing disk space
→ performing backups
• The root account doesn't need to be separately created but comes with every system.
• The root account's password is generally set at the time of installation of the system and has to be used on logging in:

```
Login: root
Password: ********* [Enter]
# -
```

- The command prompt of root is hash (#).
- Once you login, you are placed in root's home directory "/".
- /sbin and /usr/sbin contains administrative commands of the system.

2) su: Becoming Super User

- Any user can acquire superuser status with the su command if they know the root password.
- For example, the user "kumar" becomes a superuser in this way:

```
$ su
Password: *****
# pwd
/home/kumar
```

- Though the current directory doesn't change, the # prompt indicates that kumar now has powers of a superuser.
- To be in root's home directory on superuser login, use

```
su –l
```

Creating a User's Environment

- User's often rush to the administrator with the complaint that a program has stopped running.
- The administrator first tries running it in a simulated environment.
- su command with a – (hyphen) can be used to recreate the user's environment without the login-password.

```
su –kumar
```

- This sequence executes kumar's .profile and temporarily creates kumar's environment.
- su runs in a separate sub-shell, so this mode is terminated by hitting [ctrl-d] or using exit.

MODULE 1: UNIX FILE SYSTEM

1.11 UNIX Files

- A file is the container for storing information.
- The file doesn't store 1) file-size and 2) file-name.
- Some attributes of file are file-type, permissions, links, owner, group-owner etc.
- These file attributes are stored in inode table which is accessible only to kernel.
- 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 etc are considered to be files.
- Generally, all devices including I/O devices are also considered to be files.

1.12 Naming Files

- A filename can consist up to 255 characters.
- Files may or may not have extensions.
- Files consist of any ASCII character expect the "/" and NULL character.
- Following is recommended for filenames:
  - → Alphabetic characters and numerals
  - → period(.), hyphen(-) and underscore(_)
- However, users are permitted to use control characters or other unprintable characters in a filename.
  Example: .last_time list. @ # $ % * abcd a.b.c.d.e
- UNIX is case sensitive; chap01, Chap01 and CHAP01 are three different filenames.

1.13 Basic File Types

- Three categories of files are:
  - 1.13.1 Ordinary file (or regular file) – It contains only data as a stream of characters.
  - 1.13.2 Directory file – it contains files and other sub-directories.
  - 1.13.3 Device file – all devices and peripherals are represented by files.

1.13.1 Ordinary (Regular) File

- An ordinary file is a file on the system that contains data, text, or program instructions.
- An ordinary file can be either a text file or a binary file.

1) A text file contains only printable characters and you can view and edit them.
   - Examples: All C and Java program sources, shell scripts are text files.
   - Every line of a text file is terminated with the newline character.

2) A binary file contains both printable and non printable characters that cover the entire ASCII range.
   - Examples: Most Unix commands, executable files, pictures, sound and video files are binary files.

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.

1.13.2 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,
  1) filename and
  2) 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.
- Unix directories are equivalent to windows folders.

1.13.3 Device File

- All the operations on the devices are performed by reading or writing the file representing the device.
- Advantage of device file is that some of the commands used to access an ordinary file also work with device file.
- Device filenames are generally 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 de ice from its attributes and uses them to operate the device.

1.14 Parent Child Relationship

- 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.

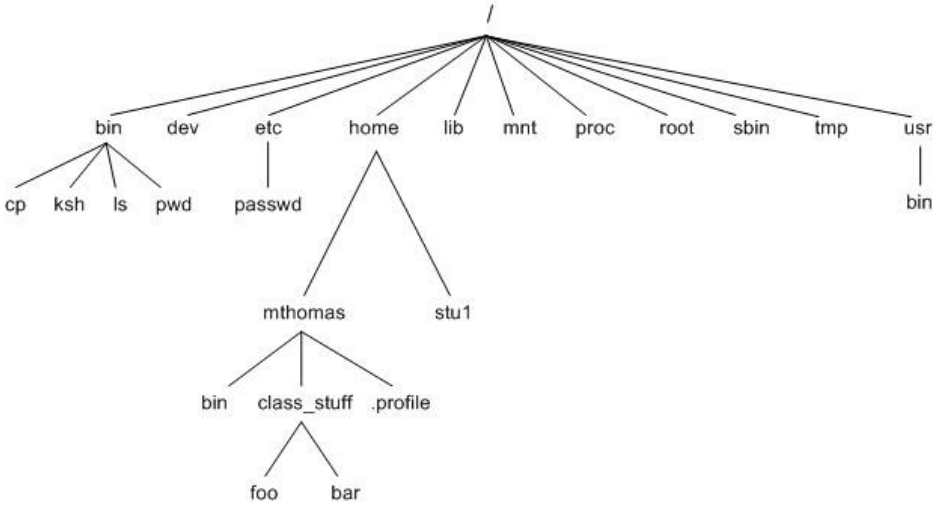

Figure 2.4: Parent Child Relationship

- The feature of UNIX file system is that there is a top, which serves as the reference point for all files.
- This top is called root (represented by a "/").
- The root is actually a directory.
- The root directory (/) has a number of subdirectories under it.
- The subdirectories in turn have more subdirectories and other files under them.
- Every file apart from root, must have a parent, and it should be possible to trace the ultimate parentage of a file to root.
- In parent-child relationship, the parent is always a directory.

1.15 Standard Directories (UNIX Filesystem)

- Following are the directories that exist on the major versions of Unix

1) / (root directory)
- This contains only the directories needed at the top level of the file structure.

2) /bin
- This contains common programs, shared by the system, the system administrator and the users.

3) /dev
- This contains references to all the CPU peripheral hardware, which are represented as files with special properties. These are device drivers.

4) /etc
- This contains system configuration files.

5) /home
- This contains home directories of the common users.

6) /lib
- This contains library files, including files for all kinds of programs needed by system and users.

7) /sbin
- This contains programs for use by the system and the system administrator.

8) /tmp
- This contains temporary space for use by the system. This space is cleaned upon reboot. So, don't use this for saving any work.

9) /usr
- This contains programs, libraries, documentation etc. for all user-related programs.

10) /var
- This contains storage for all variable files and temporary files created by users, such as
  - → log files
  - → mail queue or
  - → print spooler area

1.16 HOME Variable

- When a user logs into the system, the user will be placed in a directory called home directory.
- Home directory is created by the system when the user account is created.
- The shell-variable HOME indicates the home directory of the current user.
- This variable is set for a user by the system admin in /etc/passwd.
- Example:

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

- Here, absolute pathname is shown.
- Absolute pathname is a sequence of directory names starting from root (/).
- The subsequent slashes are used to separate the directories.

1.17 PATH Variable

- This variable specifies the locations in which the shell should look for commands.
- Example:

```
$ echo $PATH
/bin: /usr/bin:
```

- 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 also includes the current directory.
- 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:

1.17.1 Built in commands (such as cd and history) are executed within the shell.

1.17.2 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.

1.17.3 Otherwise, the PATH variable is used.

1.18 Relative and Absolute Pathname

- A pathname is a text string made up of one or more names separated by a "/".
- A pathname specifies how to traverse (navigate) the hierarchical directory names in the file system to reach some destination object.

1.18.1 Absolute Pathname

- An absolute pathname begins with a slash (/).
- The Absolute path defines the location of a Directory or a file from the root file system (/).
- The absolute path contains the full path to the directory or file.

1.18.2 Relative Pathname

- The relative pathname do not begin with "/".
- It specify the location relative to your current working directory.
  1) (a single dot) - This represents the current directory.
  2) .. (two dots) - This represents the parent directory.
- Example:
- 'date' command can executed in two ways as follows:

Using Absolute Pathname	**Using Relative Pathname**
$ /bin/date Thu Sep 7 10:20:29 IST 2017	$ date Thu Sep 7 10:20:29 IST 2017

- Example:
- P1.java can be copied to kumar under home directory in two ways as follows:

Using Absolute Pathname	**Using Relative Pathname**

$ pwd
/home/kumar
$ cp /home/sharma/P1.java /home/kumar
	$ pwd
	/home/kumar
	$ cp /home/sharma/progs .

- Example:
- cd & mkdir can be used in two ways as follows:

Using Absolute Pathname	Using Relative Pathname

$ pwd
/home/kumar
$ mkdir /home/kumar/progs
$ cd /home/kumar/progs
$ pwd
/home/kumar/progs
$ cd /home/kumar
$ pwd
/home/kumar
	$ pwd
	/home/kumar
	$ mkdir progs
	$ cd progs
	$ pwd
	/home/kumar/progs
	$ cd ..
	$ pwd
	/home/kumar

1.19 Directory Commands

1.19.1 pwd (print working directory)

- This command can be used to display the current working directory.
- Example:

```
$ pwd
/home/kumar
```

- Like HOME, it displays the absolute path.

1.19.2 cd

- This command can be used to change the current working directory.
- Syntax

```
cd PATHNAME
```

Case 1:

- This command can be used with the argument.
- This command can work with both absolute and relative path names.
- Example:

Using relative pathname	Using absolute pathname

$ pwd
/home/kumar
$ mkdir progs
$ cd progs
$ pwd
/home/kumar/progs
	$ pwd
	/home/kumar
	$ mkdir /home/kumar/progs
	$ cd /home/kumar/progs
	$ pwd
	/home/kumar/progs

Case 2:

- This command can also be used without the argument.
- When used without argument, this command changes the working directory to home directory.
- Example:

```
$ pwd
/home/kumar/progs
$ cd
$ pwd
/home/kumar
```

Case 3:

- This command can also be used with short hand notations.
- Example:

```
$ cd /          // changes the working directory to root directory (/)
$ cd ..         // changes the working directory to the one level up parent directory (..)
$ cd ../..      // changes the working directory to the two level up parent directory (../..)
```

1.19.3 mkdir

- This command can be used to create a new directory.
- Syntax:
  mkdir DIRECTORY_NAME
- This command can work with both absolute and relative path names.
- Example:

Using relative pathname	Using absolute pathname

$ pwd
/home/kumar
$ mkdir progs
$ cd progs
$ pwd
/home/kumar/progs
	$ pwd
	/home/kumar
	$ mkdir /home/kumar/progs
	$ cd /home/kumar/progs
	$ pwd
	/home/kumar/progs

Case 2:

- This command can also accept more than one directory name as arguments.
- Example:

```
$mkdir usp ade dms                  // creates 3 directories under current directory
$mkdir sem3 sem3/usp sem3/ade       // creates 3 subdirectories – sem3, usp, ade
```

- The order of specifying arguments is important. You cannot create subdirectories before creation of parent directory.
- System refuses to create a directory due to following reasons:
  1) User doesn't have permission to create directory. (i.e. directory write protected).
  2) The directory already exists.
  3) There may be ordinary file by that name in the current directory.

1.19.4 rmdir

- This command can be used to delete a directory.
- Syntax:
  rmdir DIRECTORY_NAME

Case 1:

- Example:

```
$ rmdir usp                         // delete usp directory under current directory
```

Case 2:

- This command can also accept more than one directory name as arguments.
- Example:

```
$ mkdir sem3 sem3/usp sem3/ade      // creates 3 subdirectories – sem3, usp, ade
$ rmdir sem3/ade sem3/usp sem3      // deletes 3 subdirectories – sem3, usp, ade
```

- Here, rmdir expects the arguments to be reverse of mkdir's arguments.
- The order of specifying arguments is important. You cannot delete parent directories before deletion of subdirectories.
- System refuses to delete a directory due to following reasons:

  1) User doesn't have permission to delete directory. (i.e. write protected directory).
  2) The directory doesn't exist in system.
  3) The directory is your present working directory.

1.20 File Related Commands

1.20.1 cat

- This command can be used to display the content of a file on the terminal.
- Syntax:

  cat FILENAME

Case 1:

- Example:

```
$ cat P1.c
WELCOME TO UNIX          // contents of P1.c
$ cat P2.c
WELCOME TO PERL          // contents of P2.c
```

Case 2:

- This command can also accept more than one filename as arguments.
- Example:

```
$ cat P1.c P2.c
WELCOME TO UNIX          // contents of P1.c
WELCOME TO PERL          // contents of P2.c
```

- Here, the content of the second file is shown immediately after the first file.
- So, this command concatenates two files- hence its name (cat).

Case 3:

- This command can also be used to create a new file.
- Syntax:

  cat > FILENAME

- Example:

```
$ cat > P3.c
WELCOME TO SHELL                 // contents of P3.c
[ctrl-d]                         // Terminates P3.c
$ cat P3.c
WELCOME TO SHELL                 // contents of P3.c
```

cat Options

1) Displaying Non-printing Characters (-v)

- By default, without any option, this command displays only printing ASCII characters of the file.
- -v option can be used to display even non-printing ASCII characters of the file.

2) Numbering Lines (-n)

- -n option can be used to number the lines of the file.
- This option helps the programmer in debugging programs.

1.20.2 wc

- This command can be used to get a count of the total number of lines, words, and characters contained in a file.
- Syntax:

  wc FILENAME

Case 1:

- Example:

```
$ cat P1.c
WELCOME              // contents of P1.c
TO
UNIX
$ wc P1.c
LINE   WORD  CHARATCTER  FILENAME
3      3     15          P1.c
```

- The header includes the following attributes:

1. **LINE**
   - This represents the total number of lines in the file.
2. **WORD**
   - This represents the total number of words in the file (excluding space, tab and newline).
3. **CHARATCTER**
   - This represents the total number of characters in the file (including space, tab and newline).
4. **FILENAME**
   - This represents the name of the file.

Case 2:

- This command can also accept more than one filename as arguments.
- Example:

```
$ wc  P1.c P2.c
3       3       15              P1.c
3       3       15              P2.c
```

wc Options

–l option can be used to count only number of lines
–w option can be used to count only number of words
–c option can be used to count only number of characters

- Example:

```
$ wc -l P1.c
3       P1.c
$ wc -c P1.c
15  P1.c
```

1.20.3 cp

- This command is used to copy a file(s) from one location to another location.
- It creates an exact image of the file on the disk with a different name.
- Syntax:

  cp SOURCE_FILE DESTINATION_FILE

Case 1:

- This command can be used to copy a file within current working directory.
- Example:

  $ cp FILE1 FILE2 // copies contents of FILE1 to FILE2 in current working directory

- Here,
  1. If the destination file doesn't exist, it will first be created before copying takes place.
  2. If the destination file exists, it will be overwritten without any warning from the system.

Case 2:

- This command can also be used to copy a file to the another directory.
- Example:

  $ cp FILE1 part2/FILE2

- Here, this copies the file FILE1 from your current working directory to the file FILE2 in the subdirectory "part2".

Case 3:

- This command can also be used with .(dot) to signify the current directory as the destination.
- Example:

  $ cp part2/FILE2 . same as $ cp part2/FILE2 FILE2

- Here, this copies the file FILE2 in the subdirectory "part2" to your current working directory.

Case 4:

- This command can also accept more than 2 filenames as arguments.
- In this case, the last filename must be a directory.
- Example:

```
$ cp FILE1 FILE2 FILE3 module       // copies 3 files to "module" directory
```

- Here, "module" directory should already exist because cp cannot create a directory.

Case 5:

- This command can also be used with metacharcters (* or ?).
- Example:

```
cp *.pdf DIR1          //copies all the files with extensions .pdf to directory DIR1
```

cp Options

1) Interactive Copying (-i)

- –i option can be used to warn the user before overwriting the destination file.
- Example:

```
$ cp –i FILE1 FILE2                    // copies contents of FILE1 to FILE2 if "Y" is entered
$ cp: overwrite FILE2 (yes/no)? Y
```

2) Recursive Copying (-R)

- –R option can be used to recursively copy an entire directory structure from one location to another location.
- Entire directory including all files in its subdirectories will be copied.
- Example:

```
cp –R DIR1 DIR2
```

- If directory DIR2 doesn't exist, cp creates it along with the associated subdirectories.

1.20.4 mv

- This command renames or moves files.

Case 1:

- This command can be used to rename a file in the current directory.
- Syntax:

```
mv OLDFILENAME NEWFILENAME
```

- It doesn't create a copy of the file; it merely renames it.
- No additional space is consumed on disk during renaming.
- Example:

```
$ mv OLDFILE NEWFILE                // renames the OLDFILE by NEWFILE
```

Case 2:

- This command can also be used to move a group of files to a directory.
- In this case, the last filename must be a directory.
- Example:

```
$ mv FILE1 FILE2 FILE3 module      // moves 3 files to "module" directory
```

1.20.5 rm

- This command can be used to delete a file.
- Syntax:

```
rm FILENAME
```

- Example:

```
$ rm FILE1                    // deletes FILE1
$ rm FILE1 FILE2 FILE3        // deletes three files
$ rm *.pdf                    // deletes all files with extensions .pdf in the currant directory
```

rm Options

1) Interactive Deletion (-i)

- –i option can be used to warn the user before deleting the file.
- Example:

```
$ rm –i FILE1 FILE2                          // delete FILE1 & FILE2 if "Y" is entered
rm: remove FILE1 (yes/no)? ? Y
rm: remove FILE2 (yes/no)? ? Y
```

2) Recursive Deletion (-r or -R)

- –R option can be used to recursively delete an entire directory structure.
- Entire directory including all files in its subdirectories will be deleted.
- Example:

```
$ rm –r DIR1          // delete DIR1 and all its subdirectories & files
$ rm -r *             // deletes all files in the current directory and all its subdirectories.
```

3) Forceful Deletion (-f)

- By default, this command cannot delete a file which is write-protected.
- –f option can be used to delete even the write-protected file.
- Example:

```
rm -rf *              // deletes all files in the current directory and all its subdirectories.
```

1.20.6 Od

- This command can be used to display the content of executable file in a ASCII octal form.
- Example:

```
$ cat P1.obj
abcd efgh                    // content of file P1.obj
abcd efgh
```

od Options

1) byte (-b)

- -b option can be used to display octal value of each printable character.
- Each line displays 16 bytes of data in octal, preceded by the offset of the first byte in the line.
- Example:

```
$ od –b file
   offset   <-----------------  16 bytes of data in octal  ------------------------>
0000000 141 142 143 144 040 145 146 147 148 040 040 040 040 040 040 012
0000020 141 142 143 144 040 145 146 147 148 040 040 040 040 040 040 012
```

2) character (-c)

- -c option can be used to display the printable characters and its corresponding octal value.

```
$ od –bc file
od -bc ofile
0000000 141 142 143 144 040 145 146 147 148 040 040 040 040 040 040 012
          a   b   c   d       e   f   g   f                           \n
0000020 141 142 143 144 040 145 146 147 148 040 040 040 040 040 040 012
            a       b       c       d       e
```