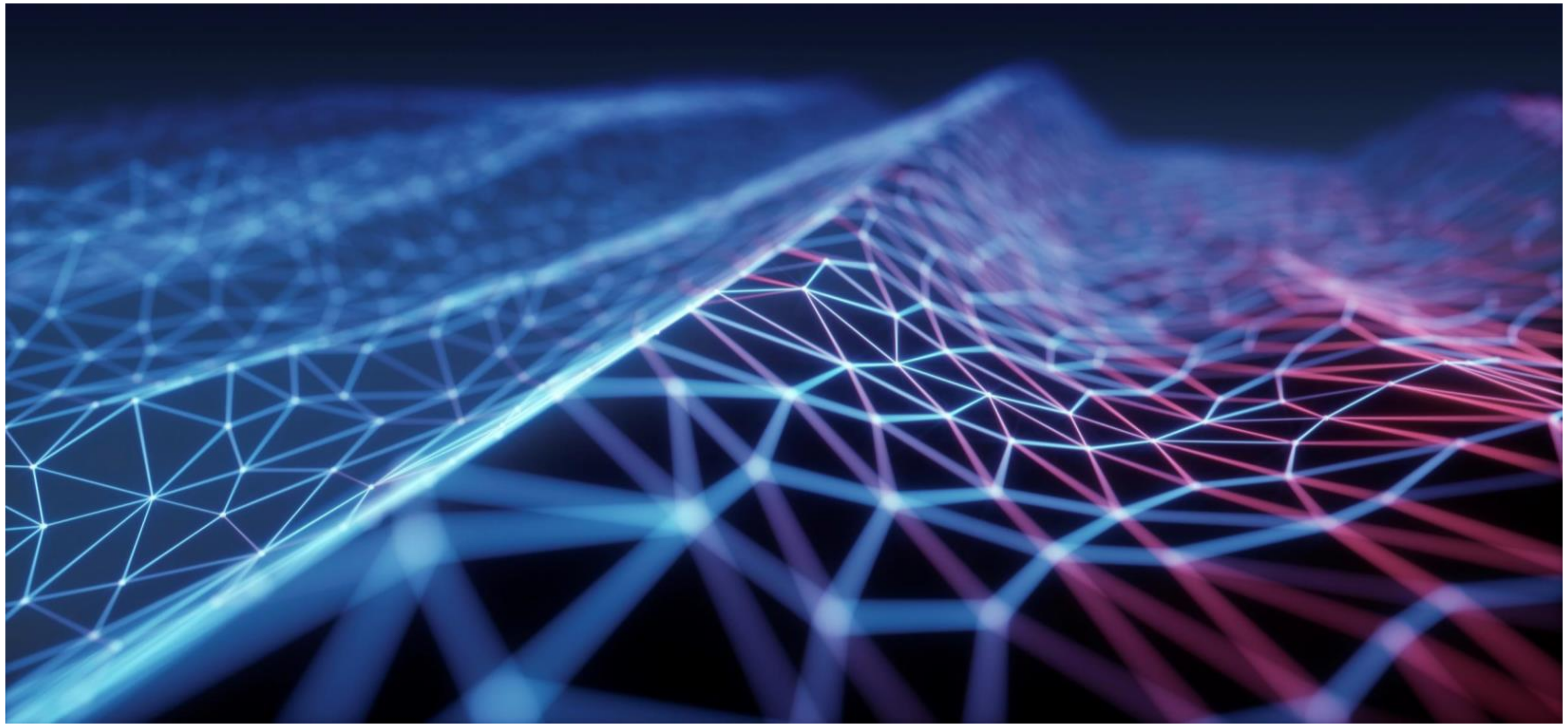


Vidyayāmruthamashnuthe

UNIX PROGRAMMING – MODULE I

AKASH HEGDE

ASSISTANT PROFESSOR

DEPARTMENT OF ISE

MODULE I

Introduction	About UNIX
	Architecture of UNIX
	Features of UNIX
	UNIX commands
	Basic commands
UNIX Files	File types
	Directories
	HOME and PATH variables
	Directory commands
	File-related commands

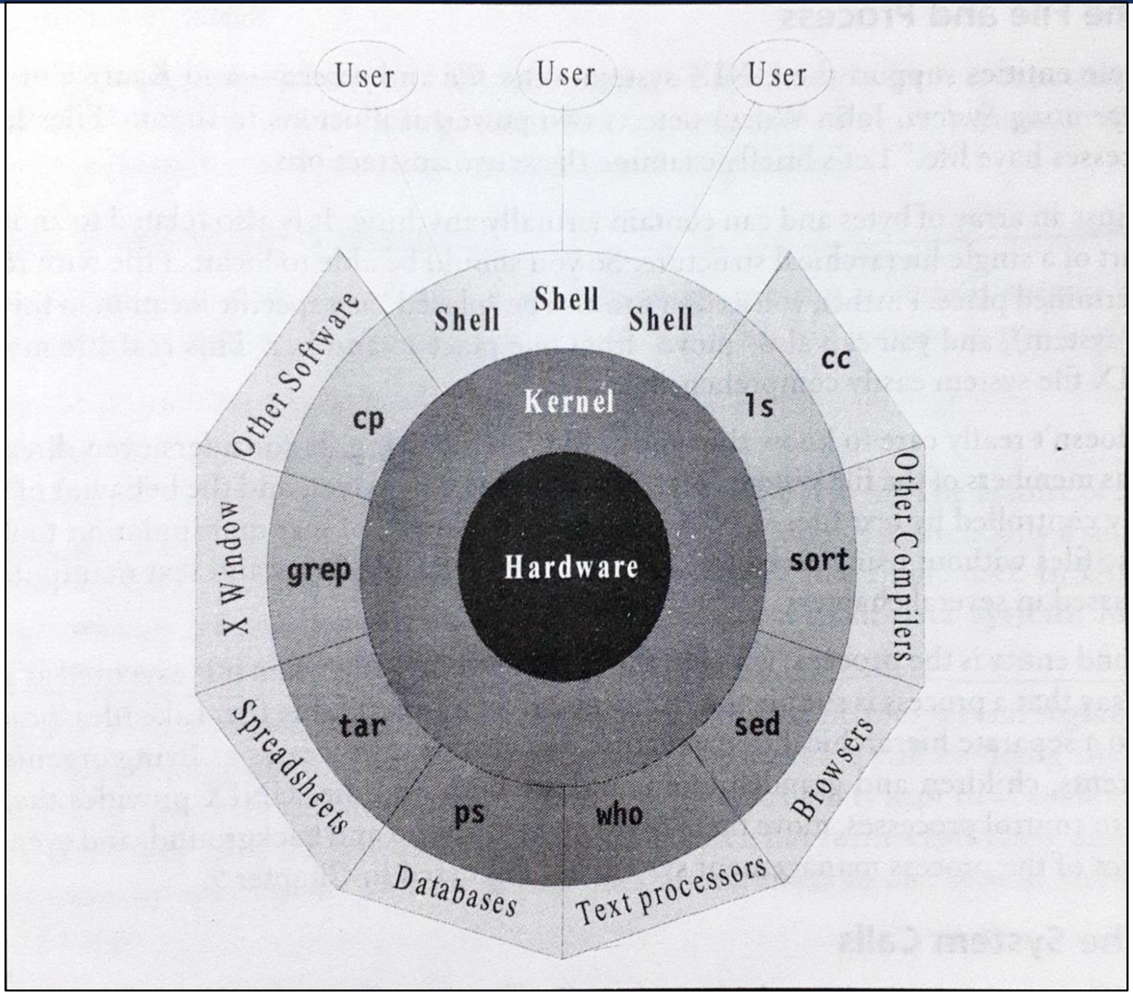
INTRODUCTION

- Operating System (OS):
  - Software that manages the computer's hardware and provides a convenient and safe environment for running programs.
  - An interface between programs and hardware resources.
  - Loaded into memory when computer is switched ON and remains active as long as the machine is running.
- Examples of OS – Microsoft Windows, MS-DOS, UNIX, Linux, macOS

INTRODUCTION

- UNIX Operating System:
  - Built earlier than MS-DOS and Microsoft Windows.
  - Developed in the 1970s at AT&T Bell Labs by Ken Thompson, Dennis Ritchie and other fellow researchers.
- Interaction with UNIX is through a command interpreter known as "*shell.*"
- Any word/character input in the shell is considered as a command.
- Power of UNIX – combining commands to perform various functions.

UNIX ARCHITECTURE



Architecture of UNIX: The Kernel-Shell Relationship

UNIX ARCHITECTURE

- Two main components – *kernel* and *shell.*
- Kernel – interacts with the machine's hardware.
- Shell – interacts with the user.

UNIX ARCHITECTURE

- Kernel:
  - Core of the operating system.
  - Collection of routines written in C.
  - Loaded onto the memory when system is booted.
  - Communicates directly with the hardware.
  - User programs accessing hardware – kernel performs the job on behalf of the user. (*system calls*)
  - Housekeeping work – memory management, process scheduling, priority of processes.

UNIX ARCHITECTURE

- Shell:
  - Outer part of the operating system.
  - Translating commands into action – command interpreter.
  - Interface between the user and the kernel.
  - "One kernel, many shells" concept.
  - Shell prompts – **$, %, #**
  - First shell command – **echo $SHELL**

UNIX ARCHITECTURE

- File and Process:
  - File – array of bytes that can contain anything.
    - Related to another file by being part of a single hierarchical structure.
    - UNIX does not care about the type of file – directories and devices are part of the file system.
    - Dominant file type – *text*.
  - Process – name given to a file when it is executed as a program.
    - Processes also belong to a hierarchical tree structure.
    - Similar to living organisms – parent, child, born, die, zombie, orphan.

UNIX ARCHITECTURE

- System Calls:
  - More than a thousand commands in UNIX – all of them use *system calls* to communicate with the kernel.
  - Major advantage of UNIX-based systems – all of them use the same system calls.
  - System calls are built into the kernel – easy portability from one UNIX machine to another UNIX machine.
  - *write* (system call) in UNIX vs. *fprintf* (standard library function) in Windows

FEATURES OF UNIX

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

FEATURES OF UNIX

- Multiuser System
  - Fundamental view – multiprogramming system
  - Multiple programs can run and compete for the attention of the CPU.
    - Multiple users – separate single jobs.
    - Single user – multiple jobs.
  - Resources are shared between all users in UNIX, unlike Windows.
  - The illusion of a multiuser system – UNIX breaks up a unit of time into multiple segments and allots these segments to all the users of the system.

FEATURES OF UNIX

- Multitasking System
  - Single user – multiple tasks concurrently.
  - Kernel is designed to handle a user's multiple needs – edit file, print file, send email, browse the Internet.
  - One job runs in the *foreground*, remaining jobs run in the *background*.
  - Switching between jobs.

FEATURES OF UNIX

- Building-Block Approach
  - UNIX consists of few hundred commands which perform simple jobs.
  - Too many features are NOT packed into a few tools.
  - Combining simple commands to perform powerful functions – *pipes* and *filters*.
  - Better to handle specialized function than try to solve multiple problems.
  - Output of one tool can act as input to another tool.

FEATURES OF UNIX

- UNIX Toolkit
  - Diverse range of tools – general-purpose tools, text manipulation utilities, compilers and interpreters, networked applications and system administration tools.
  - Choice between shells.
  - New tools added with each UNIX release, old tools modified/removed.

FEATURES OF UNIX

- Pattern Matching
  - Sophisticated pattern matching features.
  - Special characters (*metacharacters*) – `* ? [ ] ' " \ $ ; & ( ) | ^ < > new-line space tab`
  - Regular Expressions – special expressions formed from the metacharacter set.

FEATURES OF UNIX

- Programming Facility
  - Designed for programming – control structures, loops, variables.
  - Shell scripts – programs that can invoke UNIX commands.
  - Controlling and automating system functions.
  - Career opportunity – ***system administration***.

FEATURES OF UNIX

- Documentation
  - **man** command – Detailed manual; reference for commands and their configuration files.
  - Websites, articles, videos.
  - Online terminals for practice.

POSIX AND THE SINGLE UNIX SPECIFICATION

- POSIX – Portable Operating System Interface for Computer Environments
- Group of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.
- POSIX.1 and POSIX.2
  - POSIX.1 – specifies the system calls
  - POSIX.2 – deals with the shell and utilities
- Single UNIX Specification (SUS) – "*write once, adopt anywhere*" approach
- Easy portability between POSIX-compliant systems

COMMAND STRUCTURE

- Commands and arguments.
  - Example: **echo abcd**
- Separation between command and corresponding arguments – *whitespace*
- Permission to use multiple whitespaces to separate words.
- Range of arguments – options, expressions, instructions, filenames.

COMMAND STRUCTURE

- Options – special type of argument (minus sign).
  - Example: **ls –l**
- Preceded by –(minus) sign to distinguish from filenames.
- List of options is predetermined, whereas it is <u>not</u> the case for normal arguments.

COMMAND STRUCTURE

- Using a command with the wrong/undefined option.
  - Example: **ls –z note**
- Necessity of providing whitespace between command and argument.
  - Example: **ls-l**
- Using multiple options on the same line.
  - Example: **ls –l –a –t**
- Combining multiple options.
  - Example: **ls -lat**

COMMAND STRUCTURE

- Filename arguments – command takes input from the file.
- Generally last argument of a command.
- Multiple filenames as arguments.
  - Example: **ls –lat chap1 chap2 chap3**
- Command + arguments/options = command line

COMMAND STRUCTURE

- Exceptions to the command structure:
  - Commands with no arguments.
    - Example: **pwd**
  - Commands which may or may not have arguments.
    - Example: **who**
  - Commands which can run without arguments, with only options, with only filenames, or using combination of both.
    - Example: **ls**

COMMAND STRUCTURE

- Exceptions to the command structure:
  - Commands which compulsorily need to have an option.
    - Example: **cut**
  - Commands with arguments which have expressions.
    - Example: **grep**
  - Commands with arguments which have a set of instructions.
    - Example: **sed**
  - Commands with arguments which have an entire program.
    - Examples: **awk** and **perl**

LOCATING COMMANDS

- UNIX commands are case-sensitive – all commands are lowercase
  - Example: **ECHO**
- Commands – files containing programs written in C
- Storage of files – directories
- Knowing the location of a command – **type** command
  - Example: **type echo**
- **type** only looks in directories specified in PATH
- **–a** option for checking aliases

LOCATING COMMANDS

- PATH – environment variable which specifies a set of directories where programs are stored.
- **echo $PATH**
- / – root directory (forward slash)
- Delimiter used in UNIX for PATH variable – : (colon)
- Delimiter used in Windows for PATH variable – ; (semicolon)
- Essential UNIX commands stored in /bin and /usr/bin

INTERNAL AND EXTERNAL COMMANDS

- Program/file having an independent existence in the /bin (or /usr/bin) directory – *external* command.
  - Example: **man** command
- Most commands – external in nature.
- Set of built-in commands that are not stored as separate files – *internal* command.
  - Example: **echo** command

INTERNAL AND EXTERNAL COMMANDS

- Shell – special type of external command; contains its own set of internal commands.
- Command exists as both internal command of shell and external to shell (in /bin or /usr/bin) – shell will prioritize its own internal command
- Best example – **echo** command

BASIC COMMANDS

- **echo** – display line of text/string that is passed as an argument to the command.
- Often used in shell scripts to display diagnostic messages on the terminal.
- Also used to issue prompts to take user input.
- **echo** was an external command, but now integrated into the shell.

BASIC COMMANDS

- Escape sequence – generally two-character string beginning with a \ (backslash).
- Usually placed at the end of a string to act as an input prompt. (**\c** option)
- Can also use octal values of ASCII characters as escape sequences.
- Octal value must be preceded by **\0**
- Escape sequences in Bash shell – **-e** option is required.
    - Example: **echo -e "Enter your name: \c"**

BASIC COMMANDS

Escape Sequence	Significance
\a	Bell
\b	Backspace
\c	No newline (cursor in same line)
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\0*n*	ASCII character represented by octal value *n*, where *n* ≤ *0377*

BASIC COMMANDS

- **printf** – alternative to **echo** command
- Most shells use **printf** as external command, only Bash has it built-in.
- Does not automatically insert new line unless **\n** is specified.
- Uses formatted strings just like in C language.
  - Example: **printf "My current shell is %s\n" $SHELL**
- Multiple formats can be used in single **printf** – need to specify as many arguments as there are format strings, and in the <u>right order</u>.

BASIC COMMANDS

Format Specifier	Significance
%s	String
%30s	String, but printed in a space 30 characters wide
%d	Decimal integer
%6d	Decimal integer, but printed in a space 6 characters wide
%o	Octal integer
%x	Hexadecimal integer
%f	Floating point number

BASIC COMMANDS

- **cd** – change current working directory
- Change to root – **cd /**
- Change to home – **cd ~** or just **cd**
- Change to some directory – **cd dir1/dir2/dir3**
- Change to parent directory – **cd ..**
- Change to directory with spaces in its name – **cd "dir name"** or **cd dir\ name**

BASIC COMMANDS

- **ls** – list files in a directory
- Default arrangement – alphabetically with uppercase having precedence over lowercase (*ASCII collating sequence*)
- List files – **ls**
- List files with similar filenames – **ls chap***
- List files with detailed descriptions – **ls –l chap***
- **echo** can also be used to list files in a directory – **echo ***

BASIC COMMANDS

- **who** – account of all the users who are logged on to the system.
- Detailed descriptions with headers – **–Hu** option
- To know the user who is currently on the active account – **whoami**

BASIC COMMANDS

- **date** – display system date
- UNIX systems – internal clock that is running since 01 January 1970 (the Epoch)
- 32-bit counter stores these seconds

BASIC COMMANDS

Format Specifier	Significance
+%d	Date
+%m	Month number
+%h	Month name
+%y	Year
+%H	Hour
+%M	Minute
+%S	Seconds
+%D	Date in the format *mm/dd/yyyy*
+%T	Time in the format *hh:mm:ss*
+"%d %m %y %H:%M:%S"	Multiple format specifiers together (enclose within double quotes, use single **+** symbol before the quotes)

BASIC COMMANDS

- **passwd** – changing user password
- Expects 3 responses – old password, new password, re-enter new password
- Enter password – encrypted by the system
- Encrypted password stored in **/etc/shadow**
- Password framing rules

BASIC COMMANDS

- **cal** – see the calendar of any specific month or a complete year
- Default – calendar of current month
- Example to see calendar of March 2011 – **cal 03 2011**
- Example to see calendar of 2020 – **cal 2020**

COMBINING COMMANDS

- Allows more than one command in the command line.
- Separated by ; (semicolon) – shell understands the separate processing of commands
- Example: **wc chap01 ; ls –l chap01**
- Redirection of output – **(wc chap01 ; ls –l chap01) > newlist**

ROOT ACCESS

- Superuser or root user – system administrator with access to everything.
- System administration tasks – maintaining user accounts, security, managing disk space to perform backups.
- Special login name for administrator – **root**
- Comes along with every UNIX system.
- Prompt of root – **#** (hash)

ROOT ACCESS

- Acquiring superuser status – **su** command
- Prompt changes, but directory remains same.
- To be in root's home directory – **su –l**
- Superuser constantly navigates file system – PATH for superuser different than other users and does not include current directory.
- Recreation of user environment – **su – name**

UNIX FILES – INTRODUCTION

- File – container for storing information.
- UNIX files do not contain end-of-file (EOF) mark.
- All file attributes (name, size) are stored in the kernel – not accessible to humans.
- Everything is a file in UNIX!

FILE TYPES

- Main categories – 3 types of files in UNIX.
- Ordinary file – contains only data as a stream of characters (*regular* file).
- Directory file – contains names of files and other directories, and a number associated with each name.
- Device file – all devices and peripherals are represented by files.

FILE TYPES

- Ordinary file –
  - Text file – contains only printable characters.
    - Examples: C, Java programs, shell, perl scripts.
    - Contains characters where every line is terminated with the *newline* character (*line feed*).
  - Binary file – contains both printable and unprintable characters from full ASCII range.
    - Examples: UNIX commands, object code and executables, picture, sound, video files.
    - Usually cannot be displayed properly.

FILE TYPES

- Directory file –
  - Contains no data, but keeps details of files and subdirectories.
  - Contains an entry for every file and subdirectory in that directory.
  - Each entry has two components – *filename* and unique identification number (*inode*)
  - ***A directory contains the filename and not the file's contents.***
  - Cannot write a directory, but actions like creating or removing files updates the contents.

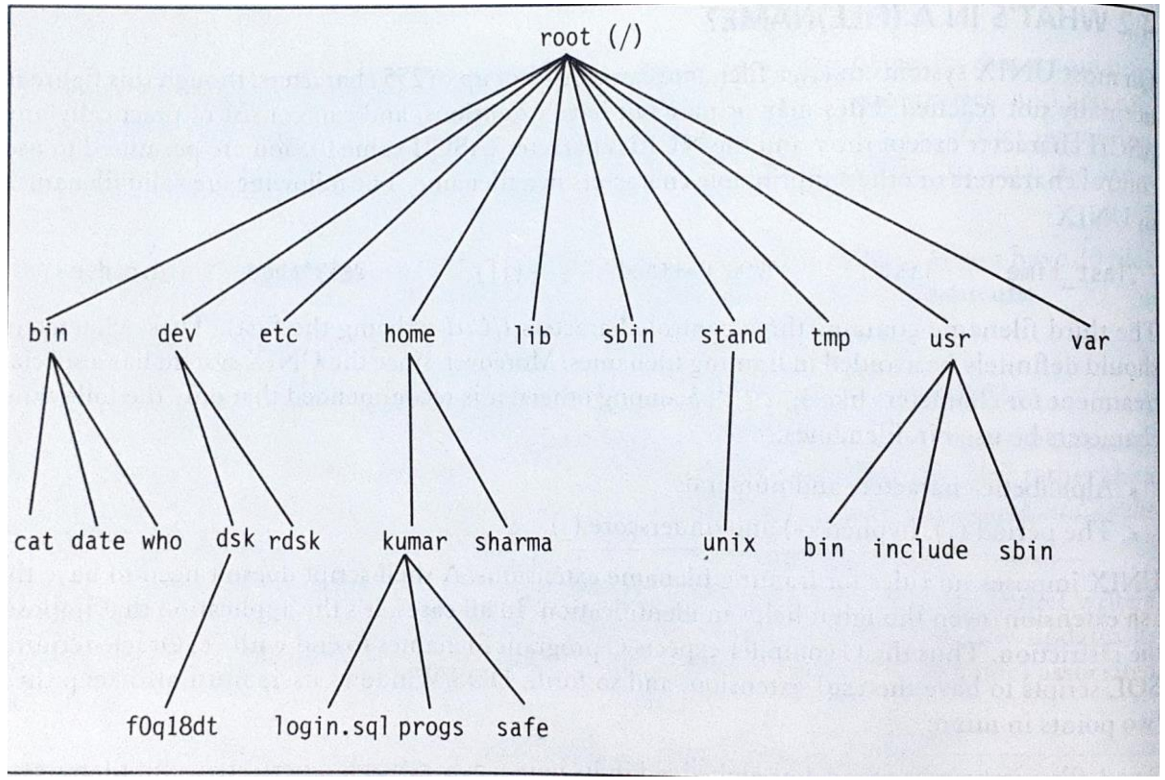
FILE TYPES

- Device file –
  - Does not contain anything at all.
  - Operation of device file governed by attributes of its associated file.
  - Kernel identifies a device from its attributes and uses them to operate the device.
    - Example: copying files to pendrive or micro-SD card.
  - Device filenames usually located at **/dev**

NAMING FILES

- Filename can consist of up to 255 characters and is case-sensitive.
- Names may or may not have extensions and can contain any character except / and **NULL** character (ASCII value 0).
- Permitted to use control characters or other unprintable characters.
- Recommended characters for filenames –
  - Alphabetic characters and numerals.
  - **.** (Dot), **-** (hyphen) and _ (underscore). (hyphen not recommended at the beginning!)

ORGANIZATION OF FILES



UNIX File System Tree:
The Parent-Child Relationship

ORGANIZATION OF FILES

- All files in UNIX are related to one another.
- UNIX file system – collection of all files in a hierarchical (*inverted tree*) structure.
- Topmost file – *root* directory file (represented by */*)
- <u>Not</u> the same as root user
- Root – subdirectories – more subdirectories, and so on.
- Every file has a parent (apart from *root*).

STANDARD DIRECTORIES

Directory	Significance
/bin	Programs needed for using and managing the system (*binaries*)
/dev	System device files – interface to a particular device
/etc	System-specific configuration files and files essential for start-up
/home	Home directories for all users of the system
/mnt	Temporary file systems are mounted
/opt	Software files that are not installed when the OS is installed (products by third-party vendors)
/sbin	Programs for system administration (*system binaries*)
/tmp	Holding temporary files (*scratch directory*)
/usr	Programs and data related to users of a system (read-only and can be shared on a network)
/var	Files with varying content (log files, mail system files, print spooling system files)

HIDDEN FILES

- Files that are not usually displayed in a directory listing.
- Filenames begin with a **.** (dot)
  - Example: **.abcde**
- Show up when full listing is used, such as **ls –a**
- Concept came into existence to store configuration and informational text.
- Also known as dotfiles.

HOME DIRECTORY AND HOME VARIABLE

- Default directory upon login – *home* directory.
- Created by the system when a user account is opened.
- Login name gives you a directory under the *home* directory
  - Example: **/home/abcde**
- Shell variable **HOME** knows the location of the *home* directory.
  - **echo $HOME**
- ~ (tilde) symbol followed by / (forward slash) – used to refer to the *home* directory.

PATH VARIABLE

- **PATH** variable – environment variable which specifies a set of directories where programs are stored.
- **echo $PATH**
- Manipulating the **PATH** –
  - by changing the value of PATH to include a directory.
  - by using a pathname in the command line.

ABSOLUTE AND RELATIVE PATHNAMES

- Absolute pathname –
  - location of a file with respect to the root /
  - *always starts with /*
  - Example: **/home/abcde/sample.txt**
- Relative pathname –
  - location of a file with respect to the current working directory.
  - *never starts with /*
  - Example (if current working directory is */home*): **abcde/sample.txt**

. (DOT) AND .. (DOUBLE DOT)

- . (dot) – current directory.
  - Example: **cat ./Downloads/sample.txt**
  - **cd Downloads** same as **cd ./Downloads**
- .. (double dot) – parent directory of the current directory.
  - Example: **cat ../abcde/Downloads/sample.txt**
  - **cd ..** moves up one level

DIRECTORY COMMANDS

- **pwd** – print working directory
- Gives the absolute path of the current directory where the user is located.
- Example: **pwd**

DIRECTORY COMMANDS

- **cd** – change current working directory
- No need for the absolute path when specifying the argument.
- Change to root – **cd /**
- Change to home – **cd ~** or just **cd**
- Change to parent directory – **cd ..**
- Change to some directory – **cd dir1/dir2**

DIRECTORY COMMANDS

- **mkdir** – create a new directory
- Create one directory – **mkdir abcde**
- Create multiple directories – **mkdir abcde fghij klmno**
- Create directory trees – **mkdir abcde abcde/dir1 abcde/dir2**
- <u>Order is extremely important</u> when creating directory trees.
- Failed directory creations – directory already exists, ordinary file exists with same name or insufficient permissions to create directory.

DIRECTORY COMMANDS

- **rmdir** – remove a directory
- Directory to be removed must be empty.
- Remove one directory – **rmdir abcde**
- Remove multiple directories – **rmdir abcde fghij klmno**
- Remove directory trees – **rmdir abcde/dir1 abcde/dir2 abcde**
- Reverse order is extremely important when removing directory trees.
- Failed directory removals – directory not empty, not present in the parent directory of the directory to be removed or insufficient permissions to remove directory.

FILE-RELATED COMMANDS

- **cat** – display the contents of a file on the terminal.
- Accepts more than one filename as argument – concatenation operation.
- Additional options – **–v** (display non-printable characters) and **–n** (numbering lines)
- Creating a file – **cat > filename** (Ctrl-d to end input)
- Print in reverse order – **tac filename**
- Versatile command – create, display, concatenate and append to files.

FILE-RELATED COMMANDS

- **cp** – copy a file or group of files.
- Requires at least two filenames to be specified in the command line.
  - Example: **cp file1 file2**
- Destination file does not exist – created before copying contents.
- Destination file exists – overwritten without warning.
- Only one file to be copied – destination can be ordinary file or directory.
- Often used with current directory – **cp /home/abcde/file.txt .**

FILE-RELATED COMMANDS

- Multiple files to be copied – last filename must be a directory and that directory must exist.
  - Example: **cp chap1 chap2 chap3 newdir** or **cp chap* newdir**
- Interactive copying – **-i** option warns the user before overwriting.
- Recursive copying – **-R** option copies an entire directory structure and subdirectories.
- Not possible to copy a file – read-protected file or destination is write-protected.

FILE-RELATED COMMANDS

- **mv** – move files.
- Two distinct functions – rename file and move files to a different directory.
- Example – **mv chap2 book2**
- Destination file does not exist – created before moving contents.
- Destination file exists – overwritten without warning.

FILE-RELATED COMMANDS

- Multiple files to be moved – last filename must be a directory and that directory must exist.
  - Example: **mv chap2 chap3 newdir**
- Interactive moving – **-i** option warns the user before overwriting.
- Not possible to move a file – read-protected file or destination is write-protected.

FILE-RELATED COMMANDS

- **rm** – remove/delete one or more files.
- Operates silently and should be used with caution. File once deleted cannot be recovered.
- Example – **rm chap1 chap2**
- **rm** won't *normally* remove a directory but can remove within a directory.

FILE-RELATED COMMANDS

- Remove all files in a directory – **rm ***
- Interactive deletion – **-i** option warns the user before deleting.
- Recursive deletion – **-r** or **-R** option deletes files *and* directories recursively.
- Thorough recursive search before deletion and then deletes everything.
- Forcing removal – **-f** option overrides any write-protected files and deletes them.
- Most dangerous command – **rm -rf ***

FILE-RELATED COMMANDS

- **wc** – counts lines, words and characters.
- One or more filenames as arguments and displays four-columnar output.
- Four columns – number of lines, words, characters and filename.
  - Example – **wc file3**
- Three options – **-l** (lines), **-w** (words), **-c** (characters)
- Multiple filenames as arguments – multiple outputs + total count

FILE-RELATED COMMANDS

- **od** – displays the octal dump of the specified data.
- Requires an option and filename to display readable output.
- Display octal values of each character separately – **od -b file2**
- Display the characters and their octal values – **od -bc file2**
- Octal representations in first line, printable characters and escape sequences in second line.

THANK YOU