MODULE 1

UNIX ARCHITECTURE AND COMMANDS

What is an Operating System?

- 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.

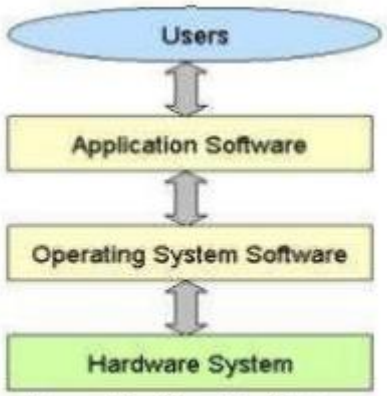


Figure 1.1 : Operating System

- 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.

UNIX ARCHITECTURE

- The UNIX operating system (OS) consists of a kernel layer, a shell layer , an application layer & files.
- 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:
  - Memory management
  - Process management (using commands: kill, ps, nohup)
  - File management (using commands: rm, cat, ls, rmdir, mkdir)
- To access the hardware, user programs use the services of the kernel via system calls.

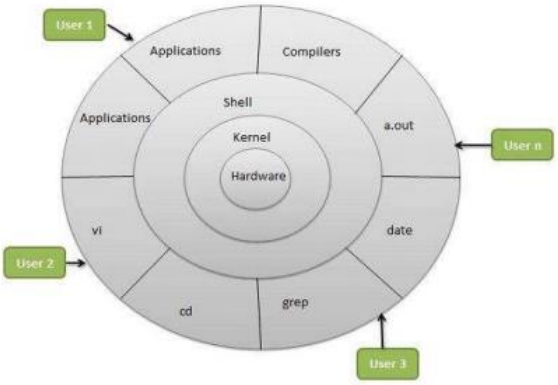


Figure 1.2 : Unix Architecture

- 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:
    - interprets the commands the user types in and
    - 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 UNIXOS.
  - 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.

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 shellscripts.

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.

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.

GENERAL FEATURES OF UNIX COMMANDS

- 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 // -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 -.

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 alloptions.
- 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.

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

UNDERSTANDING SOME BASIC COMMANDS

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

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
```

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
```

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
```

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 formatspecifiers:
  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
```

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
$
```

cal

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

• Syntax:

cal [[month] 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
```

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

• Example:

```
$ cal 9 2017
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

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.

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.

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.

ESSENTIAL SYSTEM ADMINISTRATION

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.

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.

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.

/etc/passwd and /etc/shadow Files

• There are four main user administration files:

1) /etc/passwd: Keeps the user account and password information. This file holds the majority of information about accounts on the Unix system.

2) /etc/shadow: Holds the encrypted password of the corresponding account.

3) /etc/group: This file contains the group information for each account.

4) /etc/gshadow: This file contains secure group account information.

1) /etc/passwd

• This file contains following user account information:

1) Username

 This is the name used for logging into a UNIXsystem.

2) Password

 This stores the encrypted password which looks like ******.

3) UID

 This is user's numerical identification.

 No two users can have the same UID.

4) GID

 This is user's numerical group identification.

5) Comments or GCOS

 This is contains user details or name address.

 This name is used at the front of the email address for this user.

6) Home Directory

 The directory where the user ends up on logging in.

 The login program reads this field to set the variable HOME.

7) Login shell

 This is the first program executed after logging in.

 This is usually the shell (/bin/ksh).

 The login program reads this field to set the variable SHELL.

2) /etc/shadow

• This file contains encrypted password of the corresponding account.

• This is the control file used by passwd to verify the correctness of a user's password.

• For every line in /etc/passwd, there's a corresponding entry in /etc/shadow.

• As a regular user, you do not have read or write access to this file for security reasons.

• Only superuser can access this file.

THE UNIX FILE SYSTEM

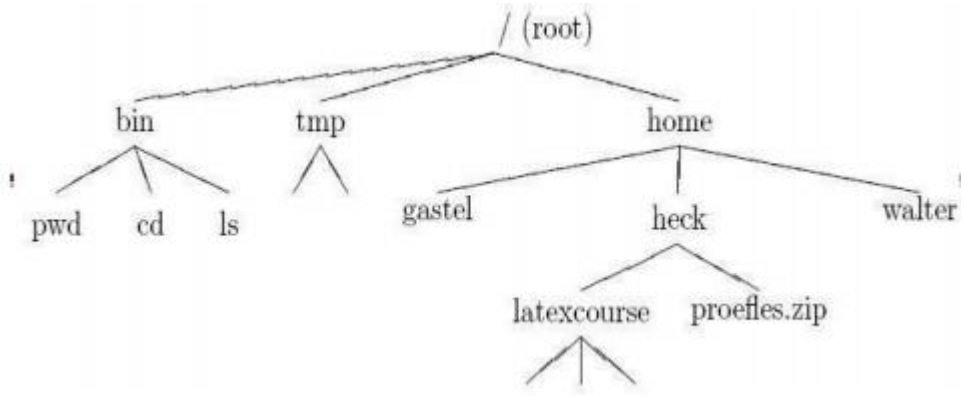
Types of files

- 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.
- The types of files in UNIX are as follows**:**
    - Ordinary (Regular) File :
        - This is the most common file type.
        - It 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 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.

- **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.

- 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.
- 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.

- **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.



- **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 , this is an absolute pathname, which is a sequence of directory names starting from root (/).
- The subsequent slashes are used to separate the directories.

- **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

- **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) .

- **mkdir - make a directory**
    - it is possible to extend home hierarchy by making sub-directories underneath it. This is done with the mkdir, make directory, command.
    - It is specified either through 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 current directory and progs and data as subdirectories under pis
    - The order of specifying arguments should be such that 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.

- **rmdir - remove directory**
    - A directory needs to be empty before it can removed.
    - If it’s not, the files need to be removed first.
    - Also, a directory cannot be removed if it is the present working directory; So, it must first be changed out of that directory.
    - A subdirectory can be removed unless it is placed in a directory which is hierarchically above the one 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.

ABSOLUTE PATHNAME

- Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.
- Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.
- Following are some examples of absolute filenames.

  /etc/passwd
  /users/kumar/progs/cprogs
  /dev/rdsk/Os3
- Example
  - date command can executed in two ways as

    $date // Relative path
    Thu Sep 7 10:20:29 IST 2017
    $/bin/date // Absolute path
    Thu Sep 7 10:20:29 IST 2017

RELATIVE PATHNAME

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

  progs/cprogs
  rdsk/Os3

Using . and .. in relative path name

- User can move from working directory /home/kumar/progs/cprogs to home directory /home/kumar using cd command like

  $pwd
  /home/kumar/progs/cprogs
  $cd /home/kumar
  $pwd
  /home/kumar
- Navigation becomes easy by using common ancestor.
- . (a single dot) - This represents the current directory
- .. (two dots) - This represents the parent directory
- Assume user is currently placed in /home/kumar/progs/cprogs

  $pwd
  /home/kumar/progs/cprogs
  $cd ..
  $pwd
  /home/kumar/progs
- This method is compact and easy when ascending the directory hierarchy. The command cd .. Translates to this ―change your current directory to parent of current directory‖.
- The relative paths can also be used as:

```
$pwd

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

- The following command copies the file prog1.java present in javaprogs, which is present is parent of current directory to current directory.

```
$pwd
/home/kumar/progs/cprogs
$cp ../javaprogs/prog1.java .
```

- Now prog1.java is copied to cprogs under progs directory.

THE PATH ENVIRONMENT VARIABLE

- Environmental variables are used to provide information to the programs you use.
- One such variable is called HOME.
- A command runs in UNIX by executing a disk file.
- When a command like date is specified, 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.

```
$echo $PATH
as /bin/myprog.
```

- Whenever a UNIX command is entered, it actually specifies 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.

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).

o If the file is a symbolic link, an arrow (-->) precedes the pathname of the linked-to file.

o 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-xr-x 512 Jun 7 11:12 .
rwxr-xr-x 512 May 29 09:59
-rwxr-xr-x 532 May 20 15:31 cshrc

THE UNIX FILESYSTEM

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.

FILE RELATED COMMANDS:

cat: DISPLAYING AND CREATING FILES

- cat command is used to display the contents of a small file on the terminal.

```
$ cat cprogram.c
# include <stdioh>
void main ()
{
 Printf(—hello‖);
}
```

- As like other files cat accepts more than one filename as arguments

```
$ cat ch1 ch2
```

It contains the contents of chapter1
It contains the contents of chapter2

- In this the contents of the second files are shown immediately after the first file without any header information. So cat concatenates two files- hence its name.

cat OPTIONS

- Displaying Nonprinting Characters (-v)
  - cat without any option it will display text files. Nonprinting ASCII characters can be displayed with –v option.
- Numbering Lines (-n)
  - -n option numbers lines. This numbering option helps programmer in debugging programs.

Using cat to create a file

- cat is also useful for creating a file. Enter the command cat, followed by > character and the filename.

```
$ cat > new
This is a new file which contains some text, just to
Add some contents to the file new
[ctrl-d]
$_
```

- When the command line is terminated with [Enter], the prompt vanishes. Cat now waits to take input from the user. Enter few lines; press [ctrl-d] to signify the end of input to the system .To display the file contents of new use file name with cat command.

$ cat new

This is a new file which contains some text, just to

Add some contents to the file new

cp: COPYING A File

- The cp command copies a file or a group of files. It creates an exact image of the file on the disk with a different name. The syntax takes two filename to be specified in the command line.
- When both are ordinary files, first file is copied to second.
  $ cp csa csb
- If the destination file (csb) doesn‘t exist, it will first be created before copying takes place. If not it will simply be overwritten without any warning from the system.
- Example to show two ways of copying files to the cs directory:
  $ cp ch1 cs/module1 ch1 copied to module1 under cs
  $ cp ch1 cs ch1 retains its name under cs
- cp can also be used with the shorthand notation, .(dot), to signify the current directory as the destination. To copy a file „new‟ from /home/user1 to your current directory, use the following command:
  $cp /home/user1/new new destination is a file
  $cp /home/user1/new . destination is the current directory
- cp command can be used to copy more than one file with a single invocation of the command. In this case the last filename must be a directory.
- Ex: To copy the file ch1,ch2,ch3 to the module , use cp as **$ cp ch1 ch2 ch3 module**
- The files will have the same name in module. If the files are already resident in module, they will be overwritten. In the above diagram module directory should already exist and cp doesn‘t able create a directory.
- UNIX system uses * as a shorthand for multiple filenames.
- Ex:
  $ cp ch* usp Copies all the files beginning with ch .

cp options

- Interactive Copying(-i) : The –i option warns the user before overwriting the destination file, If unit 1 exists, cp prompts for response
  $ cp -i ch1 unit1
  $ cp: overwrite unit1 (yes/no)? Y
  A y at this prompt overwrites the file, any other response leaves it uncopied.

Copying directory structure (-R) :

- It performs recursive behaviour command can descend a directory and examine all files in its subdirectories.
- -R : behaves recursively to copy an entire directory structure
  $ cp -R usp newusp
  $ cp -R class newclass
- If the newclass/newusp doesn‘t exist, cp creates it along with the associated subdirectories.

rm: DELETING FILES

- The rm command deletes one or more files.
- Ex: Following command deletes three files:
  $ rm mod1 mod2 mod3
- Can remove two chapters from usp directory without having to cd
- Ex:
  $rm usp/marks ds/marks
- To remove all file in a directory use *
  $ rm *
  Removes all files from that directory

rm options

- Interactive Deletion (-i) : Ask the user confirmation before removing each file:
  $ rm -i ch1 ch2
  rm: remove ch1 (yes/no)? ? y
  rm: remove ch1 (yes/no)? ? n [Enter]
  A ‗y‘ removes the file (ch1) any other response like n or any other key leave the file undeleted.
- Recursive deletion (-r or -R): It performs a recursive search for all directories and files within these subdirectories. At each stage it deletes everything it finds.
  $ rm -r * Works as rmdir
  It deletes all files in the current directory and all its subdirectories.
- Forcing Removal (-f): rm prompts for removal if a file is write-protected. The -f option overrides this minor protection and forces removal.
  rm -rf* Deletes everything in the current directory and below.

mv: RENAMING FILES

- The mv command renames (moves) files. The main two functions are:
  - It renames a file(or directory)
  - It moves a group of files to different directory
- It doesn't create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming.
- Ex: To rename the file csb as csa we can use the following command
  $ mv csb csa
- If the destination file doesn‘t exist in the current directory, it will be created. Or else

it will just rename the specified file in mv command.

- A group of files can be moved to a directory.
- Ex: Moves three files ch1,ch2,ch3 to the directory module
  $ mv ch1 ch2 ch3 module
- Can also be used to rename directory
  $ mv rename newname
- mv replaces the filename in the existing directory entry with the new name. It doesn't create a copy of the file; it renames it
- Group of files can be moved to a directory
  mv chp1 chap2 chap3 unix

wc: COUNTING LINES,WORDS AND CHARACTERS

- wc command performs Word counting including counting of lines and characters in a specified file. It takes one or more filename as arguments and displays a four columnar output.
  $ wc ofile
  4 20 97 ofile
- Line: Any group of characters not containing a newline
- Word: group of characters not containing a space, tab or newline
- Character: smallest unit of information, and includes a space, tab and newline
- wc offers 3 options to make a specific count.
  –l option counts only number of lines,
  -w and –c options count words and characters, respectively.
  $ wc -l ofile
  4 ofile
  $ wc -w ofile
  20 ofile
- Multiple filenames, wc produces a line for each file, as well as a total count.
  $ wc -c ofile file
  97 file
  15 file
  112 total

od: DISPLAYING DATA IN OCTAL

- od command displays the contents of executable files in a ASCII octal value.
- $ more ofile
  this file is an example for od command
- ^d used as an interrupt key
- -b option displays this value for each character separately.
- Each line displays 16 bytes of data in octal, preceded by the offset in the file of the
- first byte in the line.
  $ od –b file

```
0000000 164 150 151 163 040 146 151 154 145 040 151 163 040 141 156 040
0000020 145 170 141 155 160 154 145 040 146 157 162 040 157 144 040 143
0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144 040 141
0000060 163 040 141 156 040 151 156 164 145 162 162 165 160 164 040 153
0000100 145 171
```

- -c character option
  - Now it shows the printable characters and its corresponding ASCII octal representation
    $ od –bc file

```
od -bc ofile
0000000 164 150 151 163 040 146 151 154 145 040 151 163 040 141 156 040
          T   h   i   s       f   i   l   e       i   s       a   n
0000020 145 170 141 155 160 154 145 040 146 157 162 040 157 144 040 143
          e   x   a   m   p   l   e       f   o   r       o   d       c
0000040 157 155 155 141 156 144 012 136 144 040 165 163 145 144 040 141
          o   m   m   a   n   d  \n   ^   d       u   s   e   d       a
0000060 163 040 141 156 040 151 156 164 145 162 162 165 160 164 040 153
          s       a   n       i   n   t   e   r   r   u   p   t       k
0000100 145 171
          e   y
```

- Some of the representation:
  - The tab character, [ctrl-i], is shown as \t and the octal vlaue 011
  - The bell character , [ctrl-g] is shown as 007, some system show it as \a
  - The form feed character,[ctrl-l], is shown as \f and 014
  - The LF character, [ctrl-j], is shown as \n and 012
  - Od makes the newline character visible too.