UNIX PROGRAMMING
18CS56

Module 1

Introduction

UNIX Files

Module 1: 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 path names. 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.

Introduction

Operating System

OS is the software that manages the computer's hardware and provides a convenient and safe environment for running programs.

OS acts as an interface between programs and the hardware resources that these programs access.

OS is loaded into memory when a computer is booted and remains active as long as the machine is running.

UNIX is an OS

COMMAND INTERPRETER called SHELL is used in UNIX to interact with user.

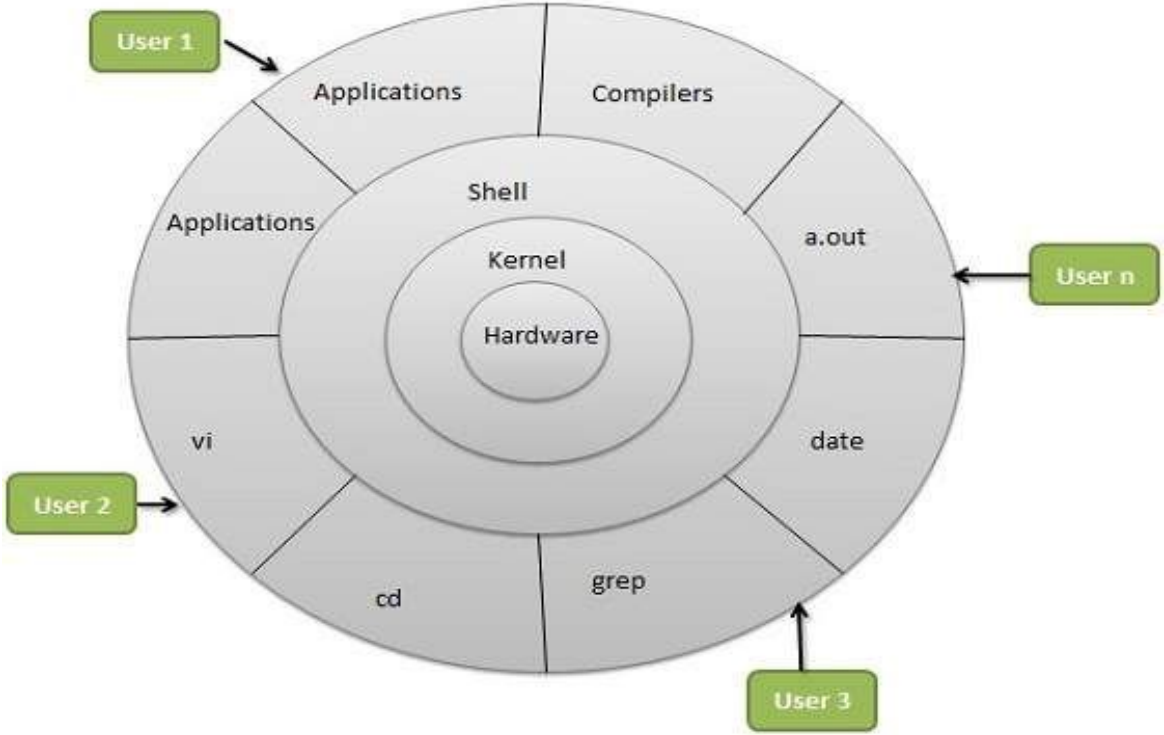
Commands are keyed in the form of words, which will be interpreted by Shell.

UNIX OS is security-conscious and the persons who maintain an account with it can only use it. List of accounts is maintained by OS.

UNIX Architecture

Division of Labor: Kernel and Shell

Kernel interacts with hardware and Shell interacts with user



UNIX Architecture

KERNEL

Kernel is the core of OS - collection of routines written in C.

Kernel routines are loaded into memory when the system is booted.

User programs that need to access hardware, will use the services of kernel. Programs access the kernel through a set of functions called *system calls*.

Kernel also manages the system's memory, schedules processes, etc.,

Kernel is often called ***"the OS"*** - a program's gateway to the computer's resources.

SHELL

Command interpreter is the major job that is performed by shell.

Converting the commands typed by user to kernel understandable form.

Shell is the outer part of OS, an interface between user and kernel.

UNIX Architecture

SHELL Contd…
A single kernel can have many shell's executing simultaneously.

When a command is entered via keyboard, shell thoroughly examines command for special characters.

If found, it rebuilds a simplified command, and finally communicates it with kernel for execution.

$ ls > dir.txt

ls command will list the files in current directory, > is a special character requesting to transfer file names to dir.txt file.

Shell after receiving this command will collect the names of the files in the current working directory and finally will pass on the same to dir.txt file.

UNIX Architecture

Note:

System's bootstrap program loads the kernel into memory at startup.

Shell is represented by **sh**(Bourne shell)**, csh**(C shell)**, ksh**(Korn shell) or **bash** (bourne again shell), one of these shells will be running to help users to log in.

Command to know which shell is running **echo $SHELL.**

Bash shell is the default shell used by UNIX OS.

The File and Process

File

Two simple entities support the UNIX system, File and Process.

"Files have places and processes have life"

A **file** can be informally defined as a collection of (typically related) data, which can be logically viewed as a stream of bytes (i.e. characters). A file is the smallest unit of storage in the Unix file system.

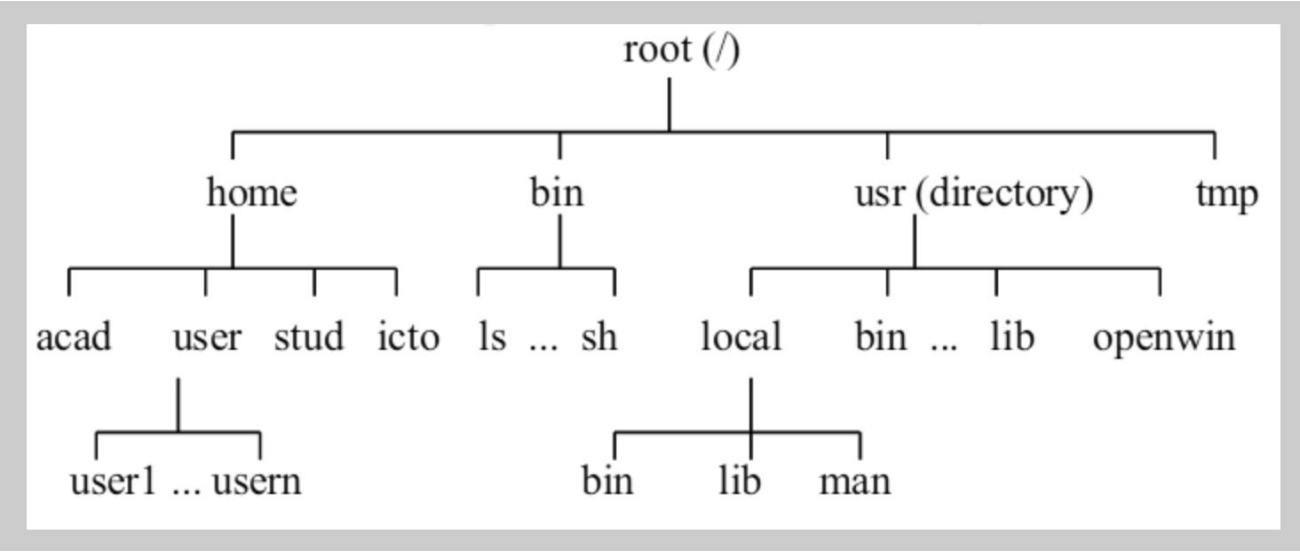
UNIX Architecture

A **file system** consists of files, relationships to other files, as well as the attributes of each file.

File are related to another file(!!!) by being part of a single hierarchical structure.

All files in the Unix file system can be categorized in 3 types, specifically:

Ordinary files Directory files Device files



UNIX Architecture

Process

Process is the name given to a file when it is executed as a program. Process is simply "time image" of an executable file.

Process like files also belong to a separate hierarchical tree structure.

Processes have parents, children and grandchildren. Processes will be created and destroyed.

UNIX provides tools that allow user to control processes, move them between *foreground* and *background* and even *kill* them.

$ pstree ; command to see the process hierarchy

$tree ; command to see the file structure hierarchy "sudo apt install tree"

UNIX Architecture

The System Calls

System calls provide the interface between a process and OS.

Several thousands of commands exists in UNIX OS.

These commands use a handful of functions, called system calls, to communicate with kernels.

UNIX system calls can be invoked directly from a C/C++ program.

Popular system calls are open , read , write , close , wait , exec , fork , exit , and kill.

System calls in Unix are used for file *system control*, *process control*, *interprocess communication* (IPC !!!) etc. Access to the Unix kernel is only available through these system calls. Generally, system calls are similar to function calls, the only difference is that they remove the control from the user process.

Currently there are around 80 system calls in the Unix OS interface.

Features of UNIX

Features of UNIX OS are

- **Multiuser System**
- **Multitasking System**
- **Building-block Approach**
- **UNIX Toolkit**
- **Pattern Matching**
- **Programming Facility**
- **Documentation**

Multiuser System

UNIX is a multiprogramming system. It permits multiple programs to run and compete for the attention of CPU, which can happen in two ways

- Multiple users can run separate jobs.
- A single user can also run multiple jobs.

Resources in UNIX are shared between all users.

Features of UNIX

Multiuser System Contd...

Computer breaks up a unit of time into several segments, and each user is allotted a segment. So at any point in time, machine will be executing the job of a single user.

The moment the allocated time expires, the previous job is suspended and the next users job is taken up. This process goes on until the clock has turned full-circle and the first user's job is considered once again.

Multitasking system

A single user can also run multiple tasks concurrently in UNIX OS.

Ex: A user can edit a file, print another one, send email and browse WWW, all these can be done without pausing or terminating any of the applications.

UNIX OS is designed to handle a user's multiple needs

Features of UNIX

Multitasking system Contd...

In multitasking environment, one job will be executing in foreground, the rest are executed in the background.

Jobs can be switched from background and foreground, suspend or even terminate them.

Background jobs are executed when, cpu is idle.

Building-block Approach

UNIX OS is made up of more simple commands that can be combined to make up a complex one, *"Small is beautiful"*.

It is via pipes and filters UNIX implements small is beautiful philosophy.

Commands in UNIX are **specialized** in solving one task rather than solving multiple tasks in one.

Features of UNIX

Multiuser system Contd...

Features of UNIX

Multitasking system Contd...

Features of UNIX

Building-block Approach Contd...

UNIX tools are designed with the requirement that the output of one tool be used as input to another.

UNIX commands are designed in such a manner that they must ensure that they will not throw out excessive verbiage and clutter the output. (one reason UNIX programs are not interactive)

Ex: If the output of **ls** command contained column headers or if it prompted the user for any information, this output could not have been used as useful input to the **wc** command.

UNIX Toolkit

UNIX represents the kernel, but the kernel by itself does not do much that can benefit users.

In order to use the real power of UNIX OS, host of applications are attached to it. The application programs are diverse in scope.

Ex: text manipulation utilities, compilers, interpreters, networked applications and system administration tools, etc.,

Features of UNIX

UNIX Toolkit

This host of applications is the one area that's constantly changing with every UNIX release. New tools are added and the older ones are removed are modified.

Pattern Matching

UNIX features very sophisticated pattern matching features.

Ex: ls chap*

Above command will list all the files in the current working directory, which starts with 'chap' and any characters further.

'*' is termed as **metacharacter** and one of the several other characters used in UNIX for pattern matching tasks.

Regular expression are the one's which is also framed by using these metacharacters and are widely used in pattern matching tasks.

Features of UNIX

Programming Facility

UNIX shell is a programming language. It has all the necessary structures like, control, loops and variables, which establishes itself as a powerful programming language.

These features are used to design *shell script* programs that can also invoke UNIX commands.

Many of the systems functions can be controlled and automated by using these shell scripts.
Shell scripting (Existing commands and some keywords) &
System programming (API's)

Documentation

The principal online help facility available is the **man** command, which remains the most important reference for commands and their configuration files.

Apart from the online documentation, there is a vast ocean of UNIX resources available on the internet.

POSIX and the Single UNIX specification

Early UNIX OS did not confine to portability characteristic of programs.

"Write once execute anywhere"

Later, *Portable Operating System Interface for Computer Environments (POSIX)* was developed by IEEE (Institute of Electrical and Electronics Engineers), which referred to operating systems in general based on UNIX.

Two of the most standards from POSIX family are known as POSIX.1 and POSIX.2. POSIX.1 specifies the C applications program interface - **System calls.** POSIX.2 deals with shell and utilities.

A joint initiative of X/Open and IEEE resulted in the unification of the two standards, termed as *Single UNIX Specification, Version 3 (SUSV 3)*, "Write once, adopt everywhere" means once program has been developed in an POSIX compliant system, it can be executed in another system which must also be POSIX compliant with slight modifications.

Locating Commands

UNIX commands are seldom more than 4 characters long. All commands are single words like ls, cat, who etc., which are in lowercase.

If an unknown command is entered, an error message will be flagged by **Shell** which is executing it.

Ex: $ LS

Bash: LS: command not found

There are some predetermined set of commands, that will be searched by shell before flagging an error message.

Commands are essentially files containing programs, mainly written in C. These files are stored in directories. To know the location of an executable program **type** command can be used.

Ex: $ type ls

ls is /bin/ls

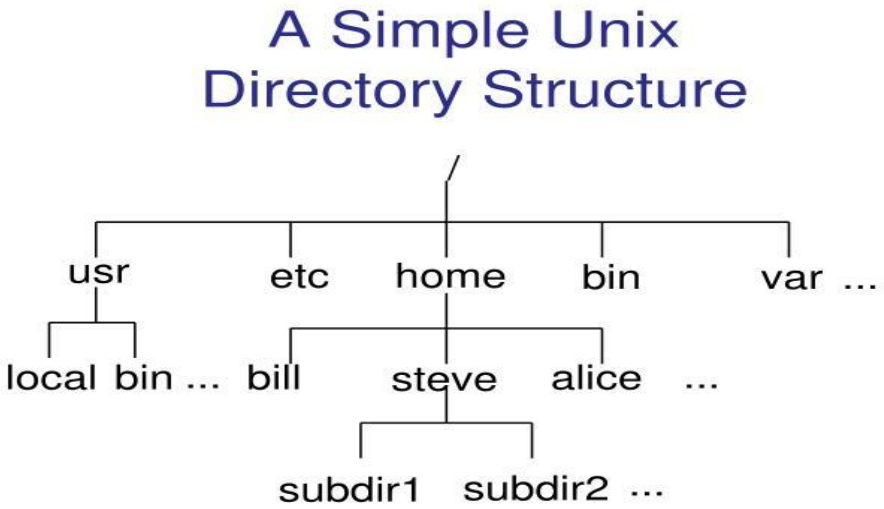
PATH

Sequence of directories that the shell searches to look for a command is specified in its PATH variable.

Ex: $ echo $PATH

/bin:/usr/bin:/usr/local/bin:/usr/ccs/bin:/usr/local/java/bin:

Above is a list of 5 directories, which are separated with colon.



Command Structure

Consider the command below

Ex: echo hello students

echo is considered as a command, *hello* and *students* are considered as arguments for the command.

Commands are made up of two parts 1. Command 2. Arguments

Arguments can be classified into two types

1. Known/Determined 2. Unknown/Not determined

Ex: ls -l

-l is an argument for the command ls, which designates long listing, this option prints all the information wrt the files in the current directory. -l is a determined or known argument from the shell perception.

But “hello students” is an unknown argument.

Basic UNIX Commands

echo

is used to display messages on terminal or to issue prompts for taking user input and also it is used to evaluate shell variables.

Ex: echo $SHELL, echo $PATH

Escape sequence is a two character-string beginning with a \ (backslash).

When escape sequence is placed in the string which is used an argument to echo, shell considers that as a directive and suitable action will be performed.

\c - places the cursor and prompts in the same line that display the output

$ echo -e “Enter filename: \c”

Enter filename: _

\t - A tab(8) space is provided.

\n - After printing the characters, cursor moves to new line.

Basic UNIX Commands

echo

NOTE: In Bash shell to see the effect of escape sequences echo command must be coded with -e option

Ex: echo -e “Hai,\t it is over\n”

ASCII characters can also be represented by their octal values. echo interprets a number as octal when it is preceded by \0.

Ex: echo -e ‘\07’

Generates a beep sound.

Escape Sequence	Significance
\a	Bell
\b	Backspace
\c	No newline
\f	Form feed
\n	New line

Escape Sequence	Significance
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\0n	ASCII character represented by the octal value n, where n can’t exceed 0377 (decimal value 255)

Basic UNIX Commands

printf

printf works similar to echo.

printf too accepts all escape sequences used by echo, but unlike echo, it does not automatically insert a newline unless the \n is used explicitly.

Format used in c, such as %f, %d … can be used in printf.

Ex: $ printf “My current shell is %s” $SHELL

%s - String

%30s - To print string in 30 character wide

%d - Decimal integer

%6d - To print decimal integer in a space of 6 characters wide

%o - Octal integer

%x - Hexadecimal integer

%f - Floating point number

Basic UNIX Commands

printf

Multiple formats can also be used in a single printf command. There must be corresponding argument for each format.

Ex: $ printf “Value of 255 in octal is %o and %x in hexa” 255 255

ls

List information about the FILEs (current directory by default).

Ex: ls

ls command without any options lists files and directories in a plain format without displaying much information like file types, permissions, modified date and time

Ex: ls -l

Lists the file in long list format, which includes name, size, permission, modified date and time etc.,

Ex: ls -r To list files in reverse order

Basic UNIX Commands

who

UNIX maintains an account of all users who are logged on to the system.

Ex: who

```
root     console     Aug  1  07:51   (:0)
Kumar    pts/10      Aug  1  07:56   (pc123.heavens.com)
```

First column displays usernames/user-ids of currently working on the system.
Second column displays device names of their respective terminals.
Ex: Kumar's terminal has the name pts/10, file named 10 in pts directory.

Third, fourth and fifth column displays date and time of logging in.
Last column displays machine name from where the user has logged in.

Users can log in remotely to a UNIX system, user kumar has logged in remotely and root is normal login.

Basic UNIX Commands

who

Ex: who -Hu

NAME	LINE	TIME	IDLE	PID	COMMENTS
root	console	Aug 1 07:51	0:48	11040	(:0)
kumar	pts/10	Aug 1 07:56	.	13678	(pc123.heavens.com)

argument ‘H’ is to display headers and ‘u’ to display users.

A ‘.’ in IDLE indicates the user is active.

PID determines Process-id, a number that uniquely identifies a process

Basic UNIX Commands

date

UNIX system maintains an internal clock meant to run perpetually. When the system is shut down, a battery backup keeps the clock ticking, which stores the number of seconds elapsed since the epoch (i.e January 1, 1970).

A 32-bit counter stores these seconds(except on 64-bit machine)
(counter will overflow sometime in 2038.)

Current date can be displayed using date command.

Date command can be formatted with format specifier as arguments. Each format is preceded by the + symbol, followed by % operator, and a single character describing the format.
Ex: $ date +%m prints the current month in numerical form.

Basic UNIX Commands

date

Ex: $ date +%h prints the name of the month.

Other format specifiers are
d - day of the month (1 to 31)
y - last two digits of the year
H, M & S - hour, minute and second, respectively
D - date in the format mm/dd/yy
T - time in the format hh:mm:ss

When multiple format specifiers are used it must be enclosed within quotes and by using a single + symbol before it.

Ex: $ date +"%h %m"
Aug 08

Basic UNIX Commands

passwd

passwd command changes passwords for user accounts.

A normal user may only change the password for his/her own account, while the superuser may change the password for any account.

Ex: $ passwd
Changing password for kumar
Enter login password: ******
New password: ******
Re-enter new password: ******
passwd (SYSTEM): passwd successfully changed for kumar

passwd command expects three input. First, it prompts for old password, next it checks whether valid old password has been entered, if done, it prompts for new password. After entering new password, the same has to be re-entered for confirmation purpose. If entered properly the new password is registered by the system.

Basic UNIX Commands

passwd

When password is entered, the string is *encrypted* by the system. Encryption generates a string of seemingly random characters that UNIX uses to determine the authenticity of a password.

Encrypted password is stored in a file named **shadow** in /etc directory.

Even if the user is able to see the encrypted password in the file, password cannot be decrypted.

cal

This command to see the calendar of any specific month or a complete year.

cal [[month] year] month is optional but not year

[ ] *represents optional*

Ex: cal display current month

cal 03 2006 display's March month of 2006

cal 2003 | more display's content in paused manner.

Combining Commands

UNIX allows more than one command to enter in the same line.

Ex: wc t.txt ; ls -l t.txt

wc counts lines, words and bytes of the file passed as an argument to it. There are two commands one before semicolon and another after semicolon.

First wc command will be executed and result will be displayed, then ls will be executed.

Output of a command can be also be redirected to a file.

Ex: (wc t.txt ; ls -l t.txt) > res.txt

Combined output of two commands will be sent to the file res.txt.

When a command line contains a semicolon, shell executes the command on each side of it separately. ';' is known as *metacharacter.*

Meaning of Internal and External command

Commands in UNIX can be categorized as internal and external commands.

Internal Commands

Commands which are built into the shell.
Executing the commands present in shell is fast, because the shell doesn't have to search the command in the path's present in PATH variable, and also no process needs to be spawned for executing it.
Ex: echo, cd etc.

External Commands

Commands which aren't built into the shell.
When an external command has to be executed, the shell looks for its path given in the PATH variable, and also a new process has to be spawned and the command gets executed.
They are usually located in /bin or /usr/bin.
Ex: to execute "cat" command, which usually is at /usr/bin,
the executable /usr/bin/cat gets executed.

type Command

type command is used to find out the information about a UNIX command.

It can be found whether the given command is an alias, shell built-in, file, function, or keyword using "type" command. Actual path of the command can also be found using type.

Ex: $ type mkdir
 mkdir is /bin/mkdir

The root login

root-user / **superuser** / **administrator**, is a special user account in UNIX, used for system administration.

It is the most privileged user on the Linux system and it has access to all commands and files.

root user can do many things an ordinary user cannot, such as installing new software, changing the ownership of files, and managing other user accounts.

The root login

It is not recommended to use **root** for ordinary tasks, such as browsing the web, writing texts, e.g. A simple mistake can cause problems with the entire system, for example if you mistype a command.

It is advisable to create a normal user account for such tasks.

If root permissions are needed on timely basis, *su* and *sudo* commands can be used.

su Command

su command, which is short for *substitute user* or *switch user*, enables the current user to act as another user during the current login session.

Syntax: su [*options*] [*username*]

If no username is specified, **su** defaults to becoming the superuser (root).

user will be prompted for a password, if password is appropriate, login will be successful. Invalid passwords produce an error message.

su Command

Ex: Consider two users in a unix system termed as ‘dos’ and ‘unix’, if currently a user has logged in as ‘dos’, he can change his identity using su command to ‘unix’, provided the password of unix is known to him.

```
dos$ su unix
Password: *****

unix$  lsa
```

UNIX Files

File is a container for storing information. File can be understood as sequence of characters.

In UNIX file does not contain eof (end-of-file) mark similar to DOS. File's size, file name are also not stored in the file.

All file attributes are kept in a separate area of hard disk, not directly accessible to programers, but only by kernel.

UNIX treats directories and devices as files. A directory is a folder where filenames and other directory names are stored within it.

All physical devices like HDD, memory, CD-ROM, printer are treated as files.

UNIX Files

Files are divided into 3 categories in UNIX

Ordinary file: or regular file, contains only data as a stream of characters

Directory file: Contains names and a number associated with each name (file or directory)

Device file: To read or write a device, operation has to be performed on its associated file.

File's attributes mainly depends on the type of file.

Ex: Read permission for an ordinary file, differs from the read permission on directory file.
Device file on the other hand is not a stream of characters.

Ordinary/Regular File

This is the most common file type. Ordinary files are divided into 2 types

1. Text file

2. Binary file

UNIX Files
Ordinary File
Text File
A text file contains only printable characters, which can be viewed and understood by users.
Ex: All C, java source programs, shell and perl scripts are text files.

A text file contains lines of characters where every line is terminated with *newline* character, also known as *linefeed(LF)*.

When Enter key is pressed, while inserting a text, LF character is appended to every line.

Command od is used to visualize LF characters in escape sequence format.
Ex: od -t c x.c
od is used to dump files in octal and other formats
-t c argument is used to select printable characters or backslash escapes

cat -e x.c command displays LF in $ format.

UNIX Files

Ordinary File

Binary File

Contains both printable and unprintable characters that belongs to ASCII range (0 to 255).

Most UNIX commands are binary files, and the object code and executables that are produced by compiling C programs are also binary files.

Picture, sound and video files are also binary files. These file contents cannot be displayed using **cat** command.

Directory File

A directory file contains no data, but it contains the details of the files and subdirectories.

Directories and subdirectories are created to group set of files pertaining to a specific application. This allows two or more files in different directories to have same name.

UNIX Files

Directory File

A directory file contains an entry for every file and subdirectory that is has. Each entry has two components

The filename

A unique identification number for the file or directory (called *inode* number)

A directory contains filename and not the file's contents.

User cannot write anything to a directory file, but can execute some commands, which makes the kernel write to a directory file. (touch , cat, mkdir for sub-directory creation)

Ex: When a file is created or removed, the kernel automatically updates its corresponding directory by adding or removing the entry (inode number and filename) associated with a file.

UNIX Files

Device File

Task of reading the contents from or writing the contents to peripheral devices are performed by reading or writing the file representing the device.

Perception of treating devices as files helps UNIX OS to use the same commands that are applicable for ordinary files.

Device filenames are generally found inside a directory structure /dev.

Device files are empty files.

Operations on device file is entirely governed by the attributes of the file. Kernel identifies a device from its attributes and then uses them to operate the device.

UNIX Files

File name

It is recommended that only the following characters be used in file names:

Alphabetic characters and numerals

Period (.), hyphen (-) and underscore (_).

UNIX imposes no rules for framing filename extension. A shell script need not have the .sh extension. In all the other cases, it is the application that imposes restriction.

C compiler expects C program filename to end with .c, oracle requires SQL script to have the .sql extension

A filename can have as many dots as needed; a.b.c.d.e is a perfectly valid filename. A filename can also begin and end with a dot

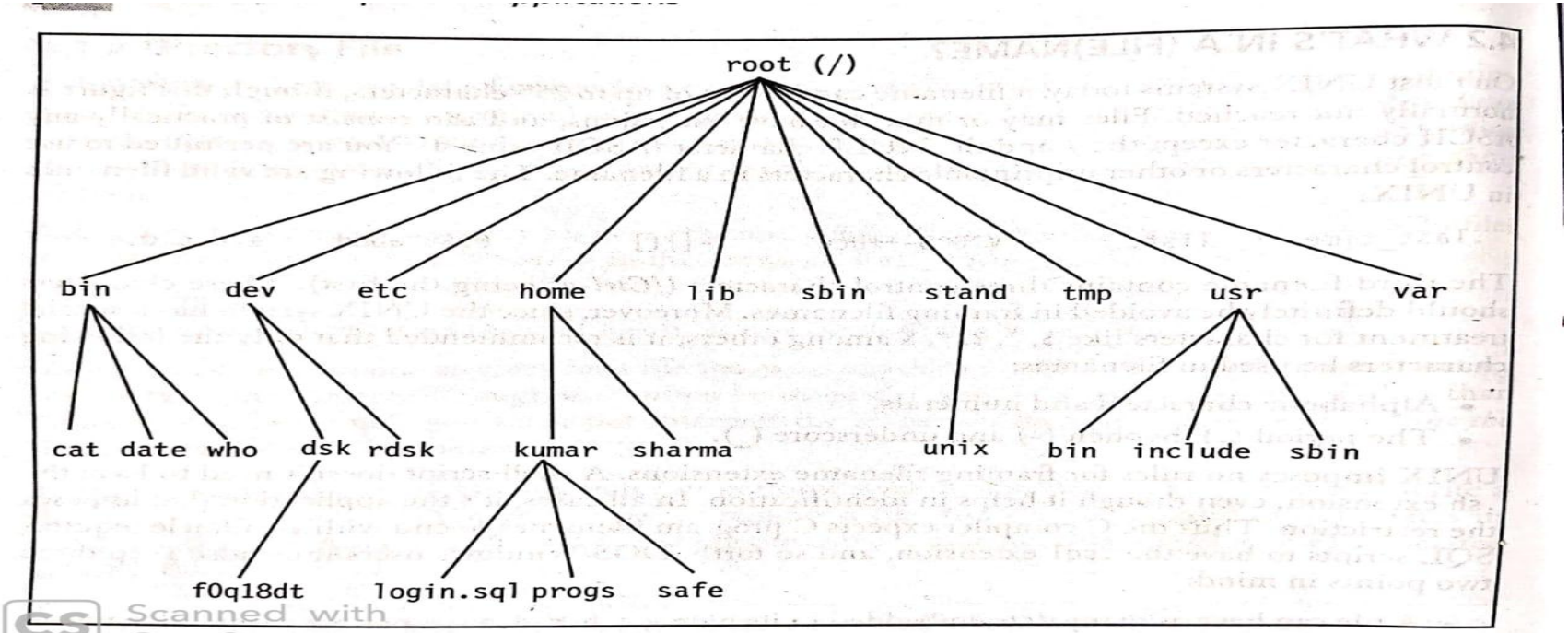
UNIX is sensitive to case; chap01, Chap01 and CHAP01 are three different filenames and its may exist in the same directory.

UNIX Files

Organization of Files

The Parent-Child relationship

File system in UNIX is a collection of all of these related files (ordinary, directory and device files) organized in a hierarchical (an inverted tree) structure.



Scanned with CamScanner

Fig. 4.1 The UNIX File System Tree

UNIX Files

Organization of Files: The Parent-Child relationship

Implicit feature of UNIX file system is that there is a top directory, which serves as reference point for all files.

This top directory is called *root* and is represented by a / (front slash).

Root is actually a directory, which is conceptually different from the user-id root used by the system administrator to log in.

Root directory has a number of subdirectories under it. These subdirectories in turn have more subdirectories and other files under them.
Ex: bin and usr are two directories under /, while cat is a file and kumar is subdirectory.

Every file, apart from root, must have a parent, and it must be possible to trace the ultimate parentage of a file to root.

Ex: *home* is the parent directory of *kumar* sub-directory, while / is the parent of *home* and the grand-directoryof *kumar*.

UNIX Files

Organization of Files: The Parent-Child relationship

If any files are created under the directory *kumar*, *kumar* will be the parent of those files.

It is observed in this parent-child relationship, the parent is always a directory. *Login.sql* is a file under directory *kumar* and cannot have directories under it.

Hidden Files

UNIX allows users to have files which are not listed, by default, by the **ls** command.

These are called hidden files and are distinguishable from other files by the fact that their file names begin with a dot (**.**).

Ex: Such a file is **.profile** which is executed every time you log in to the system.

Hidden files are listed by adding the **-a** option to the **ls** command.

UNIX Files
Standard directories
/ Root directory
Each and every file and directories are enlisted under Root directory.

/ bin
The /bin directory contains the essential user binaries (programs) that must be present when the system is mounted in single-user mode.
Ex: Applications such as Firefox are stored in /usr/bin

While important system programs and utilities such as the bash shell are located in /bin.

/boot – Static Boot Files
This directory contains the files needed to boot the system
Ex: GRUB boot loader files and UNIX kernels are stored here.

UNIX Files

Standard directories

/etc – Configuration Files

This directory contains configuration files, which can be edited in a text editor.
/etc/ directory contains system-wide configuration files.
User-specific configuration files are located in each user's home directory.

The home directory and the HOME variable

When a user login to UNIX system, it provides a default directory called **home directory.**

home directory is created by the system when a user account is created.

shell variable HOME stores the home directory path
Ex: $ echo $HOME
/home/kumar path displayed is **absolute pathname**,

An absolute pathname shows a file's location with reference to root. First / represents root and directory kumar is placed two levels below root.

UNIX Files

The home directory and the HOME variable

A file can be referred using

$ls $HOME/x.txt

In which case the default home path is attached to the file location.

If the file x.txt is stored in /home/kumar then this technique is useful. If there is a subdirectory in kumar say kumar1 and file is present within it, then the following command cannot be used.

Alternate to $HOME is ~ symbol which can be replaced in command.

~ can be used to refer to any other users home directory

Ex: Consider user rajan the command

$ ls ~rajan/x.txt will refer to home directory of rajan

A tilde followed by / (like ~/foo) refer to one's own directory, but when followed by a string (~rajan) refers to the home directory of that user represented by the string.

UNIX Files

The home directory and the HOME variable

Ex:

```
$ pwd
/home/kumar
$ mkdir kumar1  ;  cd kumar1
$ pwd
/home/kumar/kumar1
$ cat > t.txt
A
B
^z
$ cd .. ; pwd
/home/kumar
$ cat ~/kumar1/t.txt
A
B
```

UNIX Files

Manipulating the PATH

Environment variables are used to enhance and to standardize shell environment on UNIX systems.

There are standard environment variables that the system sets up for user, user can also set up their own environment variables, or optionally change the default ones to meet their needs.
Ex: $ env
Above command lists the environment variables.

PATH is a prominent environment variable in shell.
PATH variable contains the search path for executing commands and scripts.

PATH value can be updated as shown below
PATH=$PATH:/home/dev/test
The above command adds one more path /home/dev/test to PATH variable.

UNIX Files
The PATH variable

Ex:
$ pwd
/home/kumar
$ mkdir kumar1 ; cd kumar1
$ pwd
/home/kumar/kumar1
$ cat > test.c
# include <stdio.h>
int main( ) {
printf("C program"); }^z
$ cc test.c
$ cd ..
$ pwd
/home/kumar
$
$ rm a.out
$ a.out // not ./a.out

a.out, which is present in /home/kumar/kumar1 will be executed, because that path has been added to PATH variable.

Moreover ./a.out searches for a.out in present working directory and PATH value, will not be considered.

UNIX Files: Relative and Absolute path names

Absolute path

Absolute path starts from the directory root (/) and spans up to the actual file or directory.

It contains the names of all directories that come in the middle of the directory root and the actual file or directory. Considering, pwd as root.

Ex: $ ls /home/dev/test will list all the files present in directory test.

Key points

First forward slash (/) in the absolute path represents the directory root. Besides this, all slashes in the path represent the directory separator.

Besides the last name, all names in the absolute path belong to directories. Last name can belong to file or sub-directory.

In the absolute path, directories names are written in their hierarchy order. Parent directory's name is written in the left side.

To know the absolute path of the current directory, the command pwd can be used.

UNIX Files

Absolute path variable : Exercise to list files using absolute path

Ex:

```
$ pwd
/home/kumar
$ mkdir kumar1  ;  cd kumar1
$ pwd
/home/kumar/kumar1
$ cat > t.txt
ABCDE^z
$ cd /
$ pwd
/
$ ls /home/kumar/kumar1
a.c  a.out
```

UNIX Files: Relative and Absolute path names

Relative path

Relative path starts from the current directory and goes up to the actual file/directory.

When present working directory is changed, relative path also changes.

Just like the absolute path, the name of the parent directory is written in the left side, and all slashes in the relative path represent the directory separator.

Single dot (.) and double dots (..) in UNIX

In UNIX every directory contains two dots; single dot (.) and double dots(..).

When a directory is created, both the single dot and the double dots are also automatically created in it. By default these dots are hidden and do not show in the output of the command ls.

The single dot refers to the directory itself and the double dots refers to its parent directory or the directory that contains it.

UNIX Files: Relative and Absolute path names

Single dot (.) and double dots (..) in UNIX

Shell allows us to access the current directory and the parent directory by using the single dot and the double dots respectively.

Relative path also uses these dots to represent the current directory and the parent directory respectively. With the use of these dots, relative path can be built for any file or directory from the current directory.

Ex:
$ pwd
/home/kumar/kumar1
$ ls ../../../
lists the files present in root directory

Ex:
$ pwd
/home/kumar
$ cat > ./kumar1/t.txt
ABCD^Z
$ cat ./kumar1/t.txt
ABCD

Directory Commands

pwd

pwd command is a command line utility for printing the current working directory.

It will print the full system path of the current working directory to standard output.

In most shells pwd is a shell builtin. Command is present in the shell rather than calling an external program. Code will run significantly faster than calling an external executable.

Ex:$ type pwd
pwd is shell builtin

$ echo $PWD // Environment variable PWD

$ pwd //command, which will do the task of ‘echo $PWD’

Directory Commands

cd

The *cd* command is used to change the *current directory* i.e., the directory in which the user is currently working.

Syntax: cd [options] [Directory]

Ex: $ cd ~ changes to home directory, the default directory which is allocated to user during user creation process

$ cd / changes to root directory

$ cd .. changes to parent directory

$ mkdir test\ 1 creates a directory with name ‘test 1’, which contains blank.

$ cd ‘test 1’ changes to ‘test 1’ directory.

$ cd test\ 1

Directory Commands: mkdir

mkdir allows users to create directories. The command is followed by names of the directories to be created.

Ex: $ mkdir patch directory patch is created under the current working directory.

$ mkdir patch dbs doc creates 3 subdirectories with one command

$ mkdir app app/progs app/data creates directory trees.

First app directory will be created and then subdirectories within it.

$ mkdir test

mkdir: Failed to make directory "test"; Permission denied

This can happen due to these reasons:

Directory 'test' may already exist.

There may be an ordinary file name that matches the directory name.

Permissions set for current directory does not permit the creation of file and directories by the user.

Directory Commands: rmdir

rmdir command removes directories.

Ex: $ rmdir app Considering the app directory is empty app directory will be removed. If app directory contains at least one file command generates error.

rmdir: failed to remove 'app': Directory not empty

$mkdir test test/test1

$rmdir test/test1 test

First subdirectories with test has to be deleted and then test.

A user is the owner of his home directory, and can create and remove subdirectories or files, in this directory or in any subdirectories created.

mkdir and rmdir commands work only in directories owned by the user.

User creating directory or files in other user's directory depends on permissions and ownership.

File related Commands

cat: Displaying and Creating Files

This command is used to display the contents of a file on the terminal.

$ cat x.c displays the content of the file x.c

$ cat x.c x1.c displays the content of the file x.c first followed by the contents of x1.c

-v and -n arguments of cat

-v : $ cat -v a.out

cat command will be usually used to display contents of text file only. If executable file contents are displayed using cat non-printable ASCII characters might be displayed.

Non-printable characters are used to indicate certain formatting actions, such as:
White spaces (considered an invisible graphic), Carriage returns, Tabs, Line breaks, Page breaks, Null characters.

-v option of cat helps to display these non-printable characters.

-n : $ cat -n a.out

option displays line number in addition to content.

File related Commands

cat

cat command is also used for creating file.

$ cat > t.txt cat followed by > (right chevron) character and the filename

ABC *[Enter]*

DEF^d^d [Ctrl-d] signifies the end of input to the system.

cat is a versatile command.

It can be used to create, display concatenate and append to files.

cat command is not restricted only to files, but it can also act on stream.

Output of other command can also be fed into cat.

$ ls -l > cat > t.txt

File related Commands

mv: Move Files and Rename Files

This command performs two distinct functions.

It renames a file (or directory) It moves a group of files to a different directory.

mv will not create a copy of file, it merely renames it. No additional disk space is consumed during renaming.

$ mv t.txt T.TXT

If the destination file does not exist, it will be created. By default, mv does not prompt for overwriting the destination file if it exists.

Group of files can be moved to a directory.

$ mv t.txt t1.txt t2.txt progs 3 files will be moved to directory named progs

Directory name can also be renamed using mv

$mv progs programs considering progs as directory name, it will be renamed as programs

File related Commands

rm: Deleting Files

rm command is used to delete one or more files. It will not prompt user before deletion, hence must be used with utmost care.

$ rm t.txt t1.txt t2.txt deletes the 3 files that are mentioned in the command

A file once deleted cannot be recovered.

File name in rm command can be prefixed with path too in order to delete files.

$ rm test/t.xt test/t1.txt test/t2.txt

All files in a directory can be deleted at once $ rm *

Deleting files from a directory, depends on the permission that is associated with the directory for a particular user.

$ rm t*.*

File related Commands

rm

Interactive Deletion (-i)

-i option associated with rm, prompts for user confirmation before a file is deleted.

$ rm -i t.txt

remove t.txt ?

Response as y, removes the file, any other response leaves the file un-deleted.

Recursive Deletion (-r)

Usually rm command will not remove subdirectory within the PWD, but by using option -r, subdirectories can also be deleted.

$ rm -r * deletes all files and subdirectories in PWD, without any prompt

$ rm -i -r * prompts the user upon each file/subdirectory deletion.

File related Commands

rm : **Forcing removal (-f)**

Attempt to remove the files without prompting for confirmation, regardless of the file's permissions.

If the file does not exist, do not display a diagnostic message or do not modify the exit status to reflect an error.

$ rm res.txt // assume res.txt does not exist
$ echo $? // to print the return value of the previous process
1 *; exit status 1 indicates failure of the command*
$ rm -f res.txt ; echo $?
0 *; exit status not modified*
-f option overrides any previous -i options.
$ rm -i -f t ; file ‘t‘ will be deleted without prompting user for deletion.

NOTE: Make sure before rm * is used, be doubly sure before using rm -rf *. The first command removes only ordinary files in pwd. The second one removes everything - files and directories alike.

File related Commands

cp: Copying a File

cp command copies a file or a group files. It creates an exact image of the file on disk with a different name.

Syntax requires at least two file names in the command line. When both are ordinary files, the first is copied to the second.

$ cp src.txt dest.txt contents of src.txt will be copied to dest.txt

If destination file does not exist, it will first be created before copying.
If destination file exists, its contents will be overwritten without any warning.

If there is only one file to be copied, the destination can be either a file in a directory (wherein the file mentioned will have path associated with it) or directory. If directory name is mentioned source file will be copied to the mentioned directory.

$ cp src.txt test1/dest.txt dest.txt will be created in test1 and src.txt contents gets copied
$ cp src.txt test1 src.txt file will be copied to test1 subdirectory

File related Commands

cp

$ cp test1/src.txt .

$ cp test1/src.txt dest.txt

First command copies src.txt from test1 subdirectory to pwd.

Second command copies src.txt content from test1 subdirectory to the file named dest.txt and the file dest.txt will be in PWD.

$ cp t t1 t2 res

Above command copies 3 different files to subdirectory res, by retaining the same names. If these files are already existing in res, they will be overwritten. Finally, res subdirectory must exist for the above command to execute.

$ cp t* res

Copies all the files starting with character t and any other characters further to the directory res.

File related Commands

cp: Interactive Copying (-i)

-i option prompts user before overwriting the destination file, if it exists.

$ cp -i t t1

overwrite t1 (y/n)

Copying Directory Structure (-R)

UNIX commands are capable of **recursive** behaviour, which means that a command can descend a directory and examine and apply the same command to all the files in its subdirectories.

-R command behaves recursively to copy an entire directory structure.

$ cp -R test check

Consider check is not an existing directory

Copies entire files and subdirectories and its contents in the directory test, including test directory too, to a new directory check. In other words whole directory structure of test will be copied to a new directory structure check.

File related Commands

Copying Directory Structure (-R)

$ cp -R test check

In -R mode, cp will continue copying **even if errors are detected**.

Consider check is an existing directory

Then test directory structure will be copied under check directory. There will be two copies of test directory structure, one within check directory and the other test directory structure in pwd.

wc: Counting Lines, Words and Characters

wc command is used to count number of lines, words and characters in a file. This command takes one or more files names as arguments and displays a four-columnar output.

$ cat > infile

I

Am executing

wc command^d^d

File named infile is created and the sentence keyed in will be stored in it.

File related Commands

wc: Counting Lines, Words and Characters

```
$ wc infile
2      5     19  infile
```

A **line** is any group of characters not containing a newline.

A **word** is a group of characters not containing a space, tab or newline.

A **character** is the smallest unit of information, and includes a space, tab and newline.

wc offers 3 options to make a specific count.

-l option to count only number of lines	wc -l infile
-w option to count only number of words	wc -w infile
-c option to count only number of characters	wc -c infile

When used with multiple filenames, wc produces a line for each file, as well as total count.

```
$ wc infile  t   c
4     5     6    infile
6     7     8    t
3     4    10    c
```

File related Commands

od: Displaying data in octal (Octal Decimal hex ASCII dump)

Files in UNIX containing non-printing characters can be displayed using this command, od command prints these characters in readable form (ASCII octal values).

```
$ cat > infile
I
Am the
wc command^d          file infile is made up of escape sequences

$ od infile
0000000    005151  066541  072040  062550  073412  020143  067543  066555
0000020    067141  000144
0000023
```

Each line displays 16 bytes of data in octal, preceded by the offset in the file of the first byte in the line.

File related Commands

od

In order to understand the output -b and -c arguments will be used.

$ od -bc infile

```
0000000   151 012 141 155 040 164 150 145 012 167 143 040 143 157 155 155
            i  \n   a   m       t   h   e  \n   w   c       c   o   m   m
0000020   141 156 144
            a   n   d
0000023
```

Each line is having two displays.

Octal representation are displayed in the first line.

The printable characters and escape sequences are displayed in the second line.

-b for displaying octal bytes
-c for c-style escape characters.