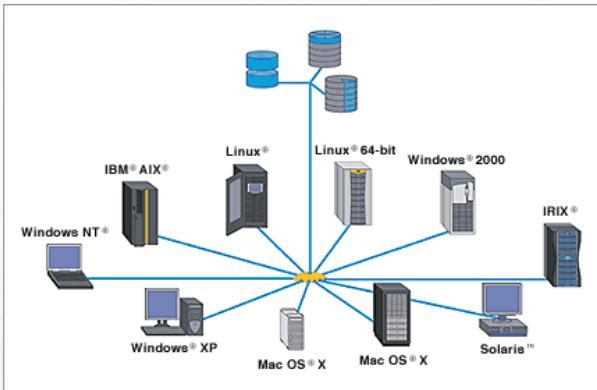


# Unix Training



High performance. Delivered.

## Goals

- Unix Architecture
- System Bootup
- Login Management
- Directories and Files
- Unix File Management
- Standard Unix Streams
- Unix Directories

## Goals

- File Permissions
- Unix Environment Commands
- Java Basic Utilities
- Unix Pipes and Filters
- Grep Command
- Unix Process Management
- Ping, FTP utility

## Goals

- Vi Editor
- Shell Commands
- Shell Scripting

# What is Unix

---

- UNIX once referred to a specific operating system.
- However, today it is not a single operating system, but rather a large family of closely related operating systems.
- These different operating systems are sometimes known as UNIX variants, or UNIX-like operating systems.
- All these operating systems are built using a collection of enabling technologies that were originally developed in the 1970s at AT&T Bell Laboratories and at the University of California, Berkeley. They have much in common and share a set of utilities and programs.

# What is Unix

---

- The Unix system is a multi-user, multi tasking operating system which means that it allows a single or multiprocessor computer to simultaneously execute several programs by one or several users.
- It has one or several command interpreters (shell) as well as a great number of commands and many utilities (assembler, compilers for many languages, text processing, email, etc.).

# What is Unix

---

- Furthermore, it is highly portable, which means that it is possible to implement a Unix system on almost all hardware platforms.
- Currently, Unix systems have a strong foothold in professional and university environments thanks to their stability, their increased level of security and observance of standards, notably in terms of networks.

# Why Is UNIX Important

---

- During the past 35 years, the operating system known as UNIX has evolved into a powerful, flexible, and versatile operating system.
- The different variants of UNIX conform to a variety of standards and are closely related.
- To understand how to use any or all of them, you need to only understand the basic conceptual model upon which UNIX is built.
- Once this conceptual model is understood, it is straightforward to learn the peculiarities of a variant of UNIX or to learn how to use a new variant of UNIX if you already know how to use another.

# Why Is UNIX Important

---

- UNIX, as it is implemented in its many variants, serves as the operating system for all types of computers, including personal computers and engineering workstations, multiuser microcomputers, minicomputers, mainframes, and supercomputers, as well as special-purpose devices.
- The number of computers running a variant of UNIX has grown explosively with more than 40 million computers now running a variant of UNIX and more than 300 million people using these systems.
- This rapid growth, especially for computers running Linux, is expected to continue, according to most computer industry experts.
- The success of UNIX is due to many factors, including its portability to a wide range of machines, its adaptability and simplicity, the wide range of tasks that it can perform, its multiuser and multitasking nature, and its suitability for networking, which has become increasingly important as the Internet has blossomed.

# Why Is UNIX Important

---

- Open Source Code
- Cooperative Tools and Utilities
- Multiuser and Multitasking Abilities
- Excellent Networking Environment(excellent platform for web servers)
- Portability(**people using the desktop environment of Mac OS X without knowing that it is built on UNIX**)
  - less work is needed to adapt it to run on a new hardware platform.

# Why Is UNIX Important

---

- Security
- Background Processing (Many jobs/tasks are executed in bg without human intervention).
- Pipes (chain od commands)
- Redirection tools
- Software Development Tools (any language which has interpreter and compiler).
- Communication

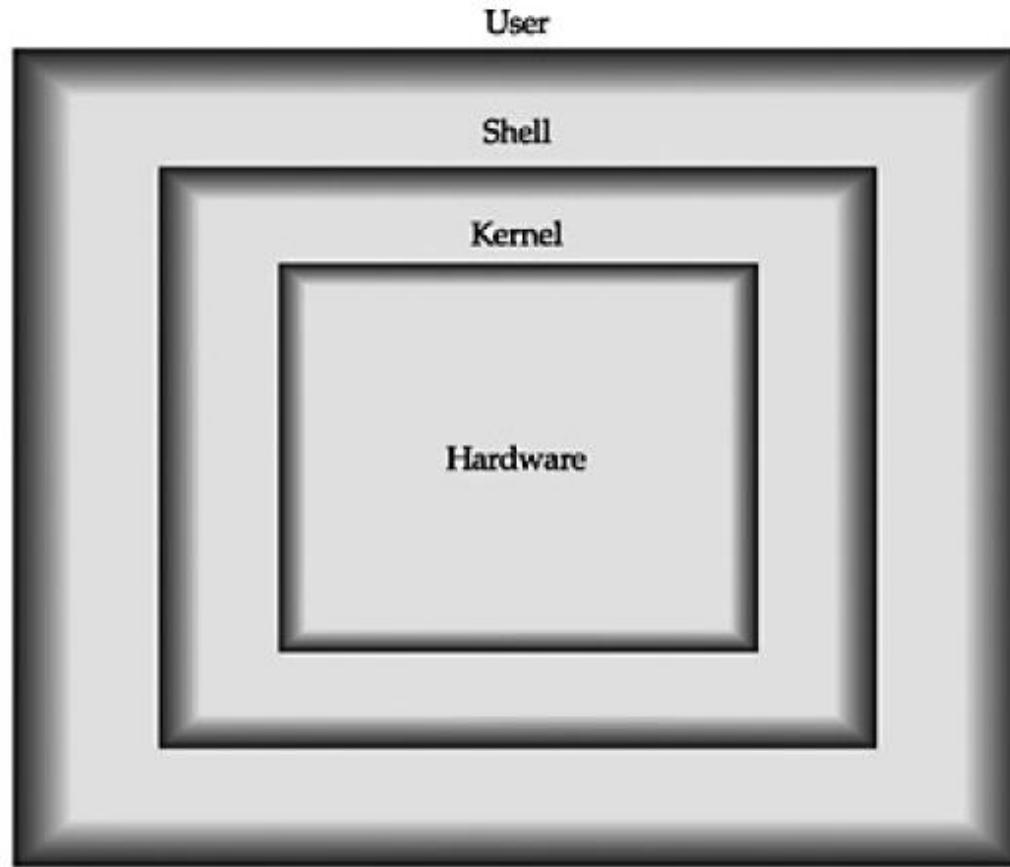


# Why Is UNIX Important

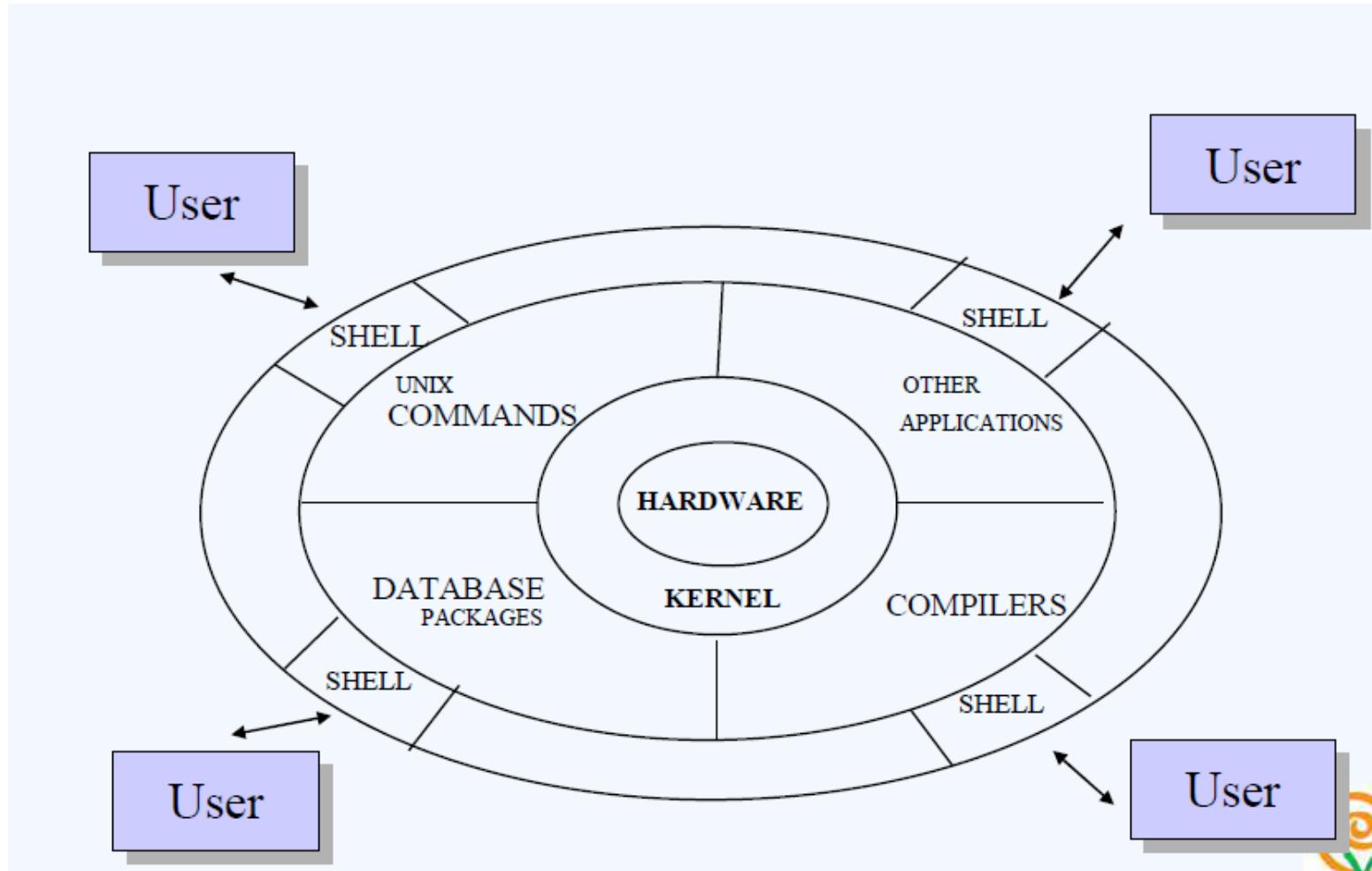
---

- Hierarchical File System
- Shells

# The Structure of the UNIX Operating System



# The Structure of the UNIX Operating System



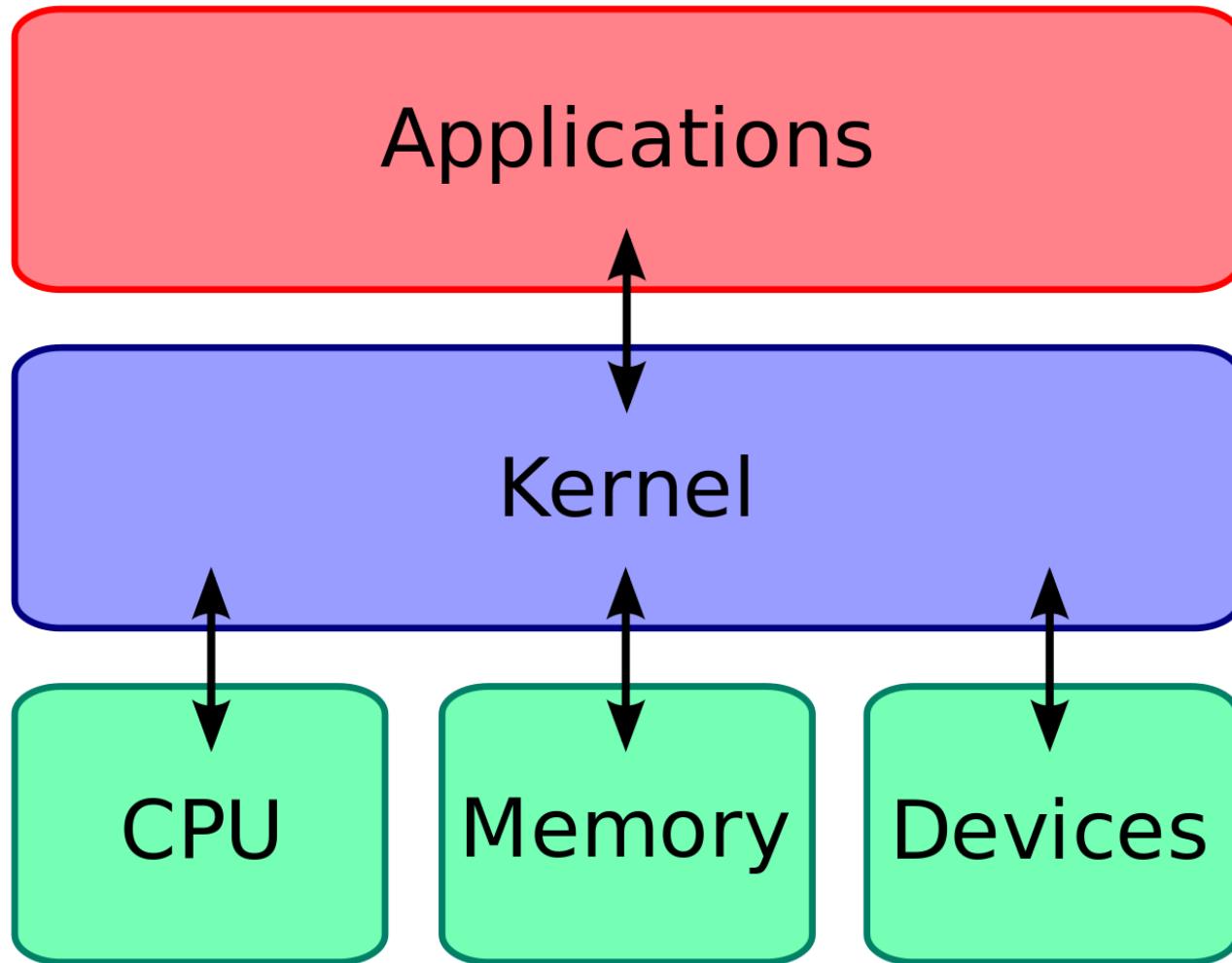
# The Structure of the UNIX Operating System



- The UNIX operating system is made up of several major components.
- These components include the kernel, the shell, the file system, and the commands (or user programs ).

# Kernel

---



# Kernel

---

- The kernel is the part of the operating system that interacts directly with the hardware of a computer, through device drivers that are built into the kernel.
- It provides sets of services that can be used by programs, insulating these programs from the underlying hardware.
- The major functions of the kernel are to
  - manage computer memory
  - control access to the computer
  - maintain the file system

# Kernel

---

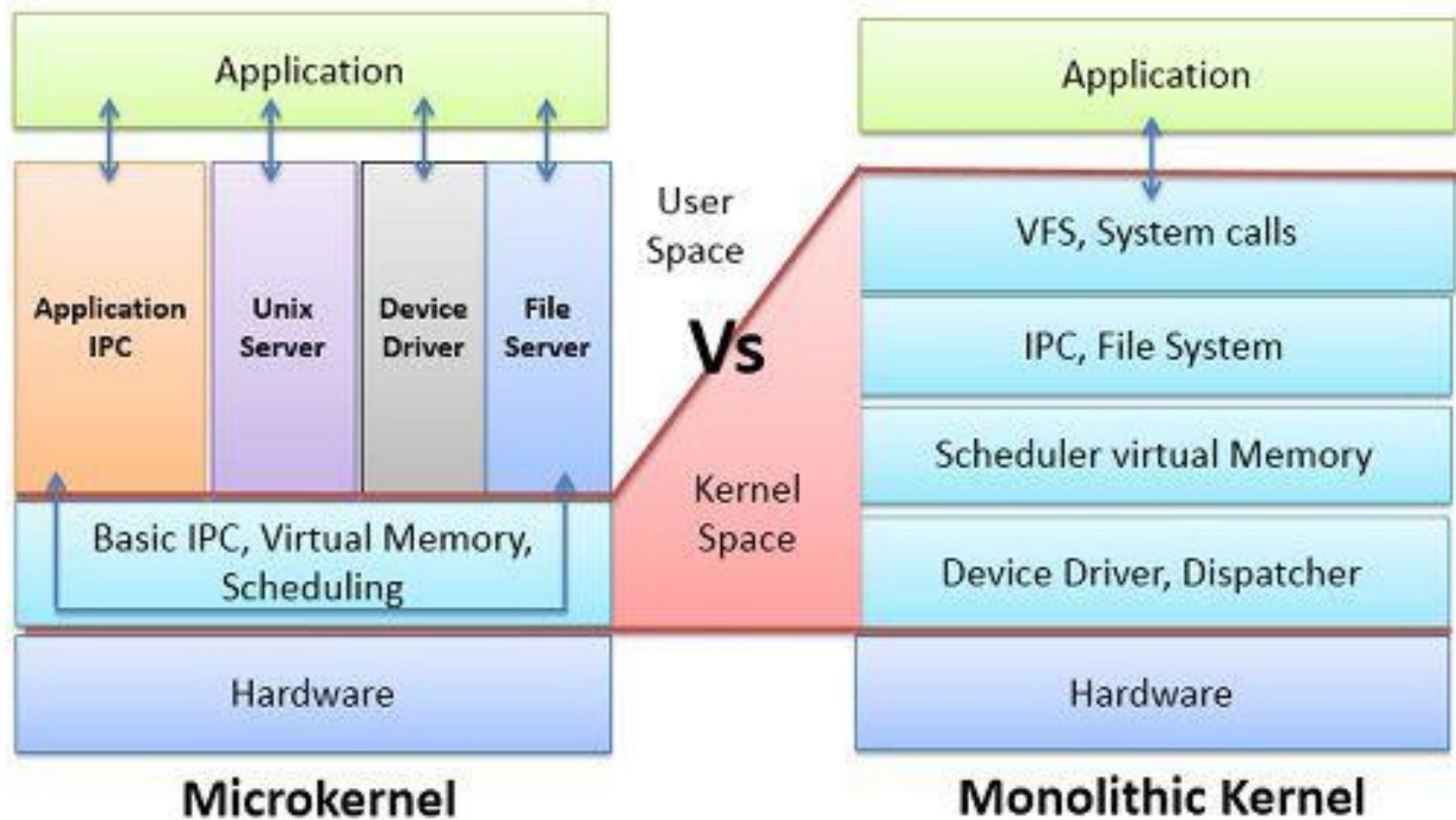
- handle interrupts (signals to terminate execution)
- handle errors
- perform input and output services (which allow computers to interact with terminals, storage devices, and printers)
- allocate the resources of the computer (such as the CPU or input/output devices) among users.

# Kernel

---

- Programs interact with the kernel through system calls.
- System calls tell the kernel to carry out various tasks for the program, such as
- Opening a file, writing to a file, obtaining information about a file, executing a program, terminating a process, changing the priority of a process.
- Getting the time of day Different implementations of a variant of the UNIX system may have compatible system calls, with each call having the same functionality.
- However, the internals, programs that perform the functions of system calls (usually written in the C language), and the system architecture in two different UNIX variants or even two different implementations of a particular UNIX variant may bear little resemblance to one another.

# Kernel



**Microkernel**

**Monolithic Kernel**

BASIS FOR COMPARISON	MICROKERNEL	MONOLITHIC KERNEL
Basic	In microkernel user services and kernel, services are kept in separate address space.	In monolithic kernel, both user services and kernel services are kept in the same address space.
Size	Microkernel are smaller in size.	Monolithic kernel is larger than microkernel.
Execution	Slow execution.	Fast execution.
Extendible	The microkernel is easily extendible.	The monolithic kernel is hard to extend.
Security	If a service crashes, it does effect on working of microkernel.	If a service crashes, the whole system crashes in monolithic kernel.
Code	To write a microkernel, more code is required.	To write a monolithic kernel, less code is required.
Example	QNX, Symbian, L4Linux, Singularity, K42, Mac OS X, Integrity, PikeOS, HURD, Minix, and Coyotos.	Linux, BSDs (FreeBSD, OpenBSD, NetBSD), Microsoft Windows (95,98,Me), Solaris, OS-9, AIX, HP-UX, DOS, OpenVMS, XTS-400 etc.

# Utilities

---

- The UNIX System contains several hundred utilities or user programs.
- Commands are also known as tools, because they can be used separately or put together in various ways to carry out useful tasks.
- You execute these utilities by invoking them by name through the shell; this is why they are called commands.
- A critical difference between UNIX and earlier operating systems is the ease with which new programs can be installed-the shell need only be told where to look for commands, and this is user-definable

# Utilities

---

- You can perform many tasks using the standard utilities supplied with UNIX.
- There are utilities for text editing and processing, for managing information, for electronic communications and networking, for performing calculations, for developing computer programs, for system administration, and for many other purposes.

# The File System

---

- The basic unit used to organize information in UNIX is called a file.
- The UNIX file system provides a logical method for organizing, storing, retrieving, manipulating, and managing information.
- Files are organized into a hierarchical file system, with files grouped together into directories.
- An important simplifying feature of UNIX is the general way it treats files.
- For example, physical devices are treated as files; this permits the same commands to work for ordinary files and for physical devices; for instance, printing a file (on a printer) is treated similarly to displaying it on the terminal screen.



# The File System

```
eswaribala@DESKTOP-55AGI0I:~$ ls /
bin  boot  dev  etc  home  init  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var
eswaribala@DESKTOP-55AGI0I:~$ .
```

## The Shell

---

- The shell reads your commands and interprets them as requests to execute a program or programs, which it then arranges to have carried out.
- Because the shell plays this role, it is called a command interpreter.
- Besides being a command interpreter, the shell is also a programming language.
- As a programming language, it permits you to control how and when commands are carried out

# Applications

---

- You can use applications built using UNIX commands, tools, and programs. Application programs carry out many different types of tasks.
- Some perform general functions that can be used by a variety of users in government, industry, and education.
- These are known as horizontal applications and include such programs as word processors, compilers, database management systems, spreadsheets, statistical analysis programs, and communications programs.
- Others are industry specific and are known as vertical applications.
- Examples include software packages used for managing a hotel, running a bank, and operating point-of-sale terminals.

# Applications

---

- Several classes of applications have experienced explosive growth in the past few years.
- The first of these involves network applications, including those that let people make use of the wide range of services available on the Internet.
- Chief among these are web browsers and web server applications.
- Another important class of applications deals with multimedia.
- Such applications let users create and view multimedia files, including audio, images, and video.

# The Birth of the UNIX System

---

- The history of the UNIX System dates back to the late 1960s when MIT, AT&T Bell Labs, and then computer manufacturer GE (General Electric) worked on an experimental operating system called Multics.
- Multics, from Multiplexed Information and Computing System, was designed to be an interactive operating system for the GE 645 mainframe computer.
- Allowed information sharing while providing security  
Development met with many delays, and production versions turned out to be slow and required extensive memory
- For a variety of reasons, Bell Labs dropped out of the project.
- However, the Multics system implemented many innovative features and produced an excellent computing environment.

# The Birth of the UNIX System

---

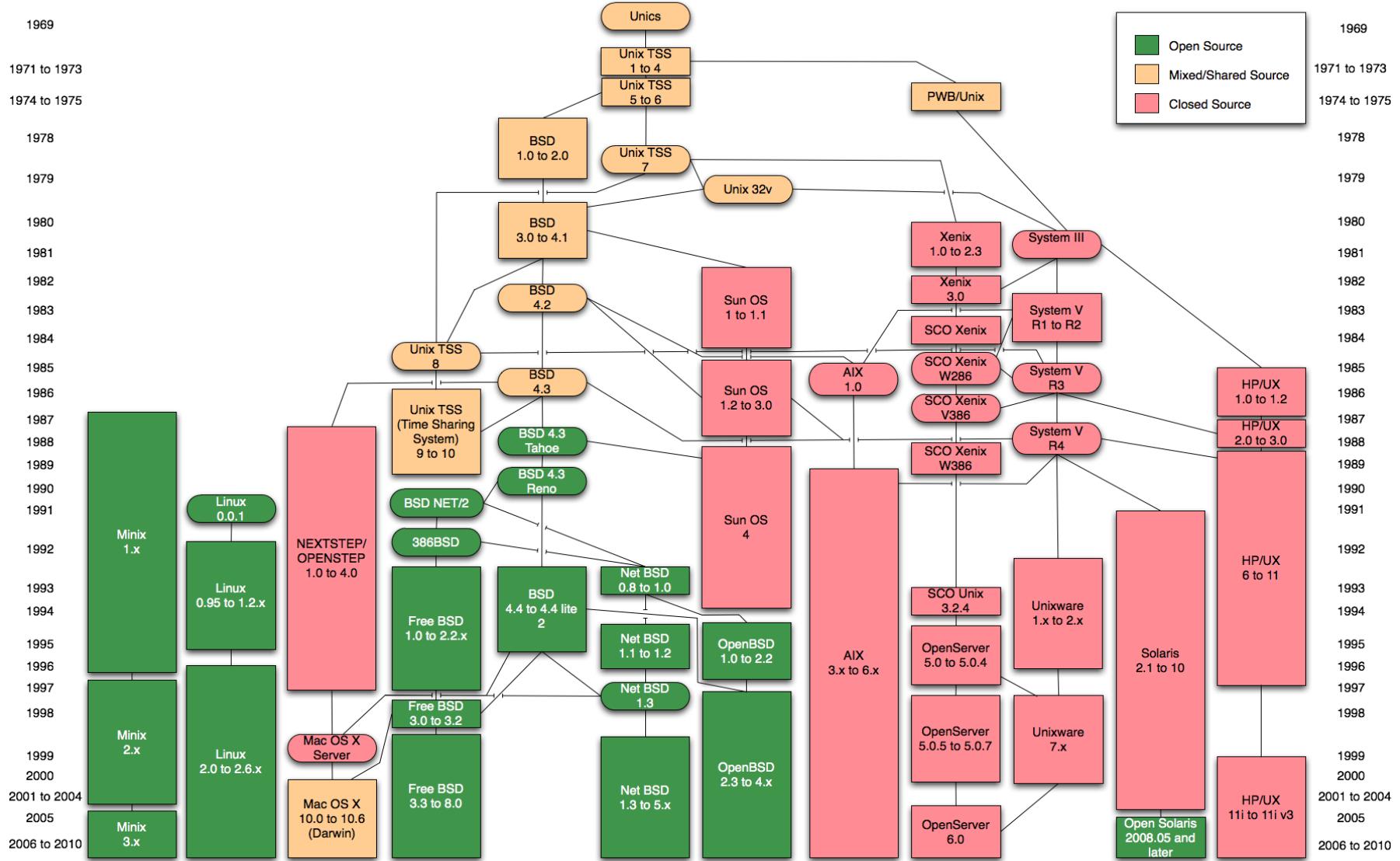
- In 1969, Ken Thompson, one of the Bell Labs researchers involved in the Multics project, wrote a game for the GE computer called Space Travel.
- This game simulated the solar system and a space ship.
- Thompson found that the game ran jerkily on the GE machine and was costly—approximately \$75 per run! With help from Dennis Ritchie, Thompson rewrote the game to run on a spare DEC PDP-7.
- This initial experience gave him the opportunity to write a new operating system on the PDP-7, using the structure of a file system Thompson, Ritchie, and Rudd Canaday had designed.
- Thompson, Ritchie, and their colleagues created a multitasking operating system, including a file system, a command interpreter, and some utilities for the PDP-7.

# The Birth of the UNIX System

---

- Because the new multitasking operating system for the PDP-7 could support two simultaneous users, it was humorously called UNICS for the Uniplexed Information and Computing System; the first use of this name is attributed to Brian Kernighan.
- The name was changed slightly to UNIX in 1970, and that has stuck ever since.

# The Birth of the UNIX System





# A UNIX System Timeline

Year	UNIX Variant or Standard	Comments
1969	UNICS (later called UNIX)	A new operating system invented by Ken Thompson and Dennis Ritchie for the PDP-7
1973	Fourth Edition	Written in C programming language; widely used inside Bell Laboratories
1975	Sixth Edition	First version widely available outside of Bell Labs; more than 600 machines ran it
1978	3BSD	Virtual memory
1979	Seventh Edition	Included the Bourne shell, UUCP, and C; the direct ancestor of modern UNIX
1980	Xenix	Introduced by Microsoft
1980	4BSD	Introduced by UC Berkeley
1982	System III	First public release outside of Bell Labs
1983	System V Release 1	First supported release

# A UNIX System Timeline

1983	4.1BSD	UC Berkeley release with performance enhancements
1984	4.2BSD	UC Berkeley release with many networking capabilities
1984	System V Release 2	Protection and locking of files, enhanced system administration, and job control features added
1986	HP-UX	First version of HP-UX released for HP Precision Architecture
1986	AIX Version 1	First version of IBM's proprietary version of UNIX, based on SVR3
1987	System V Release 3	STREAMS, RFS, TLI added
1987	4.3BSD	Minor enhancements to 4.2BSD
1988	POSIX	POSIX.1 published
1989	System V Release 4	Unified System V, BSD, and Xenix
1990	XPG3	X/Open specification set
1990	OSF/1	Open Software Foundation release designed to compete with SVR4
1991	386BSD	Based on BSD for Intel 80386



# A UNIX System Timeline

1991	Linux 0.01	Linus Torvalds started development of Linux
1992	SVR4.2	USL-developed version of SVR4 for the desktop
1992	HP-UX 9.0	Supported workstations, including a GUI
1993	Solaris 2.3	POSIX compliant
1993	4.4BSD	Final Berkeley release
1993	FreeBSD 1.0	Initial release based on 4.3BSD and 386BSD
1993	SVR4.2MP	Last version of UNIX developed by USL
1994	Linux 1.0	First version of Linux not considered a “beta”
1994	NetBSD 1.0	First multiplatform release
1994	Solaris 2.4	Motif supported
1994	AIX4	Introduced CDE support
1994	FreeBSD 2.0	Based on 4.4BSD-Lite to allow free distribution
1995	UNIX 95	X/Open mark for systems registered under the Single UNIX Specification
1995	Solaris 2.5	CDE supported
1995	HP-UX 10.0	Conformed to the Single UNIX Specification and the Common Desktop Environment (CDE)

# A UNIX System Timeline

1996	Linux 2.0	Performance improvements and networking software added
1996	OpenBSD 1.2	Initial release with strong support of security
1997	Solaris 2.6	UNIX 95 compliant, JAVA supported
1997	Single UNIX Specification, Version 2	Open Group specification set
1997	System V Release 5	Enhanced SV kernel, including 64-bit support, increased reliability, and performance enhancements
1997	UnixWare 7	SCO UNIX based on SVR5 kernel
1997	HP-UX 11.0	64-bit operating system
1997	AIX 4.3	Support for 64-bit architectures, registered with UNIX 98 mark
1998	UNIX 98	Open Group mark for systems registered under the Single UNIX Specification, Version 2
1998	FreeBSD 3.0	Kernel changes and security fixes
1998	Solaris 7	Support for 64-bit applications, free for noncommercial users
1999	Linux 2.2	Device drivers added
1999	Darwin	Apple developed UNIX-like OS, basis for Mac OS X
2000	Solaris 8	Performance and application support enhancements

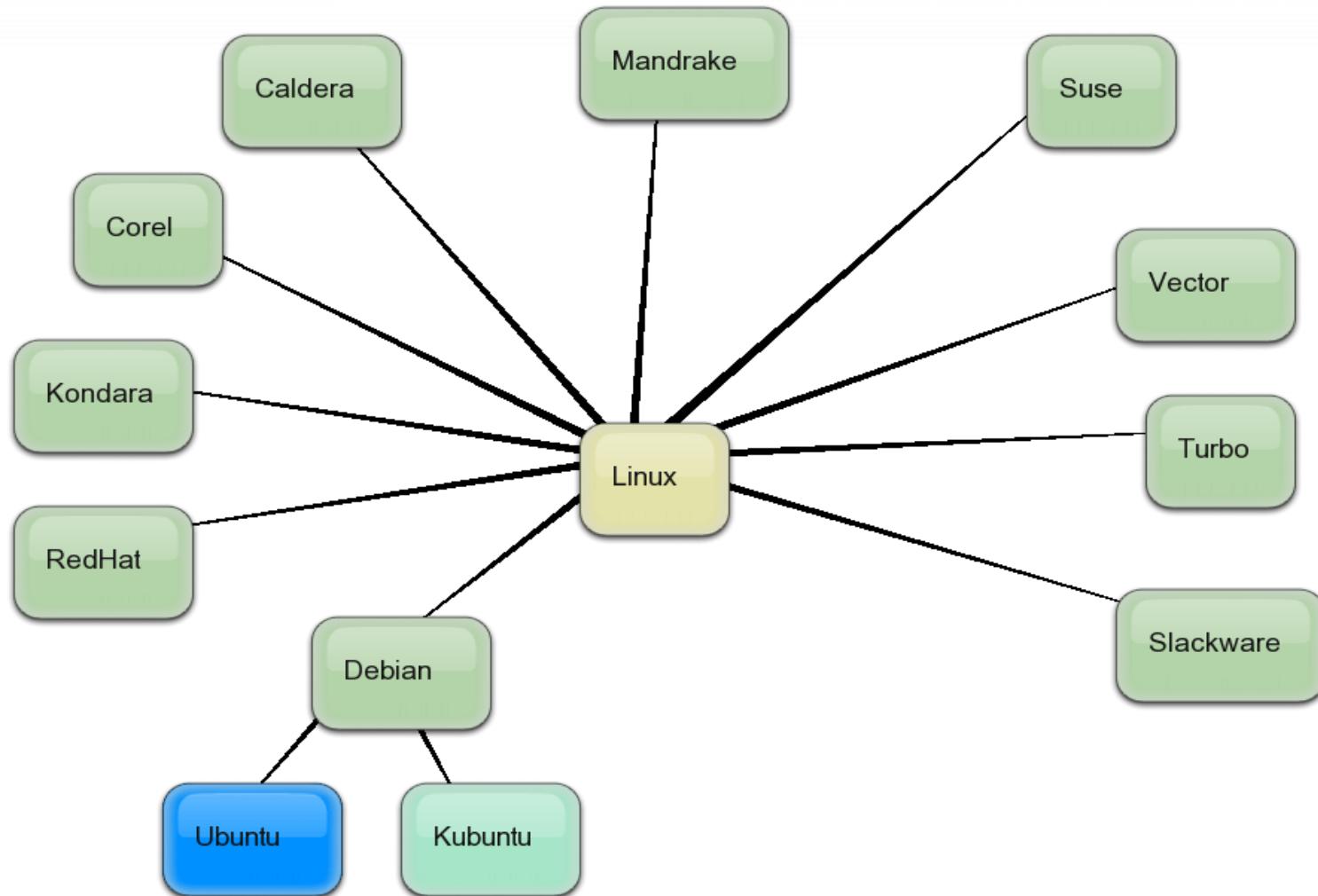
# A UNIX System Timeline

2000	HP-UX 11i	Introduces operating environments
2000	FreeBSD 4.0	Networking and security enhancements
2001	Linux 2.4	Enhanced device support, scalability enhancements
2001	AIX 5L	Introduced affinity for Linux
2001	Mac OS X 10.0 “Cheetah”	First Mac OS release based on Darwin. Incomplete and slow, but with basic OS features, device support, and software development environment
2001	Mac OS X 10.1 “Puma”	More complete than Cheetah, with performance enhancements and support for additional device drivers
2002	Solaris 9	Manageability, security, and performance enhancements
2002	Mac OS X 10.2 “Jaguar”	First solid release of Mac OS X
2003	Linux 2.6	Scalability for operation on embedded systems to large servers,

# A UNIX System Timeline

		human interface, networking, and security enhancements
2003	Mac OS X 10.3 “Panther”	Performance enhancements, an extensive update to the user interface, and greater interoperability with MS Windows
2003	Single UNIX Specification, Version 3	Developed by the Austin Group
2003	FreeBSD 5.0	Improved SMP support, TrustedBSD security features
2004	Solaris 10	Advanced security, performance, and availability enhancements
2004	NetBSD 2.0	Support for SMP
2005	OpenServer 6	Improved SMP support and support for extremely large files
2005	Mac X 10.4 “Tiger”	New features include Spotlight, a fast content and metadata-based file search tool, and support for 64-bit platforms and Intel x86 platforms
2005	Net BSD 3.0	Suppose Xen Virtual Machine Monitor

# Widely Used UNIX Variants



# UNIX Contributors

Aho, Alfred	Coauthor of the AWK programming language and author of egrep
Bourne, Steven	Author of the Bourne shell, the ancestor of the standard shell in UNIX System V
Canaday, Rudd	Developer of the UNIX System file system, along with Dennis Ritchie and Ken Thompson
Cherry, Lorinda	Author of the Writer's Workbench (WWB), coauthor of the eqn preprocessor, and coauthor of the bc and dc utilities
Honeyman, Peter	Developer of HoneyDanBer UUCP at Bell Laboratories in 1983 with David Nowitz and Brian Redman
Horton, Mark	Author of curses and terminfo, and a major contributor to the UUCP Mapping Project and the development of USENET
Joy, William	Creator of the vi editor and the C shell, as well as many BSD enhancements. Cofounder of Sun Microsystems
Kernighan, Brian	Coauthor of the C programming language and of the AWK programming language. Rewrote troff in the C language
Korn, David	Author of the Korn shell, a superset of the standard System V shell with many enhanced features, including command histories
Lesk, Mike	Developer of the UUCP System at Bell Laboratories in 1976 and author of the tbl preprocessor, ms macros, and lex

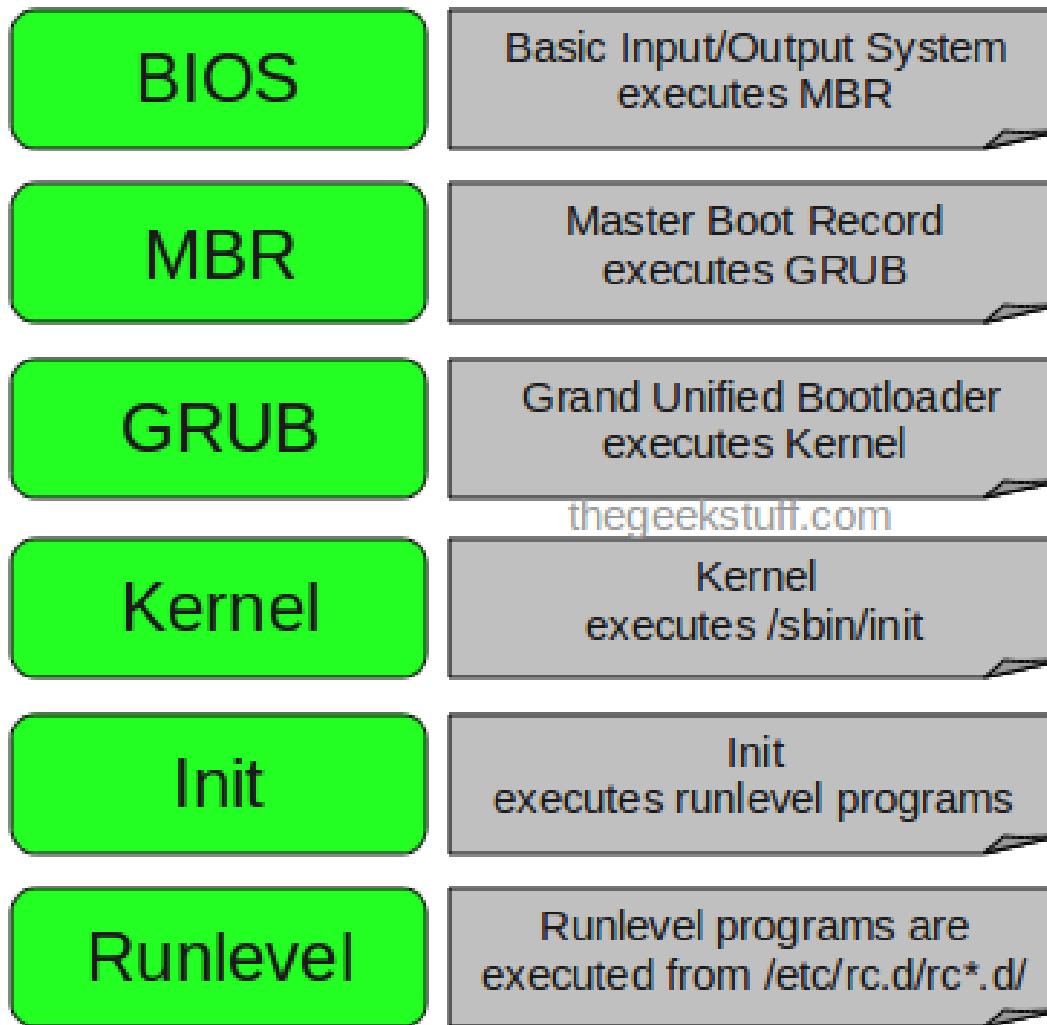
# UNIX Contributors

Mashey, John	Author of the early versions of the shell, which were later merged into the Bourne shell
McIlroy, Doug	Developed the concept of pipes and wrote the spell and diff commands
Morris, Robert	Coauthor of the utilities bc and dc
Nowitz, David	Developer of HoneyDanBer UUCP at Bell Laboratories in 1983 with Peter Honeyman and Brian Redman
Ossanna, Joseph	Creator of the troff text formatting processor
Ousterhout, John	Developer of Tcl command language
Redman, Brian	Developer of HoneyDanBer UUCP at Bell Laboratories in 1983 with Peter Honeyman and David Nowitz
Ritchie, Dennis	Inventor of the UNIX Operating System, along with Ken Thompson, at Bell Laboratories. Inventor of the C language, along with Brian Kernighan
Scheifler, Robert	Mentor of the X Window System
Stallman, Richard	Developer of the programmable visual text editor emacs, and founder of GNU project and the Free Software Foundation
Stroustrup, Bjarne	Developer of the object-oriented C++ programming language
Tannenbaum,	Creator of Minix, a program environment that led to the development of Linux

# UNIX Contributors

Andrew	
Thompson, Ken	Inventor of the UNIX operating system, along with Dennis Ritchie, at Bell Laboratories
Torek, Chris	Developer from the University of Maryland who was one of the pioneers of BSD UNIX
Torvalds, Linus	Creator of the Linux operating system, an Intel personal computer-based variant of UNIX
Wall, Larry	Developer of the Perl programming language
Weinberger, Peter	Coauthor of the AWK programming language

# Boot up Process

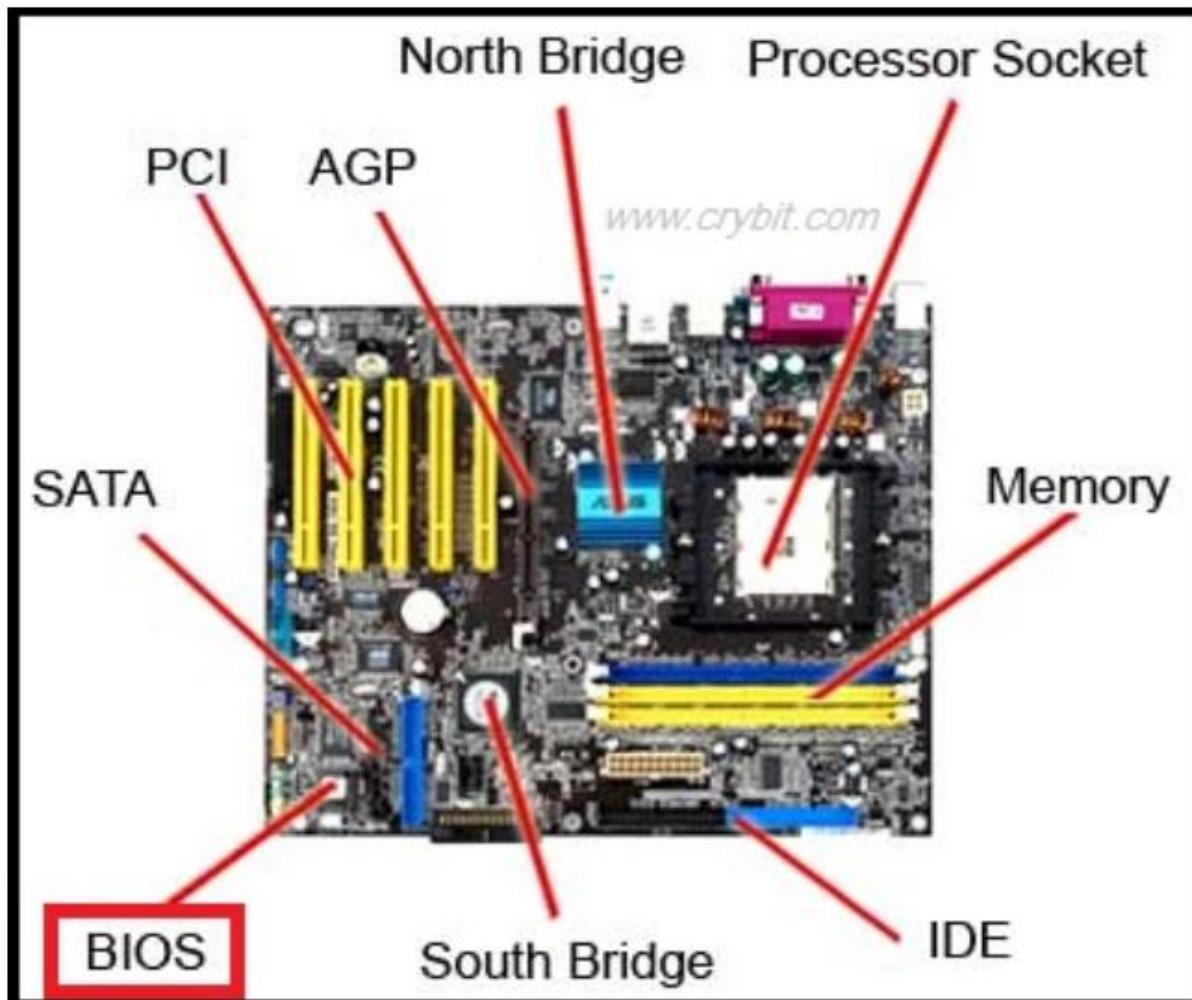


## Bios

---

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, cd-rom, or hard drive. You can press a key (typically F12 or F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

# Bios

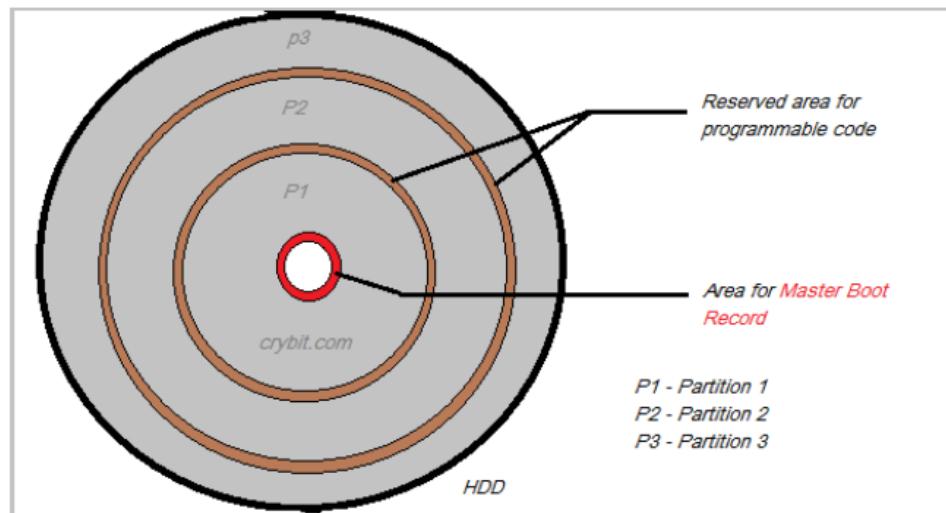
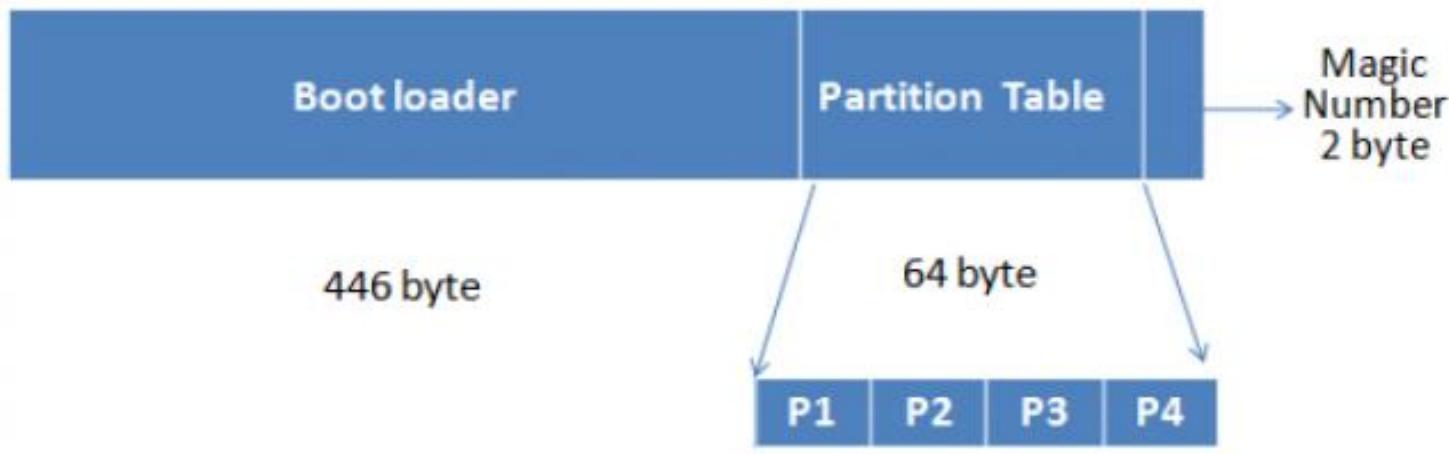


## MBR

---

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk.  
Typically /dev/hda, or /dev/sda
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB (or LILO in old systems).
- So, in simple terms MBR loads and executes the GRUB boot loader.

# MBR



# GRUB

---

- GRUB stands for Grand Unified Bootloader.
- If you have multiple kernel images installed on your system, you can choose which one to be executed.
- GRUB displays a splash screen, waits for few seconds, if you don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- Grub configuration file is /boot/grub/grub.conf (/etc/default/grub.d is a link to this).

# Kernel

---

- Mounts the root file system as specified in the “root=” in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a ‘ps -ef | grep init’ and check the pid.
- initrd stands for Initial RAM Disk.
- initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

# Init

---

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
- 0 – halt
- 1 – Single user mode
- 2 – Multiuser, without NFS
- 3 – Full multiuser mode
- 4 – unused
- 5 – X11
- 6 – reboot
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.

- Execute ‘grep initdefault /etc/inittab’ on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
- Typically you would set the default run level to either 3 or 5.

## Runlevel programs

---

- When the Linux system is booting up, you might see various services getting started. For example, it might say “starting sendmail .... OK”. Those are the runlevel programs, executed from the run level directory as defined by your run level.

## Runlevel programs

---

- Depending on your default init level setting, the system will execute the programs from one of the following directories.
- Run level 0 – /etc/rc.d/rc0.d/
- Run level 1 – /etc/rc.d/rc1.d/
- Run level 2 – /etc/rc.d/rc2.d/
- Run level 3 – /etc/rc.d/rc3.d/
- Run level 4 – /etc/rc.d/rc4.d/
- Run level 5 – /etc/rc.d/rc5.d/
- Run level 6 – /etc/rc.d/rc6.d/

## Runlevel programs

---

- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc\*.d/ directories, you would see programs that start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the syslog deamon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

# Install Ubuntu in Windows 10

---

- <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
- In the next step, it's time to install the graphical desktop programs. You do this by running these programs from Bash:
- <https://www.zdnet.com/article/how-to-run-the-native-ubuntu-desktop-on-windows-10/>
- apt-get install ubuntu-desktop
- apt-get install unity
- apt-get install compiz-core
- apt-get install compizconfig-settings-manager
- export DISPLAY=localhost:0
- Ccsm

# Install Ubuntu in Windows 10

---

- sudo apt-get install xfce4
- Xfce4-session

# How to access and log in to a UNIX system



- <https://cocalc.com/doc/terminal.html>



# How to access and log in to a UNIX system

```
eswaribala@DESKTOP-55AGI0I:~$ sudo login
[sudo] password for eswaribala:
DESKTOP-55AGI0I login: eswaribala
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 4.4.0-18362-Microsoft x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Tue Aug  4 22:30:36 IST 2020

System load:  0.52      Users logged in:      0
Usage of /home: unknown  IPv4 address for eth2:  169.254.131.111
Memory usage:  52%       IPv4 address for eth4:  192.168.148.113
Swap usage:   0%       IPv4 address for wifi0: 192.168.0.8
Processes:    12

0 updates can be installed immediately.
0 of these updates are security updates.

Last login: Sat Jul 18 18:07:18 IST 2020 on tty1
eswaribala@DESKTOP-55AGI0I:~$
```

# How to access and log in to a UNIX system



- Every UNIX system has at least one person, called the system administrator, whose job is to maintain the system.
- The system administrator is also responsible for adding new users to the system, and for setting up the initial work environment.
- If you are on a multiuser system, you will have to ask the system administrator to set up a login for you.
- If you are the only user on the system, you will be the system administrator.
- During the installation of your UNIX variant, you will be asked to select a login name and password.

# How to access and log in to a UNIX system



- In general, your login name can be almost any combination of letters and numbers, although there are a few constraints:
- Your login name must be more than two characters long.
- If it is longer than eight, only the first eight characters are relevant.
- It must contain only lowercase letters and numbers and must begin with a lowercase letter.
- No symbols or spaces are allowed.
- It cannot be the same as another login name already in use. Some login names are customarily reserved for certain uses; for example, root is often a login name for the system administrator (sometimes called the superuser ).

# How to access and log in to a UNIX system



- Choosing a Password
- If you begin by installing a UNIX variant on your own system, it will ask you to choose a password when you select a login name.
- If your account is on a remote system, your system administrator will probably assign you a temporary password, which you should change the first time you log in.

# How to access and log in to a UNIX system



- Passwords must have at least six characters.
- Passwords must contain at least two alphabetic characters (uppercase or lowercase letters), and at least one number or symbol. Note that UNIX is sensitive to case, so WIZARD is a different password than w1zard.
- Your login name with its letters reversed or shifted cannot be used as a password.
- For example, if your login name is msilver, you cannot choose silverm or revlism as a password.
- The passwords 3hrts&3lyonz and R0wkS+@r are both valid, but kilipuppy (no numeric or special characters) and Red1 (too short) are not.

# UNIX System Password Security

---

- Your first contact with security on your UNIX system is choosing a password.
- Simple passwords are easily guessed. A large commercial dictionary contains about 250,000 words, which can be checked as passwords in less than two minutes of computer time.
- All dictionary words spelled backward takes about another minute.
- All dictionary words preceded or followed by the digits 0–99 can be checked in just a few more minutes.

# UNIX System Password Security

---

- Here are some guidelines:
- Avoid easily guessed passwords, such as your name or the names of family members or pets.
- Also avoid your address, your car's license plate, and any other phrase that someone might associate with you.
- Avoid words or names that exist in a dictionary (in any language, not just English).
- Avoid trivial modifications of dictionary words. For example, normal words with replacement of certain letters with numbers: mid5umm3r, sn0wball, and so forth.



# UNIX System Password Security

## **sudo apt install cracklib-runtime**

```
eswaribala@DESKTOP-55AGI0I:~$ sudo apt install cracklib-runtime
[sudo] password for eswaribala:
Sorry, try again.
[sudo] password for eswaribala:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
 libcrack2 wamerican
The following NEW packages will be installed:
 cracklib-runtime libcrack2 wamerican
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 378 kB of archives.
Selecting previously unselected package libcrack2:amd64.ill be used.
(Reading database ... 32998 files and directories currently installed.)
Preparing to unpack .../libcrack2_2.9.6-3.2_amd64.deb ...
Unpacking libcrack2:amd64 (2.9.6-3.2) ...
Selecting previously unselected package cracklib-runtime.
Preparing to unpack .../cracklib-runtime_2.9.6-3.2_amd64.deb ...
Unpacking cracklib-runtime (2.9.6-3.2) ...
Unpacking wamerican (2018.04.16-1) ...#####
Setting up wamerican (2018.04.16-1) ...#####
Setting up libcrack2:amd64 (2.9.6-3.2) ...#####
Setting up cracklib-runtime (2.9.6-3.2) ...#####
eswaribala@DESKTOP-55AGI0I:~$
```

# UNIX System Password Security Strength



**echo viki | cracklib-check**

```
eswaribala@DESKTOP-55AGI0I:~$ echo viki | cracklib-check
viki: it is too short
eswaribala@DESKTOP-55AGI0I:~$
```

# Generate Strong Passwords

```
eswaribala@DESKTOP-55AGI0I:~$ openssl rand -base64 32
P1Nc/z4z8i1usnQi71INwZHEWq/vnwY2xcvkHyq+RP4=
eswaribala@DESKTOP-55AGI0I:~$ sudo apt-get install pwgen
[sudo] password for eswaribala:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  pwgen
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 18.1 kB of archives.
After this operation, 52.2 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 pwgen amd64 2.08-2 [18.1 kB]
Fetched 18.1 kB in 1s (21.6 kB/s)
Selecting previously unselected package pwgen.
(Reading database ... 33046 files and directories currently installed.)
Preparing to unpack .../pwgen_2.08-2_amd64.deb ...
Unpacking pwgen (2.08-2) ...
Setting up pwgen (2.08-2) ...
Processing triggers for man-db (2.9.1-1) ...
eswaribala@DESKTOP-55AGI0I:~$ pwgen 14 1
Yooh1ejaiBaj8
eswaribala@DESKTOP-55AGI0I:~$
```

# Generate Strong Passwords

```
eswaribala@DESKTOP-55AGI0I:~$ gpg --gen-random --armor 1 14
rWGRZwmsEVfYtuf0i0c=
eswaribala@DESKTOP-55AGI0I:~$ gpg --gen-random --armor 1 32
CxQ9/8xJSS6LlcjpEBW20dPGw6KDRkGBIVBwrJN+CwE=
eswaribala@DESKTOP-55AGI0I:~$
```

--tofu-policy VALUE      set the TOFU policy for a key

Options:

-a, --armor	create ascii armored output
-r, --recipient USER-ID	encrypt for USER-ID
-u, --local-user USER-ID	use USER-ID to sign or decrypt
-z N	set compress level to N (0 disables)
--textmode	use canonical text mode
-o, --output FILE	write output to FILE
-v, --verbose	verbose
-n, --dry-run	do not make any changes
-i, --interactive	prompt before overwriting
--openpgp	use strict OpenPGP behavior

(See the man page for a complete listing of all commands and options)

Examples:

-se -r Bob [file]	sign and encrypt for user Bob
--clear-sign [file]	make a clear text signature
--detach-sign [file]	make a detached signature
--list-keys [names]	show keys
--fingerprint [names]	show fingerprints

# Entering Commands

```
eswaribala@DESKTOP-55AGI0I:~$ date
Tue Aug  4 23:18:16 IST 2020
eswaribala@DESKTOP-55AGI0I:~$ ls -l
total 0
-rwxrwxrwx 1 eswaribala eswaribala 17 Jul 18 17:19 sample.txt
eswaribala@DESKTOP-55AGI0I:~$ sudo ls -l
[sudo] password for eswaribala:
total 0
-rwxrwxrwx 1 eswaribala eswaribala 17 Jul 18 17:19 sample.txt
eswaribala@DESKTOP-55AGI0I:~$ cal
        August 2020
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 31
eswaribala@DESKTOP-55AGI0I:~$ cal 4 2020
        April 2020
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
eswaribala@DESKTOP-55AGI0I:~$
```

# Entering Commands

## Who logged in whoami

```
eswaribala@DESKTOP-55AGI0I: ~
```

```
eswaribala@DESKTOP-55AGI0I:~$ whoami  
eswaribala  
eswaribala@DESKTOP-55AGI0I:~$
```

# Entering Commands

## Who logged in who -H, w, who -u, ps au

```
eswaribala@DESKTOP-55AGI0I:~$ eswaribala@DESKTOP-55AGI0I:~$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2836  0.0  0.0    8928   240 tty1      Ss  23:18   0:00 /init
eswarib+  2837  0.0  0.0   18172   3620 tty1      S  23:18   0:00 -bash
root      2857  0.0  0.0   18912   2748 tty1      S  23:20   0:00 sudo login
root      2858  0.0  0.0   18564   2668 tty1      S  23:20   0:00 login
eswarib+  2910  0.0  0.0   18080   3608 tty1      S  23:20   0:00 -bash
root      2927  0.0  0.0   18912   2740 tty1      S  23:22   0:00 sudo login
root      2928  0.2  0.0   18564   2668 tty1      S  23:22   0:00 login
eswarib+  2980  0.1  0.0   18080   3608 tty1      S  23:22   0:00 -bash
eswarib+  2992  0.0  0.0   18648   1892 tty1      R  23:23   0:00 ps au
eswaribala@DESKTOP-55AGI0I:~$
```



# Entering Commands

```
eswaribala@DESKTOP-55AGI0I:~$ sudo apt install finger
[sudo] password for eswaribala:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  finger
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 16.9 kB of archives.
After this operation, 51.2 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 finger amd64 0.17-17 [16.9 kB]
Fetched 16.9 kB in 1s (14.1 kB/s)
Selecting previously unselected package finger.
(Reading database ... 33051 files and directories currently installed.)
Preparing to unpack .../finger_0.17-17_amd64.deb ...
Unpacking finger (0.17-17) ...
Setting up finger (0.17-17) ...
Processing triggers for man-db (2.9.1-1) ...
eswaribala@DESKTOP-55AGI0I:~$ finger eswaribala
Login: eswaribala          Name:
Directory: /home/eswaribala      Shell: /bin/bash
Last login Tue Aug  4 23:22 (IST) on tty1
No mail.
No Plan.
eswaribala@DESKTOP-55AGI0I:~$
```



Type here to search





# Update Password

```
vigneshbala@DESKTOP-55AGI0I:~  
vigneshbala@DESKTOP-55AGI0I:~$ passwd  
Changing password for vigneshbala.  
Current password:  
New password:  
Retype new password:  
passwd: password updated successfully  
vigneshbala@DESKTOP-55AGI0I:~$
```

# Logout

---

- Logging Out
- When you finish your work session and wish to leave the UNIX system, type exit (or CTRL-D) to log out. After a few seconds, your UNIX system will display the “login:” prompt:
- \$ exit
- login:
- This shows that you have logged out, and that the system is ready for another user to log in using your terminal.

# Command Summary

Command	Use
<b>passwd</b>	Change your password
<b>date</b>	Get the current date and time
<b>cal</b>	Display a calendar
<b>who</b>	List all users who are currently logged in
<b>finger <i>username</i></b>	Get information about <i>username</i>
<b>write <i>username</i></b>	Send a chat message to <i>username</i>
<b>talk <i>username</i></b>	Open a chat session with <i>username</i>
<b>mesg y</b> <b>mesg n</b>	Accept or block incoming messages
<b>man <i>command</i></b>	Get information about <i>command</i>
<b>mailx</b> <b>mailx <i>address</i></b>	Read e-mail messages, or send an e-mail to <i>address</i>
<b>exit</b>	Log out of the system
<b>shutdown poweroff</b>	Turn off the machine

# Working with Files and Directories

---

- The UNIX file system provides a powerful and flexible way to organize and manage your information.
- Files
- A file is the basic structure that stores information on the UNIX System (and on Windows systems, as well).
- Conceptually, a computer file is similar to a paper document. Technically a file is a sequence of bytes that is stored somewhere on a storage device, such as a hard drive.
- A file can contain any kind of information that can be represented as a sequence of bytes.
- Word processing documents, bitmap images, and computer programs are all examples of files.

# Filenames

---

- Every file has a title, called a filename.
- A filename can be almost any sequence of characters, and up to 255 characters long. (On some older versions of UNIX, two filenames are considered the same if the first 14 characters are identical, so be careful if you use long filenames on these systems.)
- You can use any ASCII character in a filename except for the null character (ASCII NUL) or the slash (/), which has a special meaning in the UNIX file system.
- The slash acts as a separator between directories and files.

# Filenames

---

- Even though UNIX allows you to choose filenames with special characters, it is a good idea to stick with alphanumeric characters (letters and numbers) when naming files.
- You may encounter problems when you use or display the names of files containing nonalphanumeric characters. In particular, although the following characters can be used in filenames, it is better to avoid them.
- Many of these characters have special meanings in the command shell, which makes them difficult to work with in filenames.

# Filenames

! (exclamation point)	* (asterisk)	{,} (brackets)
# (pound sign)	? (question mark)	; (semicolon)
& (ampersand)	\ (backslash)	^ (caret)
(pipe)	(,) (parentheses)	tab
@ (at sign)	‘,’ (single or double quotes)	space
\$ (dollar sign)	< , > (left or right arrow)	backspace

# Filenames

---

## Capitalization

- Windows does not distinguish between uppercase and lowercase letters in filenames.
- You could save a file with the name Notes.DOC and find it by searching for notes.doc.
- The UNIX file system, however, is case-sensitive, meaning that uppercase and lowercase letters are distinct. In UNIX, NOTES, Notes, and notes would be three different files.
- If you save a file with the name Music, you will not find it by searching for music. This also applies to commands in UNIX.
- If you are trying to log out with the exit command, typing EXIT will not work.
- By the way, this explains why URLs (web addresses) can be case-sensitive, since the first web server was created on a UNIX-based platform, and many web servers still run UNIX.

# Filenames

---

## Filename Extensions

- In Windows, filenames typically consist of a base name, followed by a period and a short filename extension.
- Many Windows programs depend on the extension to determine how to use the file.
- For example, a file named solitaire.exe is considered to be a file named solitaire with the extension .exe, where the .exe extension tells Windows that it is an executable program.
- If the file extension is altered or deleted, it will be more difficult to work with the file in Windows.

# Filenames

---

## Filename Extensions

- In UNIX, file extensions are conveniences, rather than a necessary part of the filename.
- They can help you remember the content of files, or help you organize your files, but they are usually optional.
- In fact, many UNIX filenames do not have an extension.
- For example, an executable program would typically have a name like `solitaire` rather than `solitaire.exe`.
- In addition, filename extensions in UNIX can be longer than three characters.
- For example, some people use `.backup` to indicate a backup copy of a file, so `notes.backup` would be an extra copy of the file `notes`.

# Filenames

---

## Filename Extensions

Extension	File Type	Extension	File Type
.au	Audio	.mpg, .mpeg	MPEG video
.c	C language source code	.o	Object file (compiled and assembled code)
.cc	C++ source code	.pl	Perl script
.class	Compiled Java file	.ps	PostScript file
.conf	Configuration file	.py	Python script
.d	Directory	.sh	Bourne shell script
.gif	GIF image	.tar	tar archive
.gz	Compressed with gzip	.tar.Z, .tar.gz	Files that have been archived with tar and then compressed
.h	Header file for a C program	.tex	Text formatted with Tex/LaTeX
.html	Webpage	.txt	ASCII text
.jar	Java archive	.uu, .uue	Uuencoded file
.java	Java source code	.wav	Wave audio
.jpg, .jpeg	JPEG image	.z	Compressed with pack
.log	Log file	.Z	Compressed with compress

# Filenames

---

## Filename Extensions

- UNIX files can have more than one extension.
- For example, the file book.tar.Z is a file that has first been archived using the tar command (which adds the extension .tar ) and then compressed using the compress command (which adds the .Z).
- This enables a single script to both decompress the file and untar it, using the filename as input and parsing each of the extensions to perform the appropriate task.
- The flexibility of filename conventions in UNIX allow for some variation in filenames.
- A program written in Perl could have the filename program.perl, the more frequently used program.pl, or even just the name program.
- You can even create your own file extensions.

# Directories

---

- Files contain the information you work with.
- Directories provide a way to organize your files.
- A directory is just a container for as many files as you care to put in it.
- If you think of a file as analogous to a document in your office, then a directory is like a file folder.
- In fact, directories in UNIX are exactly like folders in Windows.
- For example, you may decide to create a directory to hold all of your notes.
- You could name it Notes and use it to hold only files that are your notes, keeping them separated from your e-mail, programs, and other files.

## Subdirectories

---

- A directory can also contain other directories.
- A directory inside another directory is called a subdirectory.
- You can create as many subdirectories inside a particular directory as you wish.

# Choosing Directory Names

---

- It is a good idea to adopt a convention for naming directories so that they can be easily distinguished from ordinary files.
- Some people give directory names that are all uppercase letters, some use directory names that begin with an uppercase letter, and others distinguish directories using the extension .d or .dir.
- For example, if you decide to use names beginning with an uppercase letter for directories and avoid naming ordinary files this way, you will know that Notes, Misc, Multimedia, and Programs are all directories, whereas note3, misc.note, mm\_5, and progmmA are all ordinary files.

# The Hierarchical File Structure

---

- Because directories can contain other directories, which can in turn contain other directories, the UNIX file system is called a hierarchical file system.
- Within the UNIX System, there is no limit to the number of files and directories you can create in a directory that you own.
- File systems of this type are often called tree-structured file systems, because each directory allows you to branch off into other directories and files.
- Tree-structured file systems are usually depicted upside-down, with the root of the tree at the top of the drawing.

# The Hierarchical File Structure

```
eswaribala@DESKTOP-55AGI0I:~$ sudo apt install tree
[sudo] password for eswaribala:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
tree
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 43.0 kB of archives.
Selecting previously unselected package tree.sk space will be used.
(Reading database ... 33057 files and directories currently installed.)
Preparing to unpack .../tree_1.8.0-1_amd64.deb ...
Unpacking tree (1.8.0-1) ...
Setting up tree (1.8.0-1) ...
Processing triggers for man-db (2.9.1-1) ...
eswaribala@DESKTOP-55AGI0I:~$ tree /etc
/etc
├── NetworkManager 1s (35.5 kB/s)
│   └── dispatcher.d
│       └── hook-network-manager
├── PackageKit
│   ├── PackageKit.conf
│   └── Vendor.conf
└── X11
    ├── Xreset
    ├── Xreset.d
    │   └── README
    ├── Xresources
    │   └── x11-common
    ├── Xsession
    ├── Xsession.d
    │   ├── 20dbus_xdg-runtime
    │   └── 20x11-common_process-args
```

# The Hierarchical File Structure

```
eswaribala@DESKTOP-55AGI0I:~$ tree /etc/apache2
^C
/etc/apache2
├── apache2.conf
├── apache2.conf.bak.2020-08-03_191943
├── apache2.conf.save
├── apache2.conf.save.1
└── conf-available
    ├── charset.conf
    ├── localized-error-pages.conf
    ├── other-vhosts-access-log.conf
    ├── security.conf
    └── serve-cgi-bin.conf
├── conf-enabled
    ├── charset.conf  -> ../conf-available/charset.conf
    ├── localized-error-pages.conf  -> ../conf-available/localized-error-pages.conf
    ├── other-vhosts-access-log.conf  -> ../conf-available/other-vhosts-access-log.conf
    ├── security.conf  -> ../conf-available/security.conf
    └── serve-cgi-bin.conf  -> ../conf-available/serve-cgi-bin.conf
├── envvars
└── magic
└── mods-available
    ├── access_compat.load
    ├── actions.conf
    ├── actions.load
    ├── alias.conf
    ├── alias.load
    ├── allowmethods.load
    ├── asis.load
    ├── auth_basic.load
    ├── auth_digest.load
    ├── auth_form.load
    └── authn_anon.load

eswaribala@DESKTOP-55AGI0I:~$
```

This screenshot shows a terminal window on a Windows operating system displaying the output of the 'tree' command for the '/etc/apache2' directory. The output shows a hierarchical file structure for Apache 2 configuration files. The structure includes 'apache2.conf', backup files ('apache2.conf.bak.2020-08-03\_191943'), and several configuration files ('security.conf', 'charset.conf', etc.) located in 'conf-available'. These available files are linked to their enabled counterparts in the 'conf-enabled' directory. Additionally, there are 'envvars' and 'magic' files, and a 'mods-available' directory containing various loadable modules like 'access\_compat.load' and 'auth\_digest.load'.

At the bottom of the screen, the Windows taskbar is visible, showing icons for various applications including Microsoft Edge, Google Chrome, File Explorer, and others. The system tray on the right shows standard icons for battery, signal strength, and date/time (05/08/2020, 00:03).

# Pathnames

---

- If there are two files with the same name, but in different locations in the file system.
- There is a save file in the Email directory, and another file called save in the Work directory.
- In order to distinguish files with the same name, the UNIX System allows you to specify filenames by including the location of the file in the directory tree.
- This type of name is called a pathname, because it is a listing of the directories you travel through along the path you take to get to the file.
- The path through the file system starts at root (/), and the names of directories and files in a pathname are separated by slashes.
- For example, the pathname for one of the save files is
  - /home/raf>Email/save
  - and the pathname for the other is
  - /home/raf/Work/save

## Pathnames

---

- Pathnames that trace the path from root to a file are called full (or absolute) pathnames.
- Specifying a full pathname provides a complete and unambiguous name for a file. In a full pathname, the first slash (/) refers to the root of the file system.
- All the other slashes separate the names of directories, until the last slash separates the filename from the name of the directory it's in.
- Using full pathnames can be awkward when there are many levels of directories, as in this filename:
- `/home/dkraut/Work/cs106x/Proj_1/lib/Source/strings.c`

# Pathnames

---

- **Relative Pathnames**
- You do not always have to specify the full pathnames when you refer to files.
- As a convenient shorthand, you can also specify a path to a file relative to your present directory Such a pathname is called a relative pathname.
- Instead of starting with a / for root, the relative pathname starts with the name of a subdirectory For example, suppose you are in your home directory, /home/raf.
- The relative path for the save file in the Email subdirectory is Email/save, and the relative path for the other save file is Work/save.

# Pathnames

---

- **Specifying the Current Directory**

- A single dot (.) is used as a shortcut to refer to the directory you are currently in.
- This directory is known as the current directory.

- **Specifying the Parent Directory**

- Two dots (.., pronounced “dot-dot”) refer to the parent directory of the one you are currently in.
- The parent directory is the one at the next higher level in the directory tree.
- Because the file system is hierarchical, all directories have a parent directory
- The dot-dot references can be used many times to refer to things far up in the file system. The following sequence, for example,
  - .../..
  - refers to the parent of the parent of the current directory.
  - If you are in Work, then .../.. is the same thing as the home directory, since home is the parent of raf, which is the parent of Work.

# Pathnames

---

- **Specifying a Home Directory**
- A tilde (~) can be used to refer to your home directory (Strictly speaking, this is a feature of the shell.)
- These shortcuts can be combined.
- For example, if your home directory is /home/raf, then
- ~/..../liz
- refers to the home directory for the user liz.
- You can also use a tilde followed by a login name to refer to another user's home directory For example, the shortcut ~nate refers to the user nate's home directory

# UNIX System File Types

---

- The file is the basic unit of the UNIX System. Within UNIX, there are four different types of files:
  - ordinary files, directories, symbolic links, and special files.
- **Ordinary Files**
  - As a user, most of the information that you work with will be stored as an ordinary file.
  - An ordinary file can contain data, such as text for documents or programs.
  - Image files and binary executables are also examples of ordinary files.

# UNIX System File Types

---

- **Links**
- Sometimes it is useful to have a file that is accessible from several directories, without making separate copies of the file.
- For example, suppose you are working with someone else, and you need to share information contained in a single data file that each of you can update.
- It would be convenient for each of you to have a copy in your home directory However, you do not want to make separate copies of the file, because it will be hard to keep them in sync.
- A link is not a kind of file but instead is a second name for a file. With a link, only one file exists on the disk, but it may appear in two places in the directory structure.
- This can allow two users to share the same file.
- Any changes that are made to the file will be seen by both users.
- This type of link is sometimes called a hard link, to distinguish it from a symbolic link

# UNIX System File Types

---

- **A hard Link**

- can't cross the file system boundaries (i.e. A hardlink can only work on the same filesystem),
- can't link directories,
- has the same inode number and permissions of original file,
- permissions will be updated if we change the permissions of source file,
- has the actual contents of original file, so that you still can view the contents, even if the original file moved or removed.

# UNIX System File Types

---

- **A hard Link**

- The `ln` command creates a link between files, which enables you to make a single file accessible at two or more locations in the directory system.
- The following links the file `project.main` in `dkraut`'s home directory with a new file of the same name in the current directory:
- `$ ln /home/dkraut/project.main project.main`

# UNIX System File Types

---

- **A soft link**

- can cross the file system,
- allows you to link between directories,
- has different inode number and file permissions than original file,
- permissions will not be updated,
- has only the path of the original file, not the contents.

# UNIX System File Types

- **A soft link**

- Symbolic links are created by using the `ln` command with the `-s` (symbolic) option.
- The following example shows how you could use `ln` to link a file in the `/var` file system to an entry in one of your directories within the `/home` file system:
  - `$ ln -s /var/X/docs/readme temp/x.readme`
  - This will create a symbolic link called `x.readme` in the `temp` directory
  - The second argument to `ln` is optional; if you do not specify the name of the new file, it will create a symbolic link with the same name as the target file. So, for example
    - `$ ln -s /usr/bin/firefox/firefox`
    - will create a file called `firefox` in the current directory that is a symbolic link to `/usr/bin/firefox/firefox`.
    - Symbolic links also enable you to link directories.
    - The command
      - `$ ln -s /home/dkraut/work/cs106x/proj1/lib/Source Project`

# UNIX System File Types

- **Listing Directory Contents with Marks**
- When you use the ls command, you do not know whether a name refers to an ordinary file, a program that you can run, or a directory. Running the ls command with the -F option produces a list in which the names are marked with symbols that indicate the kind of file that each name refers to.
- Names of directories are listed with / (a slash) following their names. Executable files (those that can be run as programs) are listed with \* (an asterisk) following their names.
- Symbolic links are listed with @ (an “at” sign) following their names.
- For instance, suppose that you run ls with the -F option to list the contents of a directory, producing the following result:
- **\$ ls -F**
- Email/ notes Projects@
- This example shows that the directory contains the ordinary file notes, the directory Email, and a symbolic link Projects.

# UNIX System File Types

```
eswaribala@DESKTOP-55AGI0I:~  
6 directories, 194 files  
eswaribala@DESKTOP-55AGI0I:~$ ls  
sample.txt  
eswaribala@DESKTOP-55AGI0I:~$ ls /etc/apache2  
apache2.conf           apache2.conf.save    conf-available   envvars   mods-available  ports.conf     sites-enabled  
apache2.conf.bak.2020-08-03_191943 apache2.conf.save.1  conf-enabled    magic     mods-enabled    sites-available  
eswaribala@DESKTOP-55AGI0I:~$ ls -F  
sample.txt*  
eswaribala@DESKTOP-55AGI0I:~$ ls /etc/apache2 -F  
apache2.conf           apache2.conf.save    conf-available/  envvars   mods-available/  ports.conf     sites-enabled/  
apache2.conf.bak.2020-08-03_191943 apache2.conf.save.1  conf-enabled/   magic     mods-enabled/   sites-available/  
eswaribala@DESKTOP-55AGI0I:~$ ls /etc/  
NetworkManager          debconf.conf      insserv.conf.d  nanorc      sensors.d  
PackageKit              debian_version  inxi.conf       netplan  
UPower                  default          iproute2        network  
X11                     deluser.conf    iscsi           networkd-dispatcher  
acpi                    depmod.d       issue          networks  
adduser.conf            dhcp            issue.net      newt  
alsa                   dictionaries-common java-8-openjdk nsswitch.conf  
alternatives            dpkg             kernel         openvpn  
anacrontab              e2scrub.conf   kernel-img.conf opt  
apache2                 ec2_version     kerneloops.conf os-release  
apg.conf                emacs           landscape     overlayroot.conf  
apm                     environment    ld.so.cache    overlayroot.local.conf  
apparmor                environment.d  ld.so.conf     pam.conf  
apparmor.d              ethertypes     ldap           pam.d  
apport                  firefox        legal          passwd  
appstream.conf          fonts          libao.conf    passwd-  
apt                     fprintd.conf   libaudit.conf  pcmcia  
at.deny                 fstab          fuse.conf    perl  
avahi                  fwupd          libblockdev   pki  
bash.bashrc              gai.conf       libnl-3       pm  
bash_completion          libpaper.d    libpaper.d  
                                pm
```

# UNIX System File Types

---

- **Listing Files in the Current Directory Tree**
- You can add the -R (recursive) option to the ls command to list all the files in your current directory, along with all the files in each of its subdirectories, and so on.
- For example,
  - \$ ls -R

# UNIX System File Types

---

- **Viewing Files**
- The simplest and most basic way to view a file is with the cat command. cat (short for concatenate) takes any files you specify and displays them on the screen.
- For example, you could use cat to display on your screen the contents of the file review:
- \$ cat review

# UNIX System File Types

---

- **Viewing Files with Special Characters**
- The cat command recognizes eight-bit characters. In earlier versions of UNIX, it only recognized seven-bit characters.
- This enhancement permits cat to display characters from extended character sets, such as the kanji characters used to represent Japanese words.
- Cat –v sample.txt

# UNIX System File Types

---

- **Directing the Output of cat**
- You can send the output of cat to a file as well as to the screen.
- For instance,
  - \$ cat physics > physics.backup
- In order to add information to the end of a file, do the following:
  - \$ cat notes.august >> notes

# UNIX System File Types

---

- **Combining Files and Using Wildcards**
- You can use cat to combine a number of files into one. For example, consider a directory that contains material being used in writing a chapter, as follows:
  - \$ cat section1 section2 section3 > chapter.3
  - \$ cat section\* > chapter.3
  - \$ cat \*1 \*2 > temp

# UNIX System File Types

---

- **Creating a File**
- So far, all the examples you have seen involved using cat to copy one or more normal files, either to another file or to your screen. But other possibilities exist.
- Just as your screen is the default output for cat and other commands, your keyboard is the default input.
- If you do not specify a file to use as input, cat will simply copy everything you type to its output.
- This provides a way to create simple files without using an editor.
- The command
  - \$ cat > names
  - Nate nate@engineer.com
  - Rebecca rlf@library.edu
  - CTRL-D

# UNIX System File Types

---

- Using cat in this way (`cat > names`) creates the file names if it does not already exist and overwrites (replaces) its contents if it does exist.
- You can use cat to add material to a file as well. For example,
- **\$ cat >> names**
- **Dan dkraut@bio.ca.edu**
- **CTRL-D**

# UNIX System File Types

- Another command, touch, can also be used to create a file.
- \$ touch notes

```
eswaribala@DESKTOP-55AGI0I:~$ touch notes
eswaribala@DESKTOP-55AGI0I:~$ ls
notes sample.txt software
eswaribala@DESKTOP-55AGI0I:~$
```

# UNIX System File Types

- Moving Around in Directories

```
eswaribala@DESKTOP-55AGI0I:~$ pwd
/home/eswaribala
eswaribala@DESKTOP-55AGI0I:~/home$ cd /etc
eswaribala@DESKTOP-55AGI0I:/etc$ cd /etc/apache2
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ cd ..
eswaribala@DESKTOP-55AGI0I:/etc$ cd ../../..
-bash: cd: too many arguments
eswaribala@DESKTOP-55AGI0I:/etc$ cd ../..
eswaribala@DESKTOP-55AGI0I:$ cd ~
eswaribala@DESKTOP-55AGI0I:~/home$ pwd
/home/eswaribala
eswaribala@DESKTOP-55AGI0I:~/home$
```

# UNIX System File Types

- Moving and Renaming Files and Directories

```
eswaribala@DESKTOP-55AGI0I:~$ pwd
/home/eswaribala
eswaribala@DESKTOP-55AGI0I:~/home$ cd /etc
eswaribala@DESKTOP-55AGI0I:/etc$ cd /etc/apache2
eswaribala@DESKTOP-55AGI0I:/etc/apache2$ cd ..
eswaribala@DESKTOP-55AGI0I:/etc$ cd ../..
-bash: cd: too many arguments
eswaribala@DESKTOP-55AGI0I:/etc$ cd ../..
eswaribala@DESKTOP-55AGI0I:$ cd ~
eswaribala@DESKTOP-55AGI0I:~$ pwd
/home/eswaribala
eswaribala@DESKTOP-55AGI0I:~$ mkdir files
eswaribala@DESKTOP-55AGI0I:~$ ls
files notes sample.txt software
eswaribala@DESKTOP-55AGI0I:~$ mv notes /files
mv: cannot move 'notes' to '/files': Permission denied
eswaribala@DESKTOP-55AGI0I:~$ sudo mv notes /files
[sudo] password for eswaribala:
eswaribala@DESKTOP-55AGI0I:~$ ls
files sample.txt software
eswaribala@DESKTOP-55AGI0I:~$ -
```

For example, the following command moves three files to the subdirectory called TermPaper:

```
$ mv section1 section2 section3 TermPaper
```

# UNIX System File Types

---

- **To protect mv overwrites the file**
- The following shows what happens if you try to use mv -i to rename the file totals to data when the data file already exists:
- \$ mv -i totals data
- mv: overwrite data?

# UNIX System File Types

---

- **Moving Directories**
- You can use a single mv command to move a directory and all of its files and subdirectories just as you'd use it to move a single file.
- For example, if the directory Final contains all of your finished work on a document, you can move it to a directory in which you keep all of the versions of that document, Project, as shown here:
- \$ ls Project
- Drafts
- \$ mv Final Project
- \$ ls Project
- Drafts Final

# UNIX System File Types

---

- **Copying Files**
- The cp command is similar to mv, except that it copies files rather than moving or renaming them.
- Cp follows the same model as mv: you name the files to be copied first and then give the destination.
- The destination can be a directory, a pathname for a file, or a new file in the current directory.
- The following command makes a backup copy of seattle and names the copy seattle.bk:
- `$ cp seattle seattle.bk`
- `cp -p unix.txt{,.bak.$(date +%F_%H%M%S)}`

# UNIX System File Types

- **Copying the Contents of a Directory**
- If you try to copy a directory, you will get an error message.
- A feature of cp (found on most versions of UNIX) is the -r (r ecursive) option that lets you copy an entire directory structure. Suppose you have a directory called Project, and you wish to make a backup copy
- The following command creates a new directory, called Project.Backup, and copies all of the files and subdirectories in Project to the new directory:
- `$ cp -r Project Project.Backup`

# UNIX System File Types

---

- **Removing Files**
- To get rid of files you no longer want or need, use the rm (remove) command. rm deletes the named files from the file system, as shown in the following example:
- \$ ls
- notes research temp
- \$ rm temp
- \$ ls
- notes research

# UNIX System File Types

- **Removing Multiple Files**
- The rm command accepts several arguments and takes several options.
- If you specify more than one filename, it removes all of the files you named.
- The following command removes the two files left in the directory:
  - \$ rm notes research
  - \$ ls
  - \$
- The following will remove all files in the current directory:
  - \$ rm \*

# UNIX System File Types

---

- **Safely Removing Files**
- Almost every user has accidentally deleted files.
- In the preceding example, if you accidentally hit the SPACEBAR between the \* and the extension and type
- `$ rm * .rlf`
- you will delete all of the files in the current directory As typed, this command says to remove all files (\*), and then remove a file named .rlf.

# UNIX System File Types

---

- **Safely Removing Files**
- To avoid accidentally removing files, use rm with the -i (interactive) option.
- When you use this option, rm prints the name of each file and waits for your response before deleting it.
- To go ahead and delete the file, type y. Responding n or hitting ENTER will keep the file rather than deleting it.
- For example, in a directory that contains the files notes, research, and temp, the interactive option to rm gives you the following:
- \$ rm -i \*
- notes: y
- research: <ENTER>
- temp: y
- Your responses cause rm to delete both notes and temp, but not research.

# UNIX System File Types

---

- **Restoring Files**
- When you remove a file using the rm command, it is gone. If you make a mistake, you can only hope that the file is available somewhere on a backup file system (on a tape or disk).
- You can call your system administrator and ask to have the file you removed, say /home/you/Work/temp, restored from backup.
- If it has been saved, it can be restored for you.
- Systems differ widely in how, and how often, they are backed up.
- On a heavily supported system, all files are copied to a backup system every day and saved for some number of days, weeks, or months.
- On some systems, backups are done less frequently, perhaps weekly. On personal workstations, backups occur when you get around to doing them.

# UNIX System File Types

---

- **Creating a Directory**
- You can create new directories in your file system with the `mkdir` (make directory) command. It is used as follows:
  - `$ pwd`
  - Work
  - `$ ls`
  - notes research temp
  - `$ mkdir New`
  - `$ ls`
  - notes New research temp

# UNIX System File Types

---

- **Removing a Directory**
- There are two ways to remove or delete a directory If the directory is empty (it contains no files or subdirectories), you can use the `rmdir` (remove directory) command.
- If you try to use `rmdir` on a directory that is not empty, you'll get an error message.
- The following removes the directory `New` added in the preceding example:
- `$ rmdir New`

# UNIX System File Types

- **Removing a Directory**
- To remove a directory that is not empty, together with all of the files and subdirectories it contains, use rm with the -r (recursive) option, as shown here:
- \$ rm -r Work
- The -r option instructs rm to delete all of the files it finds in Work and then go to each of the subdirectories and delete all of their files, and so forth, concluding by deleting Work itself.
- Since rm -r removes all of the contents of a directory, be very careful in using it. You can add the -I option to step through all the files and directories, removing or leaving them one at a time.
  - \$ rm -ir Work
  - rm: descend into directory 'Work'? y
  - rm: remove regular empty file 'Work/final'? y
  - rm: remove regular empty file 'Work/save'? <RETURN>
  - \$ ls Work
  - save

# UNIX System File Types

---

- **Getting Information About File Types**
- Sometimes you just want to know what kind of information a file contains.
- For example, you may decide to put all your shell scripts together in one directory. You know that several scripts are scattered about in several directories, but you don't know their names, or you aren't sure you remember all of them.
- Or you may want to print all of the text files in the current directory, whatever their content.
- You can use several of the commands already discussed to get limited information about file contents.
- For example, `ls -l` shows you if a file is executable-either a compiled program or a shell script (batch file). But the most complete and most useful command for getting information about the type of information contained in files is `file`.
- `file` reports the type of information contained in each of the files you give it.
- The following shows typical output from using `file` on all of the files in the current directory:
- `$ file *`

# UNIX System File Types

---

- **Searching for Files**
- The command `locate` searches for a pattern in a database of filenames. For example,
- `$ locate pippin`
- searches the database for filenames containing the string “pippin”.
- The database contains the full pathname for each file, so this would find files in the directory `pippin-photos` as well as files such as
- `0915-pippin.jpg`.
- The `locate` command is very fast and easy to use. However, it will only work if the database of filenames is kept up to date.
- On many systems, the database is automatically updated once per day

# UNIX System File Types

---

- **Using find**
- The find command searches through the contents of one or more directories, including all of their subdirectories.
- You have to tell find in which directory to start its search.
- The following example
- searches user jmf's directory system for the file new\_data and prints the full pathname of any file with that name that it finds:
- \$ pwd
- /home/jmf
- \$ find . -name new\_data -print
- /home/jmf/Dir/Logs/new\_data
- /home/j mf/Cmds/new\_data

# UNIX System File Types

---

- **Using find**
- To search the entire file system, start in the system's root directory, represented by the /:
  - \$ find / -name new\_data –print
  - \$ find . /tmp/project -name new\_data –print
  - \$ find -name "\*data" –print
  - \$ find / -name new\_data -print > found &
  - \$ find . -name "music" –user eswaribala -mtime +7 -print

# UNIX System File Types

---

- **List Hidden Files**
- To see all files in this directory, use ls -a:
- \$ ls -a
- . . . .mailrc .profile Email notes Work

# UNIX System File Types

---

- **Controlling the Way ls Displays Filenames**
- You can use the `-x` option to have names of files displayed horizontally, in as many lines as necessary
- For example,
- `$ ls -x`
- You also can use the `-1` (one) option to have files displayed one line per row (as the old version of ls did), in alphabetical order:
- `$ ls -l`

# UNIX System File Types

---

- **Controlling the Way Is Displays Filenames**
- Combining Options to ls
- You can use more than one option to the ls command simultaneously For example, the following shows the result of using the ls command with the options -F and -a on a home directory:
- \$ ls -aF
- \$ ls -Fat
- example of what the long format of ls might look like:
- \$ ls -l

# UNIX System File Types

- **Controlling the Way Is Displays Filenames**

The first character in each line tells you what kind of file this is.

-	Ordinary file	c	Special character file
d	Directory	I	Symbolic link
b	Special block file	P	Named pipe special file

# UNIX System File Types

- **Unix Special Characters Or Metacharacters For File Manipulation**
- **Unix Filename Wildcards – Metacharacters**
- #1) '\*' – any number of characters:
  - This wild-card selects all the files that matches the expression by replacing the asterisk-mark with any set of zero or more characters.
  - Example1: List all files that start with the name 'file'. g. file, file1, file2, filenew
  - \$ ls file\*
  - Example2: List all files that end with the name 'file'. g. file, afile, bfile, newfile
  - \$ ls \*file

# UNIX System File Types

---

- **Unix Special Characters Or Metacharacters For File Manipulation**
- **Unix Filename Wildcards – Metacharacters**
- #2) ‘?’ – single character:
- This wild-card selects all the files that matches the expression by replacing the question-mark with any one character.
- Example1: List all files that have one character after ‘file’. g. file1, file2, filea
- \$ ls file?
- Example2: List all files that have two characters before ‘file’. g. dofile, tofile, a1file
- \$ ls ??file

# UNIX System File Types

- **Unix Special Characters Or Metacharacters For File Manipulation**
- **Unix Filename Wildcards – Metacharacters**
- #3) '[' range ']' – single character from a range:
- This wild-card selects all the files that matches the expression by replacing the marked range with any one character in the range.
- Example1: List all files that have a single digit after 'file'. g. file1, file2
- \$ ls file[0-9]
- Example2: List all files that have anyone letter before 'file'. g. afile, zfile
- \$ ls [a-z]file

# UNIX System File Types

- **Unix Special Characters Or Metacharacters For File Manipulation**
- **Unix Filename Wildcards – Metacharacters**
- # 4) '[' range ']\*' – multiple characters from a range:
  - This wild-card selects all the files that matches the expression by replacing the marked range with one or more characters from the range.
  - Example1: List all files that have digits after 'file'. g. file1, file2, file33
  - \$ ls file[0-9]\*

# UNIX System File Types

---

- **Permissions**
- The UNIX file system is designed to support multiple users.
- When many users are sharing one file system, it is important to be able to restrict access to certain files.
- The system administrator wants to prevent other users from changing important system files, for example, and many users have private files that they want to restrict others from viewing.
- File permissions are designed to address these needs.

# UNIX System File Types

---

- **Permissions for Files**
- There are three classes of file permissions, for the three classes of users: the owner (or user) of the file, the group the file belongs to, and all other users of the system.
- The first three letters of the permissions field, as seen in the output from `ls -l`, refer to the owner's permissions;
- the second three letters refer to the permissions for members of the file's group;
- and the last three to the permissions for any other users.

# UNIX System File Types

---

- **Permissions for Files**
- In the entry for the file named notes in the ls -l example shown in the preceding section, the first three letters, rwx, show that the owner of the file can read (r) it, write (w) to it, and execute (x) it.
- The second group of three characters, r-x, indicates that members of the group can read and execute the file but cannot write to it.
- The last three characters, r-x, show that all others can also read and execute the file but not write to it.
- If you have read permission for a file, you can view its contents.
- Write permission means that you can alter its contents.
- Execute permission means that you can run the file as a program.

# UNIX System File Types

---

- **Special Permissions**
- There are a few other codes that occasionally appear in permission fields.
- For example, the letter s can appear in place of an x in the user's or group's permission field.
- This s refers to a special kind of execute permission that is relevant primarily for programmers and system administrators.

# UNIX System File Types

---

- **The chmod Command**
- In the ls -l example, all of the files and directories have the same permissions set.
- Anyone on the system can read or execute any of them, but other users are not allowed to write, or alter, these files.
- Normally you don't want all your files set up this way.
- You will often want to restrict other users from being able to view your files, for example.
- At times, you may want to allow members of your work group to edit certain files, or even make some files public to anyone on the system.

# UNIX System File Types

---

- **The chmod Command**
- The UNIX System allows you to set the permissions of each file you own.
- Only the owner of a file or the superuser can alter the file permissions.
- You can independently manipulate each of the permissions to allow or prevent reading, writing, or executing by yourself, your group, or all users.

# UNIX System File Types

---

- **The chmod Command**
- To alter a file's permissions, you use the chmod (change mode) command.
- You specify the changes you want to make with a sort of code.
- First, show which set of permissions you are changing with u for user, g for group, or o for other.
- Second, specify how they should be changed with + (to add permission) or - (to subtract permission).
- Third, list the permissions to alter: r for read, w for write, or x for execute.
- Finally, specify the file or files that the changes refer to.

# UNIX System File Types

---

- **The chmod Command**
- \$ ls -l quotations
- -rwxr-xr-x 1 nate group1 346 Apr 27 03:32 quotations
- \$ chmod go-rx quotations
- \$ ls -l quotations
- -rwx 1 nate group1 346 Apr 27 03:32 quotations
- \$ chmod ugo+rwx quotations
- \$ ls -l quotations
- -rwxrwxrwx 1 nate group1 346 Apr 27 03:32 quotations

# UNIX System File Types

- **The chmod Command**
- Setting Absolute Permissions
- \$ chmod 700 quotations
- \$ ls -l quotations
- -rwxrwxrwx 1 nate group1 346 Apr 27 03:32  
quotations

	Owner	Group	Other
Read	4	0	0
Write	2	0	0
Execute	1	0	0
Sum	7	0	0

# UNIX System File Types

---

- **The chmod Command**
- \$ chmod go-rwx \*
- \$ chmod 700 \*
- \$ chmod -R u+r Email

# UNIX System File Types

---

- **Using umask to Set Permissions**
- The chmod command allows you to alter permissions on a file-by-file basis.
- The umask command allows you to do this automatically when you create any file or directory.
- Everyone has a default umask setting that is either set up either by the system administrator or included in a shell configuration file.

# UNIX System File Types

---

- **Using umask to Set Permissions**
- With the umask command, you specify the permissions that will be given to all files created after issuing the command.
- This means you will not have to worry about the file permissions for each individual file you create.
- Unfortunately, using umask to specify permissions is a little bit complicated.

# UNIX System File Types

---

- **Using umask to Set Permissions**
- There are two rules to remember:
- umask uses a numeric code for representing absolute permissions just as chmod does.
- For example, 777 means read, write, and execute permissions for user, group, and others (rwxrwxrwx).
- You specify the permissions you want by telling umask what to subtract from the full permissions value, 777 (rwxrwxrwx).

# UNIX System File Types

---

- **Using umask to Set Permissions**
- For example, after you issue the following command, all new files in this session will be given permissions of `rwxr-xr-x`:
- `$ umask 022`
- In this example, we want the new files to have the permission value 755. When we subtract 755 from 777, we get 022.
- This is the “mask” we used for the command.

# List user accounts

---

- `cut -d: -f1 /etc/passwd`
- `ps aux`

## Create User

---

- sudo adduser mohan

# UNIX System File Types

---

- **Changing the Owner of a File**
- Every file has an owner. When you create a file, you are automatically its owner.
- The owner usually has broader permissions for manipulating the file than other users.
- Sometimes you need to change the owner of a file; for example, if you take over responsibility for a file that previously belonged to another user.
- Even if someone else “gives” you a file by moving it to your directory that does not make you the owner.
- One way to become the owner of a file is to make a copy of it-when you make a copy, you are the owner of the new file.

# UNIX System File Types

---

- **Changing the Owner of a File**
- However, changing ownership by copying only works when the new owner copies the file from the old owner, which requires the new owner to have read permission on the file.
- A simpler and more direct way to transfer ownership is to use the chown (change owner) command.
- The chown command takes two arguments: the login name of the new owner and the name of the file.
- The following makes liz the new owner of the file contact\_info:
- `$ chown liz contact_info`
- Only the owner of a file (or the superuser) can use chown to change its ownership
- `chown -R liz Project`

## List Groups

---

- `cut -d: -f1 /etc/group`
- `sudo groupadd unixtraining #create group`
- `sudo usermod -a -G unixtraining mohan #add user to group`

# UNIX System File Types

---

- **Changing the Group of a File**
- Groups are meant to help sets of users who need to share files more closely than other users on the system.
- For example, all the students taking a particular class may belong to the same group, so that they can more easily share files when they collaborate on projects

# UNIX System File Types

---

- **Changing the Group of a File**
- Every file belongs to a group.
- Sometimes, such as when new groups are set up on a system or when files are copied to a new system, you may want to change the group to which a particular file belongs.
- This can be done using the chgrp (change group) command.
- **The chgrp command takes two arguments, the name of the new group and the name of the file.**
- The following command changes data\_file so that it belongs to the group students:
- \$ chgrp students data\_file

# UNIX System File Types

---

- You can use the `-R` (recursive) option of `chgrp` to change the group to which all the files in a directory belong.
- It works just like the `-R` option of `chown`.

# UNIX System File Types

---

- **Viewing Long Files**
- When you use cat to display a file, it prints the contents on your screen without pausing, so that long files quickly scroll past.
- A quick solution, when you only need to view a small part of the file, is to use cat and then hit BREAK when the part you want to read comes on the screen.
- This stops the program, but it leaves the output on the screen, so if your timing is good, you may get what you want.

# UNIX System File Types

---

- **Viewing Long Files**
- A somewhat better solution is to use the sequence CTRL-S, to make the output pause whenever you get a screen you want to look at, and CTRL-Q to resume scrolling.
- This way of suspending output to the screen works for all UNIX commands, not just cat.

# UNIX System File Types

---

- **Viewing Long Files**
- The best solution is to use a pager -a program that is designed specifically for viewing files.
- UNIX gives you a choice of two pagers, pg and more, which are standard with all versions of UNIX, as well as an enhanced pager called less, available for many versions of UNIX, including Linux.
- Less has more features than more and has pretty much replaced it.

# UNIX System File Types

---

- **Viewing Long Files (more)**
- To display the file newyork, just enter the command
- \$ more newyork
- To tell more to move ahead by a screen, press the SPACEBAR.
- To move ahead one line, press ENTER.
- The commands for half-screen motions, d and CTRL-D, are the same as in pg.
- To move backward by a screen, use b or CTRL-B.

# UNIX System File Types

---

- **Viewing Long Files (less)**
- The less command, an enhanced version of the more command, is a feature-rich pager that can be used to interactively display portions of a file.
- It can be used to move either forward or backward in a file.
- Since less reads in portions of files, rather than entire files, it is very efficient for large files.
- This will display the file newyork with less:
- `$ less newyork`

# UNIX System File Types

- **Viewing Long Files (less)**
- The less command has many useful options.
- For example, the -p option can be used to start less at the first occurrence of the pattern you specify (where the pattern is entered after the -p option)
- To display the file sanfrancisco, beginning with the first time the **pattern “SFO” appears**, you can use
- **\$ less -p SFO sanfrancisco**
- Among the other options supported by less are -s, which squeezes consecutive blank lines into a single blank line;
- -S, which chops off lines longer than the screen (discarding them instead of folding them into the next line); and -U, which displays backspaces and carriage returns as control characters.

# UNIX System File Types

---

- **Viewing Long Files (less)**
- Many commands can be used with the less pager to display different parts of a file.
- For example, you can scroll forward one window by entering SPACEBAR or f, and you can scroll backward one window by entering b.
- You can scroll forward one line by entering e or pressing ENTER, and you can scroll backward one line by entering y.
- You can scroll to the next occurrence of the string “pattern” using the command /pattern, and you can scroll to the preceding occurrence of this string using the command ? pattern.

# UNIX System File Types

---

- **Viewing Long Files (less)**
- You can find the next and preceding lines that do not contain the string “pattern” using the commands `!/pattern` and `?!pattern`, respectively.
- You can use various commands to move the cursor when using less to display files.
- For example, you can move one space left using the LEFT ARROW key or `ESC-H`; you can move one space right using the RIGHT ARROW key or `ESC-L`.
- You can move one word to the left with `ESC-B` and one word to the right with `ESC-W`.
- You can also do some editing: `BACKSPACE` deletes the character to the left of the cursor, `DELETE` deletes the character under the cursor, `CTRL-BACKSPACE` deletes the word to the left of the cursor, and `CTRL-DELETE` deletes the word under the cursor.

# UNIX System File Types

---

- **Viewing the Beginning or End of a File**
  - The head and tail commands are specifically designed for these jobs.
  - head shows you the beginning of a file, and tail shows you the end.
  - For example, the command shown here displays the first ten lines of transactions.
  - \$ head transactions
- and the following command displays the last ten lines:
- \$ tail transactions

# UNIX System File Types

- **Viewing the Beginning or End of a File**
- To display some other number, say the last three lines, you give head or tail a numerical argument.
- This command shows only the last three lines:
- \$ tail -3 transactions
- A useful feature of tail is the -f (f ollow) option.
- **This lets you use tail to check on the progress of a program that writes its output to a file.**
- Suppose a file transfer program is getting information from a remote system and putting it in the file newdata.
- In this example, tail displays the last three lines of newdata, waits (sleeps) for a short time, looks to see if there has been any new input, displays any new lines, and so on:
- \$ tail -3 -f newdata

# UNIX System File Types

---

- **Printing Files**
- The UNIX System includes a collection of programs, called the **lp** system, for printing files and documents.
- You can use it to print everything from simple text files to large documents with complex formats.
- It provides a simple, uniform interface to a wide variety of printers.
- The **lp** system is itself large and complex, but fortunately its complexity is well hidden from users.

# UNIX System File Types

---

- **Printing Files**
- In fact, three basic commands, lp, lpstat, and cancel, are all you need to know to use this system.
- (On Linux, these three basic commands have different names; they are lpr, lpq, and lprm, respectively)

# UNIX System File Types

---

- **Sending Output to the Printer**
- The basic command for printing a file is `lp` (line printer) (or on Linux, `lpr`). This command prints the file `research.nov` as follows:
- `$ lp research.nov`
  - request id is `lsr1-142` (1 file)
- You can print several files at once by including all of them in the arguments to `lp`. For instance,
- `$ lp res*`
  - request id is `lsr1-154` (3 files)

# UNIX System File Types

---

- **Specifying a Printer**
- The `lp` command does not ask you which printer to use.
- There may be several printers on the system, but one of them will be the system default.
- Unless you specify otherwise, this is the printer that `lp` uses.
- To find out which printers are available, you can ask your system administrator

# UNIX System File Types

---

- **Specifying a Printer**
- The `lp` command does not ask you which printer to use.
- There may be several printers on the system, but one of them will be the system default.
- Unless you specify otherwise, this is the printer that `lp` uses.
- To find out which printers are available, you can ask your system administrator

# UNIX System File Types

---

- **Specifying a Printer**
- To specify a particular printer, use the –d (destination) option, followed by the printer's name.
- For example,
- `$ lp -d laser2 flightinfo`
- sends `flightinfo` to the printer named `laser2`.

# UNIX System File Types

---

- **Print Spooling**
- When you print a file on the UNIX System, you do not have to wait until the file is printed (or until it is sent to the printer) before continuing with other work, and you do not have to wait until one print job is finished before sending another.
- `lp` spools its input to the UNIX print system, which means that it tells the print system what file to print and how to print it, and then leaves the work of getting the file through the printer to the system.

# UNIX System File Types

---

- **Print Spooling**
- Your job is submitted and spooled, but it is not printed at the precise time you enter the lp command, and lp does not automatically tell you when your job is actually finished.
- If you want to be notified when it is printed, use the -m (mail) option.
- For example,
- `$ lp -m -d laser2 flightinfo`
- sends you mail when your file is successfully printed.

# UNIX System File Types

---

- **Print Spooling**
- If you change the file between the time you issue the `lp` command and the time it actually goes to the printer, it is the changed file that will be printed.
- In particular, if you delete the file, or rename it, or move it to another directory the print system will not find it, and it will not be printed.
- To avoid this, use the `-c` (copy) option.
- The command
- `$ lp -c -d laser2 flightinfo`
- copies `flightinfo` to a temporary file in the print system and uses that copy as the input to the printer.
- Any changes you make to `flightinfo` after you issue this command will not appear in the printed output.

# UNIX System File Types

- **Using Ipstat to Monitor the Print System**
- The Ipstat command (on LINUX this is the Ipq command) provides a way to get this and other useful information, such as which printers are currently available on the system and how many other print jobs are scheduled.
- One of the most important uses of Ipstat is to see if your print jobs are being taken care of or if there is some problem with the system.
- The following shows that a job is scheduled for printing but has not yet started printing:
- \$ Ipstat

# UNIX System File Types

---

- **Canceling Print Jobs**
- cancel inkjet-133

# UNIX System File Types

Command	Use	Command	Use
<b>ls</b>	List the contents of a directory	<b>locate</b>	Search for files by name
<b>cat</b>	Display a short file	<b>find</b>	Find files
<b>touch</b>	Create an empty file	<b>chmod</b>	Change file permissions
<b>pwd</b>	Show the present directory	<b>umask</b>	Set default file permissions
<b>cd</b>	Change present directory	<b>chown</b>	Change the owner of a file or directory
<b>mv</b>	Move a file or directory	<b>chgrp</b>	Change the group of a file or directory
<b>cp</b>	Copy a file or directory	<b>pg, more, less</b>	Display a file
<b>ln</b>	Create a link	<b>head</b>	Display beginning of a file
<b>rm</b>	Remove a file or directory	<b>tail</b>	Display end of a file
<b>mkdir</b>	Make a directory	<b>lp/lpr</b>	Print a file
<b>rmdir</b>	Remove an empty directory	<b>lpstat/lpq</b>	Check status of a print job
<b>file</b>	Get file information	<b>cancel/lprm</b>	Cancel a print job



# A list of the commonly used variables in Linux

System Variable	Meaning	To View Variable Value Type
BASH_VERSION	Holds the version of this instance of bash.	echo \$BASH_VERSION
HOSTNAME	The name of the your computer.	echo \$HOSTNAME
CDPATH	The search path for the cd command.	echo \$CDPATH
HISTFILE	The name of the file in which command history is saved.	echo \$HISTFILE
HISTFILESIZE	The maximum number of lines contained in the history file.	echo \$HISTFILESIZE
HISTSIZE	The number of commands to remember in the command history. The default value is 500.	echo \$HISTSIZE
HOME	The home directory of the current user.	echo \$HOME



# A list of the commonly used variables in Linux

System Variable	Meaning	To View Variable Value Type
IFS	The Internal Field Separator that is used for word splitting after expansion and to split lines into words with the read builtin command. The default value is <space><tab><newline>.	echo \$IFS
LANG	Used to determine the locale category for any category not specifically selected with a variable starting with LC_.	echo \$LANG



# A list of the commonly used variables in Linux

System Variable	Meaning	To View Variable Value Type
PATH	The search path for commands. It is a colon-separated list of directories in which the shell looks for commands.	echo \$PATH
PS1	Your prompt settings.	echo \$PS1



# A list of the commonly used variables in Linux

System Variable	Meaning	To View Variable Value Type
TMOUT	The default timeout for the read builtin command. Also in an interactive shell, the value is interpreted as the number of seconds to wait for input after issuing the command. If not input provided it will logout user.	echo \$TMOUT
TERM	Your login terminal type.	echo \$TERM export TERM=vt100



# A list of the commonly used variables in Linux

System Variable	Meaning	To View Variable Value Type
SHELL	Set path to login shell.	echo \$SHELL
DISPLAY	Set X display name	echo \$DISPLAY export DISPLAY=:0.1
EDITOR	Set name of default text editor.	export EDITOR=/usr/bin/vim



# A list of the commonly used variables in Linux

```
eswaribala@DESKTOP-55AGI0I:~$ eswaribala@DESKTOP-55AGI0I:~$ eswaribala@DESKTOP-55AGI0I:~$ env > env.txt eswaribala@DESKTOP-55AGI0I:~$ ls app.txt application.properties demo.txt env.txt files sample.txt software virtusatraining2020 eswaribala@DESKTOP-55AGI0I:~$ cat env.txt SHELL=/bin/bash WSL_DISTRO_NAME=Ubuntu-20.04 NAME=DESKTOP-55AGI0I PWD=/home/eswaribala LOGNAME=eswaribala HOME=/home/eswaribala LANG=C.UTF-8 LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33:01:cd=40;33:01:or=40;31:01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.ASF=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xdw=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36: LESSCLOSE=/usr/bin/lesspipe %s %s TERM=xterm-256color LESSOPEN=| /usr/bin/lesspipe %s USER=eswaribala DISPLAY=:0.0 SHLVL=1 WSLENV= XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
```

# What is SUID and SGID?

---

- There are 3 special permission that are available for executable files and directories. These are :
- 1. SUID permission
- 2. SGID permission
- 3. Sticky bit

# What is SUID and SGID?

- **Set-user Identification (SUID)**
- Have you ever thought, how a non-root user can change his own password when he does not have write permission to the /etc/shadow file.
- Well to understand the trick check for the permission of /usr/bin/passwd command :
- # ls -lrt /usr/bin/passwd

```
eswaribala@DESKTOP-55AGI0I: ~
eswaribala@DESKTOP-55AGI0I:~$ find directory -perm /etc
find: invalid mode '/etc'
eswaribala@DESKTOP-55AGI0I:~$ ls -lrt /usr/bin/passwd
-rwsr-xr-x 1 root root 68208 May 28 12:07 /usr/bin/passwd
eswaribala@DESKTOP-55AGI0I:~$
```

# What is SUID and SGID?

- **Set-user Identification (SUID)**
- If you check carefully, you would find the S's in the permission field.
- The first s stands for the SUID
- – When a command or script with SUID bit set is run, its effective UID becomes that of the owner of the file, rather than of the user who is running it.

```
eswaribala@DESKTOP-55AGI0I: ~$ find directory -perm /etc  
find: invalid mode '/etc'  
eswaribala@DESKTOP-55AGI0I: ~$ ls -lrt /usr/bin/passwd  
-rwsr-xr-x 1 root root 68208 May 28 12:07 /usr/bin/passwd  
eswaribala@DESKTOP-55AGI0I: ~$
```

# What is SUID and SGID?

---

- **Set-user Identification (SUID)**
- How to set SUID on a file?
- # chmod 4555 [path\_to\_file]
- If a capital “S” appears in the owner’s execute field, it indicates that the setuid bit is on, and the execute bit “x” for the owner of the file is off or denied.

# What is SUID and SGID?

- **Set-group identification (SGID)**
- SGID permission on executable file
- – SGID permission is similar to the SUID permission, only difference is – when the script or command with SGID on is run, it runs as if it were a member of the same group in which the file is a member.
- # ls -l /usr/bin/write

```
eswaribala@DESKTOP-55AGI0I:~$ ls -l /usr/bin/write
lrwxrwxrwx 1 root root 23 Apr 23 12:12 /usr/bin/write -> /etc/alternatives/write
eswaribala@DESKTOP-55AGI0I:~$
```

# What is SUID and SGID?

---

- How to set G UID on a file?
- # chmod 2555 [path\_to\_file]

# What is SUID and SGID?

---

- **SGID on a directory**
- – When SGID permission is set on a directory, files created in the directory belong to the group of which the directory is a member.
- – For example if a user having write permission in the directory creates a file there, that file is a member of the same group as the directory and not the user's group.
- – This is very useful in creating shared directories.
- How to set SGID on a directory
- # chmod g+s [path\_to\_directory]

## What is a profile file?

---

- A profile file is a start-up file of an UNIX user, like the autoexec.bat file of DOS.
- When a UNIX user tries to login to his account, the operating system executes a lot of system files to set up the user account before returning the prompt to the user.
- In addition to the system settings, the user might wish to have some specific settings for his own account.
- To achieve this in UNIX, at the end of the login process, the operating system executes a file at the user level, if present. This file is called profile file.
- The name of the profile file varies depending on the default shell of the user.
- The profile file, if present, should always be in the home directory of the user.

# What is a profile file?

---

- The following are the profile files of the commonly used shells:
- The specific settings which an unix user usually does is:
  - Setting of any environment variable
  - Setting of any alias.(Though it is always recommended to keep the aliases in a separate file).
  - Setting of PATH variable or any other path variables.

Shell	Profile File
Ksh	.profile
Bourne	.profile
Bash	.bash_profile
Tcsh	.login
Csh	.login

A typical ksh profile file will look as shown below:  
`#cat $HOME/.profile`

## Setting the Terminal Type:

- \$ echo \$TERM
- echo \$SHELL
- export TERM=xterm-256color
- echo \$LS\_COLORS

Shell	Command
csh or tcsh	setenv TERM vt100
sh	TERM=vt100; export TERM
ksh, bash, or zsh	export TERM=vt100

# Java Installation

```
sudo apt update
java -version
```

```
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 12.475/17.194/22.223/3.985 ms
eswaribala@DESKTOP-55AGI0I:~$ sudo apt update
[sudo] password for eswaribala:
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
eswaribala@DESKTOP-55AGI0I:~$ java -version

Command 'java' not found, but can be installed with:

sudo apt install openjdk-11-jre-headless  # version 11.0.8+10-0ubuntu1~20.04, or
sudo apt install default-jre             # version 2:1.11-72
sudo apt install openjdk-13-jre-headless # version 13.0.3+3-1ubuntu2
sudo apt install openjdk-14-jre-headless # version 14.0.1+7-1ubuntu1
sudo apt install openjdk-8-jre-headless  # version 8u252-b09-1ubuntu1

eswaribala@DESKTOP-55AGI0I:~$
```

# Java Installation

**sudo apt install openjdk-8-jre-headless**

```
eswaribala@DESKTOP-55AGI0I:~$ sudo apt install openjdk-8-jre-headless
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java java-common libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libjpeg-turbo8 libjpeg8 liblcms2-2
  libnspr4 libnss3 libpcsselite1
Suggested packages:
  default-jre cups-common liblcms2-utils pcscd libnss-mdns fonts-dejavu-extra fonts-ipafont-gothic fonts-ipafont-mincho
  fonts-wqy-microhei fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  ca-certificates-java java-common libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libjpeg-turbo8 libjpeg8 liblcms2-2
  libnspr4 libnss3 libpcsselite1 openjdk-8-jre-headless
0 upgraded, 13 newly installed, 0 to remove and 0 not upgraded.
Need to get 29.4 MB of archives.
After this operation, 107 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 java-common all 0.72 [6816 B]
Get:2 http://archive.ubuntu.com/ubuntu focal/main amd64 libavahi-common-data amd64 0.7-4ubuntu7 [21.4 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal/main amd64 libavahi-common3 amd64 0.7-4ubuntu7 [21.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal/main amd64 libavahi-client3 amd64 0.7-4ubuntu7 [25.5 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 libcups2 amd64 2.3.1-9ubuntu1.1 [233 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/main amd64 liblcms2-2 amd64 2.9-4 [140 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 libjpeg-turbo8 amd64 2.0.3-0ubuntu1.20.04.1 [117 kB]
Get:8 http://archive.ubuntu.com/ubuntu focal/main amd64 libjpeg8 amd64 8c-2ubuntu8 [2194 B]
Get:9 http://archive.ubuntu.com/ubuntu focal/main amd64 libnspr4 amd64 2:4.25-1 [107 kB]
14% [9 libnspr4 71.6 kB/107 kB 67%] 17.7 kB/s 27min 6s
```



Type here to search



19:43  
04/08/2020

# Java Installation

**sudo apt install openjdk-8-jre-headless**

```
eswaribala@DESKTOP-55AGI0I:~$ ls /  
bin  boot  dev  etc  home  init  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var  
eswaribala@DESKTOP-55AGI0I:~$ java -version  
openjdk version "1.8.0_252"  
OpenJDK Runtime Environment (build 1.8.0_252-8u252-b09-1ubuntu1-b09)  
OpenJDK 64-Bit Server VM (build 25.252-b09, mixed mode)  
eswaribala@DESKTOP-55AGI0I:~$
```

# Java Installation

---

- `sudo apt install openjdk-8-jdk-headless`

# Installing MySQL in Ubuntu|Linux|Windows Subsystem for Linux from Scratch



- sudo apt-get remove --purge \*mysql\*
- sudo rm -rf /etc/mysql /var/lib/mysql
- sudo apt-get remove --purge \*mariadb\*
- sudo apt-get autoremove
- sudo apt-get autoclean
- dpkg -l | grep mariadb
- dpkg -l | grep mysql

# Installing MySQL in Ubuntu|Linux|Windows Subsystem for Linux from Scratch



- Now to fix all broken repositories
- sudo apt-get install -f
- Now Upgrading the Repositories
- sudo apt update
- sudo apt upgrade
- Now, to install MySQL 8, simply type:
- sudo apt install mysql-server

# Installing MySQL in Ubuntu|Linux|Windows Subsystem for Linux from Scratch



- Now to fix all broken repositories
- sudo apt-get install -f
- Now Upgrading the Repositories
- sudo apt update
- sudo apt upgrade
- Now, to install MySQL 8, simply type:
- sudo apt install mysql-server
- sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
- (change port)

# Installing MySQL in Ubuntu|Linux|Windows Subsystem for Linux from Scratch



- sudo service mysql start
- sudo mysql\_secure\_installation
- mysql -u root -p

```
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
eswaribala@DESKTOP-55AGI0I:~/Ecommerce$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 8.0.21-0ubuntu0.20.04.4 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ■
```

## Uninstall packages

---

- sudo apt list –installed
- sudo apt remove xfce4
- sudo apt-get remove --auto-remove ubuntu-gnome-desktop
- sudo apt-get purge --auto-remove ubuntu-gnome-desktop

# Standard Input and Output

- The command gets its input through the channel labeled “standard input.”
- That input can come from your keyboard (the default), a file, or a command.
- Similarly, the command delivers its output through the channel labeled “standard output.”
- The output might go to your screen (the default), a file, or another command.



# Standard Input and Output

---

- The command doesn't need to know which of these sources the input comes from, or where the output goes.
- It is the shell that sets up these connections, according to the instructions in your command line.
- It does this through the I/O redirection mechanisms, which include pipes and file redirection.

# Standard Input and Output

---

- A typical use of the pipe feature is the following command:
- \$ man find | lp
- This uses a pipe to send the output from the man command to the lp command, in order to print a hard copy of the manual page for find.
- An example of file redirection is the following command:
- \$ man find > temp
- This saves the output from the man command as the file temp

# Shell Redirection Output

Symbol	Example	Function
	<code>cmd1   cmd2</code>	Run <b>cmd1</b> and send output to <b>cmd2</b>
>	<code>cmd &gt; file</code>	Send output of <b>cmd</b> to <i>file</i>
>>	<code>cmd &gt;&gt; file</code>	Append output of <b>cmd</b> to <i>file</i>
<	<code>cmd &lt; file</code>	Take input for <b>cmd</b> from file
/dev/stdin	<code>cmd /dev/stdin</code>	Take input from keyboard
2>	<code>cmd 2&gt; errorfile</code>	Send standard error to errorfile ( <b>ksh</b> , <b>bash</b> )
2>&1	<code>cmd &gt; msgs 2&gt;&amp;1</code>	Send both output and standard error to <i>msgs</i> ( <b>sh</b> , <b>ksh</b> , and <b>bash</b> )
/dev/tty	<code>(cmd &gt; /dev/tty) &gt;&amp; error</code>	Redirect output to screen, and error to error ( <b>csh</b> , <b>tcsh</b> , and <b>bash</b> )
>&	<code>cmd &gt;&amp; msgs</code>	Send both output and errors to <i>msgs</i> ( <b>csh</b> , <b>tcsh</b> , and <b>bash</b> )

# Using Pipes

---

- The pipe symbol (|) tells the shell to take the standard output of one command and use it as the standard input of another command.
- Using pipes to join individual commands together in pipelines is an easy way to use a sequence of simple commands to carry out a complex task.
- For example, suppose you want to know if the user named “Ashok” is logged in.
- One way to find out would be to use the who command to list all of the users currently logged in, and to look for a line listing “Ashok” in the output.
- However, on a large system there could be many users—enough to make it difficult to find a specific name in the list.

# Using Pipes

```
eswaribala@DESKTOP-55AGI0I:~$ ls -l /etc/passwd | more
-rw-r--r-- 1 root root 3334 Aug  6 22:54 /etc/passwd
eswaribala@DESKTOP-55AGI0I:~$ sudo nano names.txt
eswaribala@DESKTOP-55AGI0I:~$ cat names.txt
Parameswari
Bala
Vignesh
Shyam
Akshaya
Deepak
Jhansi
eswaribala@DESKTOP-55AGI0I:~$ sort names.txt | uniq
Akshaya
Bala
Deepak
Jhansi
Parameswari
Shyam
Vignesh
eswaribala@DESKTOP-55AGI0I:~$ sudo nano names.txt
eswaribala@DESKTOP-55AGI0I:~$ sort names.txt | uniq
Akshaya
Bala
Deepak
Jhansi
Parameswari
Shyam
Vignesh
eswaribala@DESKTOP-55AGI0I:~$
```



# Using Pipes

```
eswaribala@DESKTOP-55AGI0I:~/www$ cat newline.html | head -5 | tail -7
<form class="form" [formGroup]="regGroup" (submit)="register()" autocomplete="off">
  <fieldset>
    <legend>User Registration</legend>
    <mat-form-field>
      <mat-placeholder>
        <mat-placeholder>
eswaribala@DESKTOP-55AGI0I:~/www$ ls -l | find ./ -type f -name "*.txt"
./env.txt
eswaribala@DESKTOP-55AGI0I:~/www$ ls
env.txt messages_fr_FR.properties messages_hi_IN.properties newline.html test.log
eswaribala@DESKTOP-55AGI0I:~/www$ ■
```

# Using redirection

```
eswaribala@DESKTOP-55AGI0I: ~
-rw-r--r-- 1 root root      477 Oct  7 2019 zsh_command_not_found
eswaribala@DESKTOP-55AGI0I:~$ sudo nano etcfilelist.txt
eswaribala@DESKTOP-55AGI0I:~$ cat names.txt
Parameswari
Bala
Vignesh
Shyam
Akshaya
Deepak
Jhansi
Bala
Jhansi
Vignesh
eswaribala@DESKTOP-55AGI0I:~$ sort < names.txt > sortednames.txt
eswaribala@DESKTOP-55AGI0I:~$ cat sortednames.txt
Akshaya
Bala
Bala
Deepak
Jhansi
Jhansi
Parameswari
Shyam
Vignesh
Vignesh
eswaribala@DESKTOP-55AGI0I:~$
```

● Unix Training - 5th to ... 36:33 —

HM



# Using redirection

```
eswaribala@DESKTOP-55AGI0I:~$ cat /dev/stdin > writedata.log
Parameswari
Himaja
Harisha
Shobhana
^C
eswaribala@DESKTOP-55AGI0I:~$ ls
Data.txt          db.properties    names.txt      test1.conf    test9999.conf      virtusatraining2020.bak
Ecommerce         demo.txt        sample.txt    test2.conf    testa.conf
app.txt          env.txt        sampledir     test3.conf    testd.conf
application.properties etcfilelist.txt software     test4.conf    testv.conf
bankdomain       files          sortednames.txt test709.conf virtusa.2020-08-06_081511
catalina.log     mariadb.conf   test.conf     test899.conf virtusatraining2020
eswaribala@DESKTOP-55AGI0I:~$ cat writedata.log
Parameswari
Himaja
Harisha
Shobhana
eswaribala@DESKTOP-55AGI0I:~$ -
```

>) to write standard input to a file named 'writedata.log'. The output contains names: Parameswari, Himaja, Harisha, Shobhana. The terminal window has a red border. The desktop background is dark, and the taskbar at the bottom shows various application icons like Microsoft Edge, Google Chrome, File Explorer, and others." data-bbox="0 0 1000 1000"/>

# How to use grep command in Unix

---

- The grep utility searches text file.txt for a pattern and prints all lines that contain that pattern.
- Syntax: grep [ -options ] limited-regular-expression [filename ... ]

# How to use grep command in Unix

---

- **grep options or grep command options**

- c Print only a count of the lines that contain the pattern.
- i Ignore upper/lower case distinction during comparisons.
- l Print only the names of file.txt with matching lines, separated by NEWLINE characters. Does not repeat the names of file.txt when the pattern is found more than once.
- n Precede each line by its line number in the file (first line is 1).
- v Print all lines except those that contain the pattern.
- r It recursively search the pattern in all the file.txt in the current directory and all it's sub-directory.
- w It searches the exact word

# How to use grep command in Unix

---

- **To find all uses of the word “spring” (in any case) in the multiples file like a\*, and write with line numbers:**
  - grep -i -n spring a\*
- search ‘application.properties’ for ‘Trading’ anywhere in a line
  - grep Trading application.properties

# How to use grep command in Unix

---

- **grep case insensitive command .**
- By default grep command is case sensitive.
- You can use option grep -i to make it case insensitive.
- We can use grep -w option for searching the specific word not sub-string .
- The below example searches adpatch.log for word failure in any case
- grep -i -w files demo.txt

# How to use grep command in Unix

---

- **find ‘run time’ or ‘run-time’ in all txt in file.txt**
- grep run[- ]time \*.txt
- **pipe who to grep, look for appmmgr**
- cat /etc/passwd | grep mohan
- **grep recursive option .It search for oracle string in current directory files and all the files in sub directory**
- grep -r spring \*

# How to use grep command in Unix

---

- **Grep exclude option (grep -v) . We can use grep -v to exclude the search item item. It will not show the lines which has oracle string in it**
- ps -ef | grep -v eswarib+

# Understanding Regular Expressions

## Understanding Regular Expressions:

^ (Caret)      match expression at the start of a line, as in ^A.

\$                match expression at the end of a line, as in A\$.  
(Question)

\ (Back  
Slash)        turn off the special meaning of the next character, as in \^. To look for a Caret “^”  
at the start of a line, the expression is ^\^.

[ ]              match any one of the enclosed characters, as in [aeiou]. Use Hyphen “-” for a  
(Brackets)      range, as in [0-9].

[^ ]             match any one character except those enclosed in [ ], as in [^0-9].

. (Period)      match a single character of any value, except end of line. So b.b will match  
“bob”, “bib”, “b-b”, etc.

\*                match zero or more of the preceding character or expression. An asterisk  
(Asterisk)      matches zero or more of what precedes it. Thus [A-Z]\* matches any number of  
upper-case letters, including none, while [A-Z][A-Z]\* matches one or more  
upper-case letters.

# Understanding Regular Expressions

---

- **search file.txt for lines with ‘kite’**
  - grep kite file.txt
- **‘kite’ at the start of a line**
  - grep '^kite' file.txt
- **‘kite’ at the end of a line**
  - grep 'kite\$'
- **lines containing only ‘kite’**
  - grep '^kite\$'

# Understanding Regular Expressions

---

- **lines starting with '^s', "\\" escapes the ^**
- grep '\^s' file.txt
- **search for 'kite' or 'Kite'**
- grep '[Kk]ite' file.txt
- **search for TOM, Tom, TOm or ToM**
- grep 'T[oO][mM]' file.txt
- **search for blank lines**
- grep '^'

# Understanding Regular Expressions

---

- **search for pairs of numeric digits**
- grep '[0-9][0-9]' file
- **list your mail**
- grep '^From: ' /usr/mail/\$USER
- **any line with at least one letter**
- grep '[a-zA-Z]' 1.txt
- **anything not a letter or number**
- grep '[^a-zA-Z0-9]'

# Understanding Regular Expressions

---

- **line start with “.” and 2 lower case letters} letters**
- grep '^\.?[a-z][a-z]'
- If you want to search multiple words in the same grep command ,then use egrep command in UNIX
- **It search all the three words in the file**
- egrep -n -v 'Bala|Vignesh' names.txt
- **It discarded all the lines having any of these three word from the output of ps -ef**
- cat /etc/passwd | egrep -n -v 'eswaribala|vigneshbala|mohan'

# Understanding Regular Expressions

---

- **grep with pipe command**
- pipe command in Linux let u input the output of the one command to the another command.
- **Example**
- ps -ef|grep python

# Understanding Regular Expressions

- Sometimes we just want the grep to show out only the file names which matched the given pattern then we use the -l (lower-case L) option. if multiple files are there. This will simply print all the file names
- grep -l -r spring \*
- Suppose you want to count that how many lines matches the given pattern/string, then use the option -c
- cat /etc/passwd | egrep -c -n -v 'eswaribala|vigneshbala|mohan'

# Understanding Regular Expressions

- When you are searching error using grep on a huge file, it may be useful to see some lines around the match.
- Lines before the match
- grep -A 10 "TOM" 1.txt
- Lines after the match
- grep -B 10 "TOM" 1.txt
- Lines around the match
- grep -C 10 "TOM" 1.txt

# Understanding Regular Expressions

---

- When we want to show the line number of the matched pattern with in the file.we can use grep -n
- grep -n "ORA-0600" alert.log
- Grep exclude directory in recursive search. Some time we want to exclude one directory from grep recursive search
- grep -r --exclude-dir=log "TOM"

# Using Pipes

---

- A better solution is to use a pipe to redirect the output of who to grep.
- The grep command searches through its input and prints the lines that match a target pattern.
- `$ ls -l | grep application`

# Using Pipes

---

- **Output Redirection to a File**
- The `>` redirection operator sends the output of a command to a file.
- For example,
- `$ ls -l > filelist`
- causes the shell to save the output of `ls -l` as the file `filelist`

# Using Pipes

---

- **Output Redirection to a File**
- The `>>` operator appends data to a file without overwriting it.
- That means that if the file already exists, the new data will be added on to the end.
- In this example,
- `$ cat notes.jan >> research`
- the shell redirects the standard output from `cat` and appends it to the file named `research`.

# Using Pipes

---

- **Input Redirection from a File**
- Just as you can use the greater than (or right arrow) symbol, `>`, to redirect standard output, you can use the less than (or left arrow) symbol, `<`, to redirect standard input.
- The `<` symbol tells the shell to interpret the filename that follows it as the standard input to a command.

## Using Pipes

---

- You can use the < redirection operator to replace the keyboard with a file as the standard input.
- In this case, the mail command will send the contents of that file, instead of waiting for you to enter a message at the keyboard:
- \$ mail anita@bio.ca.edu < note
- The < tells the shell to run mail with the contents of note as its standard input.

# Using Pipes

---

- **You can redirect input and output at the same time.**
- The following example uses the sort command to take the information in source, alphabetize it, and put the output in dest:
- `$ sort < source > dest`
- The order in which you indicate the input and output files doesn't matter, so the following example has the same result:
- `$ sort > dest < source`

# Using Pipes

---

- **Standard Input from the Keyboard**
- You can also force commands to accept input from the keyboard.
- The logical filename /dev/stdin refers to standard input.
- When used as an argument to a command, it causes the shell to send the input from the keyboard to the command, in place of a file.

# Using Pipes

---

- **Standard Input from the Keyboard**
- For example, the command sort is used to alphabetize the lines in a file.
- If you give it the argument /dev/stdin, it will sort the lines you type in from the keyboard:
- `$ sort /dev/stdin > guestlist`

# Unix Process Management

---

- A process refers to a program in execution; it's a running instance of a program.
- It is made up of the program instruction, data read from files, other programs or input from a system user.

# Unix Process Management

---

- **Types of Processes**
- There are fundamentally two types of processes in Linux:
  - **Foreground processes** (also referred to as interactive processes) – these are initialized and controlled through a terminal session.
  - In other words, there has to be a user connected to the system to start such processes; they haven't started automatically as part of the system functions/services.
  - **Background processes** (also referred to as non-interactive/automatic processes) – are processes not connected to a terminal; they don't expect any user input.

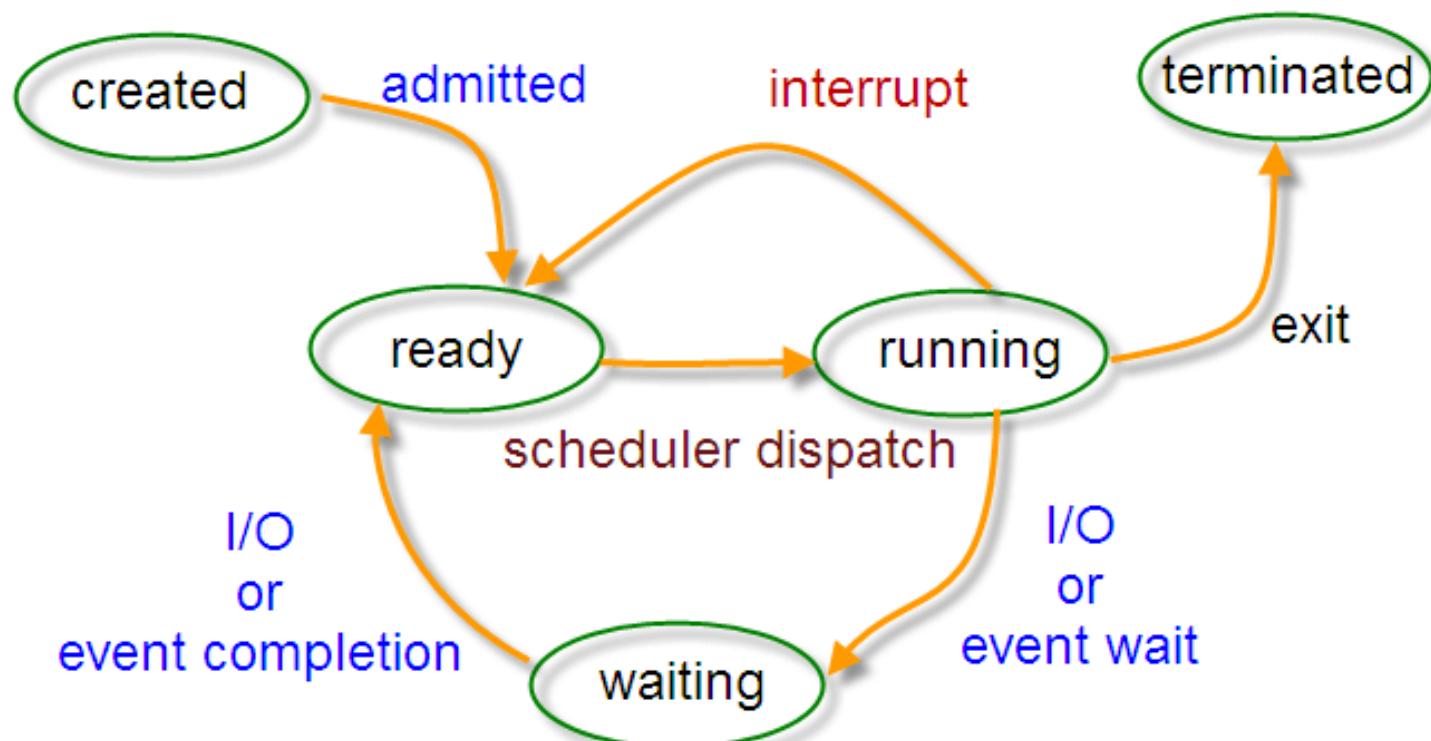
# What is Daemons

---

- These are special types of background processes that start at system startup and keep running forever as a service; they don't die.
- They are started as system tasks (run as services), spontaneously.
- However, they can be controlled by a user via the init process.

## Process States

### Process State



# Creation of a Processes in Linux

---

- A new process is normally created when an existing process makes an exact copy of itself in memory.
- The child process will have the same environment as its parent, but only the process ID number is different.
- There are two conventional ways used for creating a new process in Linux:
- Using The System() Function – this method is relatively simple, however, it's inefficient and has significantly certain security risks.
- Using fork() and exec() Function – this technique is a little advanced but offers greater flexibility, speed, together with security.
- Java process Example

# How Does Linux Identify Processes?

---

- Because Linux is a multi-user system, meaning different users can be running various programs on the system, each running instance of a program must be identified uniquely by the kernel.
- And a program is identified by its process ID (PID) as well as it's parent processes ID (PPID), therefore processes can further be categorized into:
  - Parent processes – these are processes that create other processes during run-time.
  - Child processes – these processes are created by other processes during run-time.

# The Init Process

---

- Init process is the mother (parent) of all processes on the system.
- it's the first program that is executed when the Linux system boots up;
- it manages all other processes on the system.
- It is started by the kernel itself, so in principle it does not have a parent process.
- The init process always has process ID of 1. It functions as an adoptive parent for all orphaned processes.
- You can use the pidof command to find the ID of a process:

# The Init Process

---

- To find the process ID and parent process ID of the current shell, run:
  - `$ echo $$`
  - `$ echo $PPID`

# Starting a Process in Linux

---

- Once you run a command or program (for example `ls -l *.txt`), it will start a process in the system.
- You can start a foreground (interactive) process as follows, it will be connected to the terminal and a user can send input to it:

# Linux Background Jobs

---

- To start a process in the background (non-interactive), use the & symbol, here, the process doesn't read input from a user until it's moved to the foreground.
- ls -l \*.txt &
- You can also send a process to the background by suspending it using [Ctrl + Z], this will send the SIGSTOP signal to the process, thus stopping its operations; it becomes idle:

# Linux Background Jobs

---

- To continue running the above-suspended command in the background, use the bg command:
- # bg
- To send a background process to the foreground, use the fg command together with the job ID like so:
- # jobs
- # fg %1

# States of a Process in Linux

---

- During execution, a process changes from one state to another depending on its environment/circumstances.
- In Linux, a process has the following possible states:
- **Running** – here it's either running (it is the current process in the system) or it's ready to run (it's waiting to be assigned to one of the CPUs).
- **Waiting** – in this state, a process is waiting for an event to occur or for a system resource. Additionally, the kernel also differentiates between two types of waiting processes; interruptible waiting processes – can be interrupted by signals and uninterruptible waiting processes – are waiting directly on hardware conditions and cannot be interrupted by any event/signal.

# States of a Process in Linux

---

- **Stopped** – in this state, a process has been stopped, usually by receiving a signal. For instance, a process that is being debugged.
- **Zombie** – here, a process is dead, it has been halted but it's still has an entry in the process table.

# How to View Active Processes in Linux

---

- ps Command
- top – System Monitoring Tool
- glances – System Monitoring Tool
  - glances is a relatively new system monitoring tool with advanced features

# How to View Active Processes in Linux

```

eswaribala@DESKTOP-55AGI0I: ~
DESKTOP-55AGI0I - IP 192.168.43.50/24 Pub 2401:4900:234f:6b02:802b:bd4:40fb:f97b                                         Uptime: 4:31:25
CPU [ | 17.7%] CPU    17.7% nice: 0.0% ctx_sw: 0 MEM    49.7% active: 164M SWAP   1.0% LOAD   8-core
MEM [ || 49.7%] user: 8.0% irq: 0.0% inter: 0 total: 15.8G inactive: 154M total: 27.9G 1 min: 0.52
SWAP [      1.0%] system: 7.0% iowait: 0.0% sw_int: 0 used: 7.87G buffers: 33.2M used: 274M 5 min: 0.58
                  idle: 36.0% steal: 0.0% free: 7.97G cached: 191M free: 27.6G 15 min: 0.59

NETWORK
eth2          Rx/s Tx/s  TASKS  7 (65 thr), 1 run, 6 slp, 0 oth sorted automatically by CPU consumption
eth3          0b   0b
eth3          0b   0b  CPU%  MEM%  VIRT   RES   PID USER
lo            0b   0b  19.7  0.3   440M  47.3M  8784 eswaribal
wifi0         0b   0b  0.0   0.0   17.8M  3.37M   66 eswaribal
DefaultGateway 39ms 0.0   0.0   1.91G  2.10M  8140 www-data
                  0.0   0.0   1.91G  2.09M  8138 www-data
                  0.0   0.0   8.72M  544K    65 root
                  0.0   0.0   8.68M  512K     1 root
TIME+ THR NI S R/s W/s Command
0:01 6   0 R ? ? /usr/bin/python3 /usr/bin/gla
0:01 1   0 S ? ? -bash
0:00 1   0 S ? ? /usr/sbin/apache2 -k start
0:00 27  0 S ? ? /usr/sbin/apache2 -k start
0:00 27  0 S ? ? /usr/sbin/apache2 -k start
0:00 1   0 S ? ? //init
0:00 2   0 S ? ? //init

```

2020-08-06 01:47:13 IST

# Sending Signals To Processes

---

- The fundamental way of controlling processes in Linux is by sending signals to them. There are multiple signals that you can send to a process, to view all the signals run:
- `$ kill -l`

## PS and PS Variables

---

- ps - report a snapshot of the current processes.
- This version of ps accepts several kinds of options:
  - UNIX options, which may be grouped and must be preceded by a dash.
  - BSD options, which may be grouped and must not be used with a dash.
  - GNU long options, which are preceded by two dashes.

# PS and PS Variables

---

- To see every process on the system using standard syntax:
- ps -e
- ps -ef
- ps -eF
- ps -ely

# PS and PS Variables

---

- To see every process on the system using BSD syntax:
  - ps ax
  - ps axu
- To print a process tree:
  - ps -ejH
  - ps axjf
- To get info about threads:
  - ps -eLf
  - ps axms

# PS and PS Variables

---

- To get security info:
  - ps -eo euser,ruser,suser,fuser,f,comm,label
  - ps axZ
  - ps -eM
- To see every process running as root (real & effective ID) in user format:
  - ps -U root -u root u
- To see every process with a user-defined format:
  - ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm
  - ps axo stat,euid,ruid,tty,tpgid,ses,pgrp,ppid,pid,pcpu,comm
  - ps -eopid,tt,user, fname,tmout,f,wchan

# PS and PS Variables

---

- Print only the process IDs of syslogd:
  - `ps -C syslogd -o pid=`
  - Print only the name of PID 42:
  - `ps -p 42 -o comm=`

# PS and PS Variables

- man ps

```
SIMPLE PROCESS SELECTION
a      Lift the BSD-style "only yourself" restriction, which is imposed upon the set of all processes when some BSD-style
       (without "-") options are used or when the ps personality setting is BSD-like. The set of processes selected in this
       manner is in addition to the set of processes selected by other means. An alternate description is that this option
       causes ps to list all processes with a terminal (tty), or to list all processes when used together with the x option.

-A     Select all processes. Identical to -e.

-a     Select all processes except both session leaders (see getsid\(2\)) and processes not associated with a terminal.

-d     Select all processes except session leaders.

--deselect
       Select all processes except those that fulfill the specified conditions (negates the selection). Identical to -N.

-e     Select all processes. Identical to -A.

g     Really all, even session leaders. This flag is obsolete and may be discontinued in a future release. It is normally
       implied by the a flag, and is only useful when operating in the sunos4 personality.

-N     Select all processes except those that fulfill the specified conditions (negates the selection). Identical to
       --deselect.

T     Select all processes associated with this terminal. Identical to the t option without any argument.

r     Restrict the selection to only running processes.

x     Lift the BSD-style "must have a tty" restriction, which is imposed upon the set of all processes when some BSD-style
       (without "--") options are used or when the ps personality setting is BSD-like. The set of processes selected in this
       manner is in addition to the set of processes selected by other means. An alternate description is that this option
       causes ps to list all processes owned by you (same EUID as ps), or to list all processes when used together with the
Manual page ps(1) line 80 (press h for help or q to quit)
```

# Traceroute Install

**sudo apt-get update && sudo apt-get install traceroute**

```
89 88 87
eswaribala@DESKTOP-55AGI0I:~$ sudo apt-get update && sudo apt-get install traceroute
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [149 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [52.6 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [3532 B]
Get:8 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [29.2 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-en [7732 B]
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [44.4 kB]
Get:11 http://security.ubuntu.com/ubuntu focal-security/universe Translation-en [23.6 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [316 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Metadata [1832 B]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/main Translation-en [119 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [8092 B]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [29.2 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [7732 B]
Get:18 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [146 kB]
Get:19 http://archive.ubuntu.com/ubuntu focal-updates/universe Translation-en [73.9 kB]
Get:20 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [4920 B]
Fetched 1333 kB in 15s (91.5 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  traceroute
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 45.4 kB of archives.
After this operation, 152 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 traceroute amd64 1:2.1.0-2 [45.4 kB]
```

# Traceroute Install

traceroute askubuntu.com

```
eswaribala@DESKTOP-55AGI0I:~$ traceroute askubuntu.com
traceroute to askubuntu.com (151.101.193.69), 30 hops max, 60 byte packets
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 * * *

eswaribala@DESKTOP-55AGI0I:~$ ping -c3 www.google.com
```

# Ifconfig

```
eswaribala@DESKTOP-55AGI0: ~
eswaribala@DESKTOP-55AGI0:~$ sudo apt-get install net-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libgnutls-openssl27 libgsasl7 libntlm0 msmtplib
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  net-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
2 not fully installed or removed.
Need to get 196 kB of archives.
After this operation, 864 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 net-tools amd64 1.60+git20180626.aebd88e-1ubuntu1 [196 kB]
Fetched 196 kB in 3s (61.5 kB/s)
Selecting previously unselected package net-tools.
(Reading database ... 117819 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20180626.aebd88e-1ubuntu1_amd64.deb ...
Unpacking net-tools (1.60+git20180626.aebd88e-1ubuntu1) ...
Setting up mysql-server-8.0 (8.0.21-0ubuntu0.20.04.4) ...
invoke-rc.d: could not determine current runlevel
  * Stopping MySQL database server mysqld                                         [ OK ]
Renaming removed key_buffer and myisam-recover options (if present)
Cannot open /proc/net/unix: No such file or directory
Cannot stat file /proc/1/fd/5: Operation not permitted
Cannot stat file /proc/1/fd/10: Operation not permitted
Cannot stat file /proc/1/fd/6: Operation not permitted
Cannot stat file /proc/9/fd/7: Operation not permitted
Cannot stat file /proc/9/fd/10: Operation not permitted
Cannot stat file /proc/9/fd/5: Operation not permitted
mysqld will log errors to /var/log/mysql/error.log
mysqld is running as pid 147
sleep: cannot read realtime clock: Invalid argument
```

# Ifconfig

eswaribala@DESKTOP-55AGI0I:~

```
eswaribala@DESKTOP-55AGI0I:~$ ifconfig
eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 169.254.131.111 netmask 255.255.0.0 broadcast 169.254.255.255
        inet6 fe80::58b1:d8ba:5c3c:836f prefixlen 64 scopeid 0xfd<compat,link,site,host>
            ether 02:00:4c:4f:4f:50 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.213.129 netmask 255.255.255.240 broadcast 192.168.213.143
        inet6 fe80::4daf:9aa0:a763:402 prefixlen 64 scopeid 0xfd<compat,link,site,host>
            ether 00:15:5d:61:d5:51 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 1500
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0xfe<compat,link,site,host>
            loop (Local Loopback)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wifi0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.8 netmask 255.255.255.0 broadcast 192.168.0.255
        inet6 fe80::bc62:acf0:4b32:24 prefixlen 64 scopeid 0xfd<compat,link,site,host>
            ether f8:34:41:ac:d4:6f (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
```



Type here to search

18:23  
ENG  
06/08/2020

# Ping

ping -c3 www.google.com

```
29 * * *
30 * * *
eswaribala@DESKTOP-55AGI0I:~$ ping -c3 www.google.com
PING www.google.com (216.58.197.68) 56(84) bytes of data.
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=1 ttl=120 time=22.2 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=2 ttl=120 time=12.5 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=3 ttl=120 time=16.9 ms

--- www.google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 12.475/17.194/22.223/3.985 ms
eswaribala@DESKTOP-55AGI0I:~$
```

## ftp utility

---

- FTP (File Transfer Protocol) is a relatively old and most used standard network protocol used for uploading/downloading files between two computers over a network.
- However, FTP by its original insecure, because it transmits data together with user credentials (username and password) without encryption.
- FTP

# ftp utility

```
eswaribala@DESKTOP-55AGI0I:~  
eswaribala@DESKTOP-55AGI0I:~$ ftp localhost  
Connected to localhost.  
220 Microsoft FTP Service  
Name (localhost:eswaribala): parameswari.ettiappan@outlook.com  
331 Password required  
Password:  
230 User logged in.  
Remote system type is Windows_NT.  
ftp> ls  
200 PORT command successful.  
125 Data connection already open; Transfer starting.  
01-21-20 07:36AM <DIR> aspnet_client  
07-18-18 08:06PM 54 Attendance.txt  
07-18-18 08:06PM 114 Data.txt  
07-18-18 08:06PM 114 DecData.txt  
07-18-18 08:06PM 175 EmployeeData.txt  
07-18-18 08:06PM 152 EncData.txt  
07-18-18 08:06PM 973 extractedPDFTextData.txt  
07-18-18 08:06PM 0 New Text Document.txt  
08-16-18 11:34AM 66448 OoPdfForm-Stephani.pdf  
07-18-18 08:06PM 279 publicData.txt  
07-18-18 08:06PM 114 publicdecData.txt  
08-16-18 11:40AM <DIR> RPAMockTest  
07-18-18 08:20PM <DIR> RPSAA  
226 Transfer complete.  
ftp> -
```

# ftp utility

```
eswaribala@DESKTOP-55AGI0I:~$ ftp localhost
Connected to localhost.
220 Microsoft FTP Service
Name (localhost:eswaribala): parameswari.ettiappan@outlook.com
331 Password required
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> ls
200 PORT command successful.
125 Data connection already open; Transfer starting.
01-21-20 07:36AM      <DIR>          aspnet_client
07-18-18 08:06PM          54 Attendance.txt
07-18-18 08:06PM          114 Data.txt
07-18-18 08:06PM          114 DecData.txt
07-18-18 08:06PM          175 EmployeeData.txt
07-18-18 08:06PM          152 EncData.txt
07-18-18 08:06PM          973 extractedPDFTextData.txt
07-18-18 08:06PM          0 New Text Document.txt
08-16-18 11:34AM        66448 OoPdfForm-Stephani.pdf
07-18-18 08:06PM          279 publicData.txt
07-18-18 08:06PM          114 publicdecData.txt
08-16-18 11:40AM      <DIR>          RPAMockTest
07-18-18 08:20PM      <DIR>          RPSAA
226 Transfer complete.
ftp> get Data.txt
local: Data.txt remote: Data.txt
200 PORT command successful.
125 Data connection already open; Transfer starting.
226 Transfer complete.
114 bytes received in 0.00 secs (784.0009 kB/s)
ftp>
```

# ftp utility

---

```
eswaribala@DESKTOP-55AGI0: ~
ftp> put app.txt
local: app.txt remote: app.txt
200 PORT command successful.
125 Data connection already open; Transfer starting.
226 Transfer complete.
9702 bytes sent in 0.00 secs (15.5767 MB/s)
ftp> ■
```

## ftp utility with remote service

---

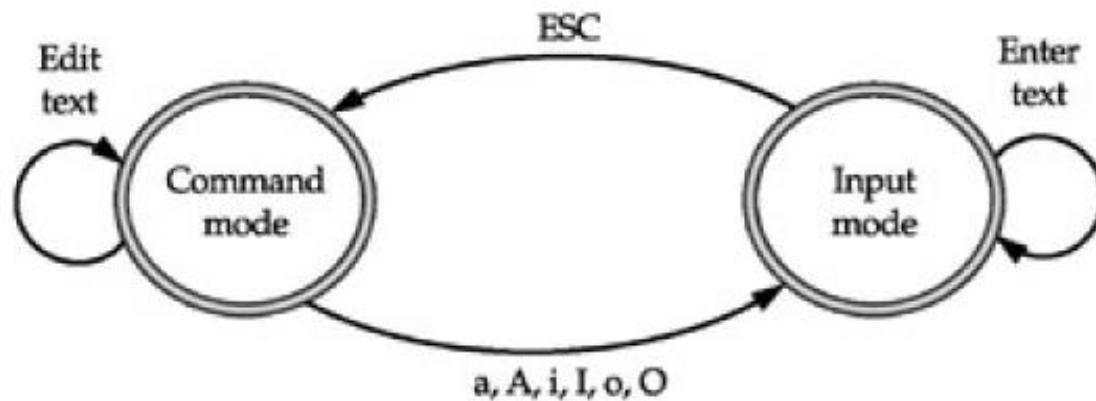
- FTP URL: ftp.dlptest.com or ftp://ftp.dlptest.com/
- FTP User: dlpuuser@dlptest.com
- Password: eUj8GeW55SvYaswqUyDSm5v6N

# ftp utility with remote service

```
eswaribala@DESKTOP-55AGI0I: ~
Not connected.
ftp> exit
eswaribala@DESKTOP-55AGI0I:~$ ftp  ftp.dlptest.com
Connected to ftp.dlptest.com.
220-#####
220-Please upload your web files to the public_html directory.
220-Note that letters are case sensitive.
220-#####
220 This is a private system - No anonymous login
Name (ftp.dlptest.com:eswaribala): dlpuser@dlptest.com
331 User dlpuser@dlptest.com OK. Password required
Password:
230-Your bandwidth usage is restricted
230 OK. Current restricted directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful
150 Connecting to port 59058
drwxr-xr-x    2 dlptest9   dlptest9      57344 Aug  7 01:08 .
drwxr-xr-x    2 dlptest9   dlptest9      57344 Aug  7 01:08 ..
-rw-r--r--    1 dlptest9   dlptest9     344648 Aug  7 01:00 1_4867950383094477124_17-9ULspeedtest.upt
-rw-r--r--    1 dlptest9   dlptest9    1056511 Aug  7 01:00 JSCLSQFP07X07P0048TH2000.zip
226-Options: -a -l
226 4 matches total
ftp> -
```

# Vi Editor

- Vi Editor
- Vi Cheat sheet



## Command Shell

---

- The original UNIX System shell, sh, was written by Steve Bourne, and as a result it is known as the Bourne shell.
- The C shell, csh, was the first attempt to enhance the original Bourne shell.
- The syntax was strongly influenced by the C programming language.
- The C shell introduced the concepts of a command history list, job control, and aliases.
- However, like sh, it lacks some important features of later shells.
- A common complaint about csh is that the new syntax is not compatible with the Bourne shell, and so some scripts may not work properly in csh.

## Command Shell

---

- The extended C shell, tcsh, has replaced csh entirely on some versions of UNIX (including Linux).
- It retains all the features of csh and adds command-line editing (a very important shell feature) and history completion.
- It is one of the more popular shells, although like csh it has been criticized for not being compatible with the Bourne shell.

## Command Shell

---

- The Korn shell, ksh, was developed at AT&T Bell Laboratories by David Korn.
- Unlike the C shell, the Korn shell has a syntax compatible with sh.
- Like tcsh, ksh includes a command history list, job control, aliases, and command-line editing.
- The Korn shell was until recently proprietary to AT&T, although it can now be downloaded for free.

## Command Shell

---

- The Bourne Again Shell, bash, is part of the GNU project.
- It extends ksh further, while remaining compatible with the original Bourne shell syntax, and adds a few features from tcsh as well.
- bash is the default shell in Linux and may be the most popular shell today On some systems, the command sh will run bash instead.

## Running the Shell

---

- When you log in to the system, a shell program is automatically started for you.
- This is your login shell.
- The particular shell program that is run when you log in is determined by your entry in the file /etc/passwd.
- This file contains information the system needs to know about each user, including name, login ID, and so forth.
- The last field of this file contains the name of the program to run as your shell.
- A typical entry might look like
- `$ grep username /etc/passwd`

# Running the Shell

---

terminal eswaribala@DESKTOP-55AGI0I: ~

```
eswaribala@DESKTOP-55AGI0I:~$ sudo grep eswaribala /etc/passwd
eswaribala:x:1000:1000:,,,:/home/eswaribala:/bin/bash
eswaribala@DESKTOP-55AGI0I:~$
```

# Changing Your Login Shell

---

- The command to change your login shell is chsh. In this example, the user rlf is changing her shell to
- bash:
- \$ chsh
- To find the full pathname for a command or executable, type which followed by the name of the command.
- For example,
- \$ which bash

# Changing Your Login Shell

⌚ eswaribala@DESKTOP-55AGI0I: ~

```
eswaribala@DESKTOP-55AGI0I:~$ sudo grep eswaribala /etc/passwd
eswaribala:x:1000:1000:,,,:/home/eswaribala:/bin/bash
eswaribala@DESKTOP-55AGI0I:~$ chsh
Password:
Changing the login shell for eswaribala
Enter the new value, or press ENTER for the default
      Login Shell [/bin/bash]:
eswaribala@DESKTOP-55AGI0I:~$ which bash
/usr/bin/bash
eswaribala@DESKTOP-55AGI0I:~$
```

## What the Shell Does

---

- After you log in, much of your interaction with the UNIX System takes the form of a dialog with the shell.
- The dialog follows this simple sequence repeated over and over:
  - 1. The shell prompts you when it is ready for input, and waits for you to enter a command.
  - 2. You enter a command by typing in a command line.
  - 3. The shell processes your command line to determine what actions to take and carries out the actions required.
  - 4. After the program is finished, the shell prompts you for input, beginning the cycle again.

# Shell Programming

---

- **Variables**
- Variables are "words" that hold a value.
- The shell enables you to create, assign, and delete variables.
- Although the shell manages some variables, it is mostly up to the programmer to manage variables in shell scripts.

# Shell Programming

---

- **Defining Variables**
- Variables are defined as follows:
- name=value
- In this example, name is the name of the variable, and value is the value it should hold.
- For example,
- FRUIT=peach
- defines the variable FRUIT and assigns it the value peach.
- Variables of this type are called scalar variables. A scalar variable can hold only one value at a time.

# Shell Programming

---

- **Defining Variables**
- Scalar variables are also referred to as name value pairs, because a variable's name and its value can be thought of as a pair.

# Shell Programming

---

- **Variable Names**
- The name of a variable can contain only letters ( a to z or A to Z), numbers ( 0 to 9) or the underscore character ( \_).
- In addition, a variable's name can start only with a letter or an underscore.
- **The following examples are valid variable names:**
- **\_FRUIT**
- **FRUIT\_BASKET**
- **TRUST\_NO\_1**
- **TWO\_TIMES\_2**
- **but**
- **2\_TIMES\_2\_EQUALS\_4**
- **is not a valid variable name**

# Shell Programming

- **To make this a valid name, add an underscore at the beginning of its name:**
- `_2_TIMES_2`
- Variable names, such as 1, 2 or 11, that start with numbers are reserved for use by the shell.
- You can use the value stored in these variables, but you cannot set the value yourself.
- The reason you cannot use other characters such as !, \*, or - is that these characters have a special meaning for the shell.
- If you try to make a variable name with one of these special characters it confuses the shell.

# Shell Programming

---

- **For example, the variable names**
- FRUIT-BASKET
- \_2\*2
- TRUST\_NO\_1!
- are invalid names.
- The error message generated by one of these variable name looks something like the following:
- \$ FRUIT-BASKET=apple
- /bin/sh: FRUIT-BASKET=apple: not found.

# Shell Programming

---

- **Variable Values**
- The shell enables you to store any value you want in a variable.
- For example,
  - FRUIT=peach
  - FRUIT=2apples
  - FRUIT=apple+pear+kiwi
- The one thing to be careful about is using values that have spaces. For example,
- **\$ FRUIT=apple orange plum.**
- results in the following error message:
- sh: orange: not found.

# Shell Programming

---

- **In order to use spaces you need to quote the value.**
- For example, both of the following are valid assignments:
- `$ FRUIT="apple orange plum"`
- `$ FRUIT='apple orange plum'`

# Shell Programming

---

- **Accessing Values**
- To access the value stored in a variable, prefix its name with the dollar sign ( \$).
- For example, the command
- `$ echo $FRUIT`
  - peach
- prints out the value stored in the variable FRUIT, in this case peach.
- If you do not use the dollar sign ( \$) to access the value of a variable, the name of the variable is printed instead of its value.
- For example,
- `$ echo FRUIT`
  - FRUIT

# Shell Programming

---

- **Accessing Values**
- The dollar sign ( \$) is used only to access a variable's value, not to define it.
- For example, the assignment
- `$ $FRUIT=apple`
- generates the following warning message
- `sh: peach=apple: not found`
- if FRUIT is defined as given previously.
- If the variable FRUIT is undefined the error would be
- `sh: =apple: not found`

# Shell Programming

---

- **Array Variables**
- The Bourne shell, sh, supports only scalar variables, which are the type of variables you have seen so far.
- The Korn shell, ksh, extends this to include array variables.
- Version 2.0 and later of the Bourne Again shell, bash, also support array variables.

# Shell Programming

---

- **Array Variables**
- The Bourne shell, sh, supports only scalar variables, which are the type of variables you have seen so far.
- The Korn shell, ksh, extends this to include array variables.
- Version 2.0 and later of the Bourne Again shell, bash, also support array variables.

# Shell Programming

---

- **Array Variables**
- The simplest method of creating an array variable is to assign a value to one of its indices.
- This is expressed as follows:
- `name[index]=value`
- Here `name` is the name of the array, `index` is the index of the item in the array that you want to set, and `value` is the value you want to set for that item.
- As an example, the following commands
  - `$ FRUIT[0]=apple`
  - `$ FRUIT[1]=banana`
  - `$ FRUIT[2]=orange`

# Shell Programming

---

- **Array Variables**
- set the values of the first three items in the array named FRUIT.
- You could do the same thing with scalar variables as follows:
  - \$ FRUIT\_0=apple
  - \$ FRUIT\_1=banana
  - \$ FRUIT\_2=orange

# Shell Programming

- **Array Variables**
- If an array variable with the same name as a scalar variable is defined, the value of the scalar variable becomes the value of the element of the array at index 0.
- For example, if the following commands are executed
  - `$ FRUIT=apple`
  - `$ FRUIT[1]=peach`
- the element `FRUIT` has the value `apple`.
- At this point any accesses to the scalar variable `FRUIT` are treated like an access to the array item `FRUIT[0]`.
- The second form of array initialization is used to set multiple elements at once. In ksh, this is done as follows:
  - `set -A name value1 value2 ... valuen`
- In bash, the multiple elements are set as follows:
- **myarray=([0]=derri [3]=gene [2]=mike [1]=terry)**

# Shell Programming

---

- **Array Variables**
- Here, name is the name of the array and the values, 1 to n, are the values of the items to be set.
- When setting multiple elements at once, both ksh and bash use consecutive array indices beginning at 0.
- For example the ksh command
  - `$ set -A band derri terry mike gene`
- or the bash command
  - `$ band=([0]=derri [3]=gene [2]=mike [1]=terry)`
  - is equivalent to the following commands:
    - `$ band[0]=derri`
    - `$ band[1]=terry`

# Shell Programming

---

- **Accessing Array Values**
- After you have set any array variable, you access it as follows:
  - \${name[index]}
- Here name is the name of the array, and index is the index that interests us.
- For example, if the array FRUIT was initialized as given previously, the command
  - \$ echo \${FRUIT[2]}
- produces the following output:
  - orange
- You can access all the items in an array in one of the following ways:
  - \${name[\*]}
  - \${name[@]}

# Shell Programming

---

- **Accessing Array Values**
- Here *name* is the name of the array you are interested in. If the `FRUIT` array is initialized as given previously, the command
  - `$ echo ${FRUIT[*]}`
  - produces the following output:
  - `apple banana orange`
  - If any of the array items hold values with spaces, this form of array access will not work and will need to use the second form.
  - The second form quotes all the array entries so that embedded spaces are preserved.
  - For example, define the following array item:
  - `FRUIT[3]="passion fruit"`

# Shell Programming

---

- **Accessing Array Values**
- Assuming that FRUIT is defined as given previously, accessing the entire array using the following command
- `$ echo ${FRUIT[*]}`
- results in five items, not four:
  - apple banana orange passion fruit
- Commands accessing FRUIT using this form of array access get five values, with passion and fruit treated as separate items.

# Shell Programming

---

- **Accessing Array Values**
- To get only four items, you have to use the following form:
- `$ echo ${FRUIT[@]}`
- The output from this command looks similar to the previous commands:
- apple banana orange passion fruit
- but the commands see only four items because the shell quotes the last item as passion fruit.

# Shell Programming

---

- **Read-only Variables**
- The shell provides a way to mark variables as read-only by using the readonly command.
- After a variable is marked read-only, its value cannot be changed.
- Consider the following commands:
- \$ FRUIT=kiwi
- \$ readonly FRUIT
- \$ echo \$FRUIT
- kiwi
- \$ FRUIT=cantaloupe
- The last command results in an error message:
- /bin/sh: FRUIT: This variable is read only.

# Shell Programming

---

- **Unsetting Variables**
- Unsetting a variable tells the shell to remove the variable from the list of variables that it tracks.
- This is like asking the shell to forget a piece of information because it is no longer required.
- Both scalar and array variables are unset using the `unset` command:
- `unset name`
- Here `name` is the name of the variable to unset.
- For example,
- `unset FRUIT`
- unsets the variable `FRUIT`.
- You cannot use the `unset` command to unset variables that are marked readonly.

# Shell Programming

---

- When a shell is running, three main types of variables are present:
- Local Variables
- Environment Variables
- Shell Variables

# Shell Programming

---

- A **local variable** is a variable that is present within the current instance of the shell.
- It is not available to programs that are started by the shell.
- The variables that you looked at previously have all been local variables.

# Shell Programming

---

- An **environment variable** is a variable that is available to any child process of the shell.
- Some programs need environment variables in order to function correctly.
- Usually a shell script defines only those environment variables that are needed by the programs that it runs..

# Shell Programming

---

- A **shell variable** is a special variable that is set by the shell and is required by the shell in order to function correctly.
- Some of these variables are environment variables whereas others are local variables.

# Shell Programming

---

- A **shell variable** is a special variable that is set by the shell and is required by the shell in order to function correctly.
- Some of these variables are environment variables whereas others are local variables.

# Shell Variables

Variable	Description
PWD	Indicates the current working directory as set by the <code>cd</code> command.
UID	Expands to the numeric user ID of the current user, initialized at shell startup.
SHLVL	Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in <code>exit</code> command ends the current session.
REPLY	Expands to the last input line read by the <code>read</code> built-in command when it is given no arguments. This variable is not available in <code>sh</code> .
RANDOM	Generates a random integer between 0 and 32,767 each time it is referenced. You can initialize the sequence of random numbers by assigning a value to <code>\$RANDOM</code> . If <code>\$RANDOM</code> is unset, it loses its special properties, even if it is subsequently reset. This variable is not available in <code>sh</code> .
SECONDS	Each time this parameter is referenced, it returns the number of seconds since shell invocation. If a value is assigned to <code>\$SECONDS</code> , the value returned on subsequent references is the number of seconds since the assignment plus the value assigned. If <code>\$SECONDS</code> is unset, it loses its special properties, even if it is subsequently reset. This variable is not available in <code>sh</code> .
IFS	Indicates the Internal Field Separator that is used by the parser for word splitting after expansion. <code>\$IFS</code> is also used to split lines into words with the <code>read</code> built-in command. The default value is the string, " <code>\t\n</code> ", where " " is the space character, <code>\t</code> is the tab character, and <code>\n</code> is the newline character.
PATH	Indicates search path for commands. It is a colon-separated list of directories in which the shell looks for commands. A common value is  <code>PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/ucb</code>
HOME	Indicates the home directory of the current user: the default argument for the <code>cd</code> built-in command.

# Command and Arithmetic Substitution

---

- Command substitution enables you to capture the output of a command and substitute it in another command, whereas arithmetic substitution enables you to perform simple integer mathematics using the shell.

# Shell Programming

---

- **Command Substitution**
- Command substitution is the mechanism by which the shell performs a given set of commands and then substitutes their output in the place of the commands.
- Command substitution is performed when a command is given as ‘command’
- Here command, can be a simple command, a pipeline, or a list.

# Shell Programming

---

- **Command Substitution**
- Command substitution is generally used to assign the output of a command to a variable. Each of the following examples demonstrate command substitution:
- DATE='date'
- USERS='who | wc -l'
- UP='date ; uptime'

# Shell Programming

- **Command Substitution**

```
Sat Aug 8 19:15:38 IST 2020
eswaribala@DESKTOP-55AGI0I:~$ USERS=`who | wc -l`
eswaribala@DESKTOP-55AGI0I:~$ UP=`date ; uptime`
eswaribala@DESKTOP-55AGI0I:~$ echo $UP
Sat Aug 8 19:16:29 IST 2020 19:16:29 up 52 min, 0 users, load average: 0.52, 0.58, 0.59
eswaribala@DESKTOP-55AGI0I:~$
```

# Shell Programming

---

- **Arithmetic Substitution**
- In ksh and bash, the shell enables integer arithmetic to be performed.
- This avoids having to run an extra program such as expr or bc to do math in a shell script.
- This feature is not available in sh.
- Arithmetic substitution is performed when the following form of command is given:
- `$((expression))`
- Expressions are evaluated according to standard mathematical conventions.

# Shell Programming

## • Arithmetic Substitution

Operator	Description
/	The division operator. Divides two numbers and returns the result.
*	The multiplication operator. Multiples two numbers and returns the result.
-	The subtraction operator. Subtracts two numbers and returns the result.
+	The addition operator. Adds two numbers and returns the result.
( )	The parentheses clarify which expressions should be evaluated before others.

```
eswaribala@DESKTOP-55AGI0I: ~$ eswaribala@DESKTOP-55AGI0I: ~$ Data=$(((500 + 4) / (45 - 23) * 79))  
eswaribala@DESKTOP-55AGI0I: ~$ echo $Data  
1738  
eswaribala@DESKTOP-55AGI0I: ~$
```

## Creating a Script File

---

- To place shell commands in a text file, first you'll need to use a text editor to create a file, then enter the commands into the file.
- When creating a shell script file, you must specify the shell you are using in the first line of the file.
- The format for this is:
- `#!/bin/bash`

## Creating a Script File

---

- After indicating the shell, commands are entered onto each line of the file, followed by a carriage return.
- As mentioned, comments can be added by using the pound sign.
- An example looks like this:
- `#!/bin/bash`
- `# This script displays the date and who's logged on`
- `date`
- `who`

# Creating a Script File

---

- days=10
- guest="Kavitha"
- echo "\$guest checked in \$days days ago"
- days=5
- guest="Kumar"
- echo "\$guest checked in \$days days ago"
- var1=100
- var2=45
- var3=\$[ \$var1 / \$var2 ]
- echo The final result is \$var3

# Creating a Script File

## The expr Command Operators

Operator	Description
<i>ARG1</i>   <i>ARG2</i>	Return <i>ARG1</i> if neither argument is null or zero; otherwise, return <i>ARG2</i> .
<i>ARG1</i> & <i>ARG2</i>	Return <i>ARG1</i> if neither argument is null or zero; otherwise, return 0.
<i>ARG1</i> < <i>ARG2</i>	Return 1 if <i>ARG1</i> is less than <i>ARG2</i> ; otherwise, return 0.
<i>ARG1</i> <= <i>ARG2</i>	Return 1 if <i>ARG1</i> is less than or equal to <i>ARG2</i> ; otherwise, return 0.
<i>ARG1</i> = <i>ARG2</i>	Return 1 if <i>ARG1</i> is equal to <i>ARG2</i> ; otherwise, return 0.
<i>ARG1</i> != <i>ARG2</i>	Return 1 if <i>ARG1</i> is not equal to <i>ARG2</i> ; otherwise, return 0.
<i>ARG1</i> >= <i>ARG2</i>	Return 1 if <i>ARG1</i> is greater than or equal to <i>ARG2</i> ; otherwise, return 0.
<i>ARG1</i> > <i>ARG2</i>	Return 1 if <i>ARG1</i> is greater than <i>ARG2</i> ; otherwise, return 0.
<i>ARG1</i> + <i>ARG2</i>	Return the arithmetic sum of <i>ARG1</i> and <i>ARG2</i> .
<i>ARG1</i> - <i>ARG2</i>	Return the arithmetic difference of <i>ARG1</i> and <i>ARG2</i> .
<i>ARG1</i> * <i>ARG2</i>	Return the arithmetic product of <i>ARG1</i> and <i>ARG2</i> .
<i>ARG1</i> / <i>ARG2</i>	Return the arithmetic quotient of <i>ARG1</i> divided by <i>ARG2</i> .
<i>ARG1</i> % <i>ARG2</i>	Return the arithmetic remainder of <i>ARG1</i> divided by <i>ARG2</i> .
<i>STRING</i> : <i>REGEXP</i>	Return the pattern match if <i>REGEXP</i> matches a pattern in <i>STRING</i> .

# Creating a Script File

match <i>STRING</i> <i>REGEXP</i>	Return the pattern match if <i>REGEXP</i> matches a pattern in <i>STRING</i> .
substr <i>STRING</i> <i>POS LENGTH</i>	Return the substring <i>LENGTH</i> characters in length, starting at position <i>POS</i> (starting at 1).
index <i>STRING</i> <i>CHARS</i>	Return position in <i>STRING</i> where <i>CHARS</i> is found; otherwise, return 0.
length <i>STRING</i>	Return the numeric length of the string <i>STRING</i> .
+ <i>TOKEN</i>	Interpret <i>TOKEN</i> as a string, even if it's a keyword.
( <i>EXPRESSION</i> )	Return the value of <i>EXPRESSION</i> .

## Exiting the Script

---

- Every command that runs in the shell uses an exit status to indicate to the shell that it's done processing.
- The exit status is an integer value between 0 and 255 that's passed by the command to the shell when the command finishes running.
- You can capture this value and use it in your scripts.

# Exiting the Script

---

- **Checking the exit status**
- Linux provides the \$? special variable that holds the exit status value from the last command that executed.
- You must view or use the \$? variable immediately after the command you want to check.
- It changes values to the exit status of the last command executed by the shell:
  - \$ date
  - Sat Sep 29 10:01:30 EDT 2007
  - \$ echo \$?

# Exiting the Script

- **Checking the exit status**

## Linux Exit Status Codes

Code	Description
0	Successful completion of the command
1	General unknown error
2	Misuse of shell command
126	The command can't execute
127	Command not found
128	Invalid exit argument
128+x	Fatal error with Linux signal x
130	Command terminated with Ctl-C
255	Exit status out of range

# Exiting the Script

- **Checking the exit status**

```
eswaribala@DESKTOP-55AGI0I: ~/shellprograms
```

```
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ ./sample3.sh
```

```
The answer is .6880
```

```
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ echo $?
```

```
0
```

```
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ ■
```

# Using Structured Commands

---

- **Working with the if-then Statement**
- The most basic type of structured command is the if-then statement.
- The if-then statement has the following format:  
if command  
then  
commands  
fi

# Using Structured Commands

---

- **The if-then-else Statement**
- The if-then-else statement provides another group of commands in the statement:
- if command
- then
- commands
- else
- commands
- fi

# Using Structured Commands

---

- **Nesting ifs**
- The elif continues an else section with another if-then statement:
- if command1
- then
- command set 1
- elif command2
- then
- command set 2
- elif command3
- then
- command set 3
- elif command4
- then
- command set 4
- fi

# Using Structured Commands

---

## The test Numeric Comparisons

Comparison	Description
<code>n1 -eq n2</code>	Check if <i>n1</i> is equal to <i>n2</i> .
<code>n1 -ge n2</code>	Check if <i>n1</i> is greater than or equal to <i>n2</i> .
<code>n1 -gt n2</code>	Check if <i>n1</i> is greater than <i>n2</i> .
<code>n1 -le n2</code>	Check if <i>n1</i> is less than or equal to <i>n2</i> .
<code>n1 -lt n2</code>	Check if <i>n1</i> is less than <i>n2</i> .
<code>n1 -ne n2</code>	Check if <i>n1</i> is not equal to <i>n2</i> .

# Using Structured Commands

---

## The test Command String Comparisons

Comparison	Description
<code>str1 = str2</code>	Check if <code>str1</code> is the same as string <code>str2</code> .
<code>str1 != str2</code>	Check if <code>str1</code> is not the same as <code>str2</code> .
<code>str1 &lt; str2</code>	Check if <code>str1</code> is less than <code>str2</code> .
<code>str1 &gt; str2</code>	Check if <code>str1</code> is greater than <code>str2</code> .
<code>-n str1</code>	Check if <code>str1</code> has a length greater than zero.
<code>-z str1</code>	Check if <code>str1</code> has a length of zero.

# Using Structured Commands

## The test Command File Comparisons

Comparison	Description
<code>-d file</code>	Check if <i>file</i> exists and is a directory.
<code>-e file</code>	Checks if <i>file</i> exists.
<code>-f file</code>	Checks if <i>file</i> exists and is a file.
<code>-r file</code>	Checks if <i>file</i> exists and is readable.
<code>-s file</code>	Checks if <i>file</i> exists and is not empty.
<code>-w file</code>	Checks if <i>file</i> exists and is writable.
<code>-x file</code>	Checks if <i>file</i> exists and is executable.
<code>-o file</code>	Checks if <i>file</i> exists and is owned by the current user.
<code>-G file</code>	Checks if <i>file</i> exists and the default group is the same as the current user.
<code>file1 -nt file2</code>	Checks if <i>file1</i> is newer than <i>file2</i> .
<code>file1 -ot file2</code>	Checks if <i>file1</i> is older than <i>file2</i> .

## Advanced if-then Features

---

- There are two relatively recent additions to the bash shell that provide advanced features that you can use in if-then statements:
  - Double parentheses for mathematical expressions
  - Double square brackets for advanced string handling functions

## Advanced if-then Features

---

- **Using double parentheses**
- The double parentheses command allows you to incorporate advanced mathematical formulas in your comparisons.
- The test command only allows for simple arithmetic operations in the comparison.
- The double parentheses command provides more mathematical symbols that programmers from other languages are used to using.
- The format of the double parentheses command is:
- `(( expression ))`
- The expression term can be any mathematical assignment or comparison expression.

# The case Command

---

- **The case command checks multiple values of a single variable in a list-oriented format:**
- case variable in
- pattern1 | pattern2) commands1;;
- pattern3) commands2;;
- \*) default commands;;
- esac

# The for Command

---

- The bash shell provides the for command to allow you to create a loop that iterates through a series of values.
- Each iteration performs a defined set of commands using one of the values in the series.
- The basic format of the bash shell for command is:
  - for var in list
  - do
  - commands
  - done

# The while Command

---

- Basic while format
- The format of the while command is:
- while test command
- do
- other commands
- done

# The until Command

---

- The until command works exactly the opposite way from the while command.
- The until command requires that you to specify a test command that normally produces a non-zero exit status.
- As long as the exit status of the test command is non-zero, the bash shell executes the commands listed in the loop.
- Once the test command returns a zero exit status, the loop stops.
- As you would expect, the format of the until command is:
- until test commands
- do
- other commands
- done

## The break command

---

- The break command is a simple way to escape out of a loop in progress.
- You can use the break command to exit out of any type of loop, including while and until loops.

# Command Line Parameters

---

- The most basic method of passing data to your shell script is by using command line parameters.
- Command line parameters allow you to add data values to the command line when you execute the script:
- \$ ./addem 10 30

## Reading parameters

---

- The bash shell assigns special variables, called positional parameters, to all of the parameters entered in a command line.
- This even includes the name of the program the shell executes.
- The positional parameter variables are standard numbers, with \$0 being the name of the program, \$1 being the first parameter, \$2 being the second parameter, and so on, up to \$9 for the ninth parameter.

# Reading parameters

eswaribala@DESKTOP-55AGI0I: ~/shellprograms

```
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ sudo nano cmdlinedemo.sh
```

```
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ ./cmdlinedemo.sh 5 6
```

The first parameter is 5.

The second parameter is 6.

The total value is 30.

```
eswaribala@DESKTOP-55AGI0I:~/shellprograms$
```

## Counting parameters

---

- Instead of testing each parameter, you can just count how many parameters were entered on the command line.
- The bash shell provides a special variable for this purpose.
- The special `$#` variable contains the number of command line parameters included when the script was run.
- You can use this special variable anywhere in the script, just as a normal variable:

## Grabbing all the data

---

- The \$\* and \$@ variables provide one-stop shopping for all of your parameters.
- Both of these variables include all of the command line parameters within a single variable.
- The \$\* variable takes all of the parameters supplied on the command line as a single word.
- The word contains each of the values as they appear on the command line.
- Basically, instead of treating the parameters as multiple objects, the \$\* variable treats them all as one parameter.

## Grabbing all the data

---

- The \$@ variable on the other hand, takes all of the parameters supplied on the command line as separate words in the same string.
- It allows you to iterate through the value, separating out each parameter supplied.
- This is most often accomplished using the for command.

## Timing out

---

- There's a danger when using the read command.
- It's quite possible that your script will get stuck waiting for the script user to enter data.
- If the script must go on regardless of if there was any data entered, you can use the -t option specify a timer.
- The -t option specifies the number of seconds for the read command to wait for input.
- When the timer expires, the read command returns a non-zero exit status:

# Select Loop

---

- Basic 'select' Loop
- In select loop, we have the similar syntax as that of for loop. As mentioned above, select loop accepts a sequence to iterate through it and present it as a numbered menu.
- For this, we use the in keywordas shown in below syntax.
- Syntax:
- `select variableName in choice1 choice2 ... choiceN`
- do
- -- Block of Commands --
- 
- done

# continue Statement

---

- The continue statement in a loop, when certain condition becomes true, skips all the subsequent statements, coming after it, and continues with the next iteration of the loop.
- So, when a continue statement is reached, further portion of code is skipped from execution and next iteration is started.
- Syntax:
- while [ condition ]
- do
- -- Some Commands --
- if [ condition ]
- then
- continue
- fi
- -- More commands --
- done

## Silent Reading

---

- There are times when you need input from the script user, but you don't want that input to display on the monitor.
- The classic example of this is when entering passwords, but there are plenty of other types of data that you will need to hide.
- The `-s` option prevents the data entered in the `read` command from being displayed on the monitor (actually, the data is displayed, but the `read` command sets the text color to the same as the background color).

## Reading from a file

---

- Read command can be used to read data stored in a file on the Linux system.
- Each call to the read command reads a single line of text from the file.
- When there are no more lines left in the file, the read command will exit with a non-zero exit status.

# Understanding Input and Output

---

## Linux Standard File Descriptors

File Descriptor	Abbreviation	Description
0	STDIN	Standard input
1	STDOUT	Standard output
2	STDERR	Standard error

# Understanding Input and Output

---

- **Permanent redirections**
- If you have lots of data that you're redirecting in your script, it can get tedious having to redirect every echo statement.
- Instead, you can tell the shell to redirect a specific file descriptor for the duration of the script by using the exec command

# Understanding Input and Output

---

- **Redirecting Input in Scripts**
- You can use the same technique used to redirect STDOUT and STDERR in your scripts to redirect STDIN from the keyboard.
- The exec command allows you to redirect STDIN to a file on the Linux system:
- `exec 0< testfile`
- This command informs the shell that it should retrieve input from the file testfile instead of STDIN.
- **Refer filereading example**

# Creating Your Own Redirection

---

- Creating output file descriptors you assign a file descriptor for output by using the exec command.
- Just as with the standard file descriptors, once you assign an alternative file descriptor to a file location, that redirection stays permanent until you reassign it.
- **The other six file descriptors are numbered from three through eight and are available for you to use as either input or output redirection.**
- **Refer fileopenclose example**

# Closing file descriptors

---

- If you create new input or output file descriptors, the shell automatically closes them when the script exits.
- There are situations though when you need to manually close a file descriptor before the end of the script.
- To close a file descriptor, redirect it to the special symbol &-
- **Once you close the file descriptor, you can't write any data to it in your script or the shell produces an error message.**

# Logging Messages

---

- Sometimes it's beneficial to send output both to the monitor and to a file for logging.
- Instead of having to redirect output twice, you can use the special tee command.
- The tee command is like a T-connector for pipes.
- It sends data from STDIN to two destinations at the same time. One destination is STDOUT.
- The other destination is a filename specified on the tee command line:
- `tee filename`
- Since tee redirects data from STDIN, you can use it with the pipe command to redirect output from any command:

# Basic Script Functions

---

- Functions are blocks of script code that you assign a name to, then reuse anywhere in your code.
- Anytime you need to use that block of code in your script, all you need to do is use the function name you assigned it (referred to as calling the function).

# Basic Script Functions

---

- **Creating a function**
- There are two formats you can use to create functions in bash shell scripts.
- The first format uses the keyword function, along with the function name you assign to the block of code:
- `function name {`
- `commands`
- `}`

# Basic Script Functions

---

- **Creating a function**
- The commands are one or more bash shell commands that make up your function.
- When you call the function, the bash shell executes each of the commands in the order they appear in the function, just as in a normal script.

# Basic Script Functions

---

- **Creating a function**
- The second format for defining a function in a bash shell script more closely follows how functions
- are defined in other programming languages:
- name() {
- commands
- }
- The empty parentheses after the function name indicate that you're defining a function.
- The same naming rules apply in this format as in the original shell script function format.

# Basic Script Functions

---

- **Returning a Value**
- The bash shell uses the `return` command to exit a function with a specific exit status.
- The `return` command allows you to specify a single integer value to define the function exit status, providing an easy way for you to programmatically set the exit status of your function:

# Basic Script Functions

---

- **Handling variables in a function**
- One thing that causes problems for shell script programmers is the scope of a variable.
- The scope is where the variable is visible.
- Variables defined in functions can have a different scope than regular variables.
- That is, they can be hidden from the rest of the script.

# Basic Script Functions

---

- **Handling variables in a function**
- Functions use two types of variables:
  - Global
  - Local

# Basic Script Functions

---

- **Global variables**
- Global variables are variables that are valid anywhere within the shell script.
- If you define a global variable in the main section of a script, you can retrieve its value inside a function.
- Likewise, if you define a global variable inside a function, you can retrieve its value in the main section of the script.

# Basic Script Functions

---

- **Local variables**
- Instead of using global variables in functions, any variables that the function uses internally can be declared as local variables.
- To do that, just use the `local` keyword in front of the variable declaration:
- `local temp`
- You can also use the `local` keyword in an assignment statement while assigning a value to the variable:
- `local temp=$[ $value + 5 ]`

# Basic Script Functions

---

- **Local variables**
- The local keyword ensures that the variable is limited to only within the function.
- If a variable with the same name appears outside the function in the script, the shell keeps the two variable values separate.

# Basic Script Functions

---

- **Array Variables and Functions**
- Passing multiple values to function using array and returning the array from function.

# Basic Script Functions

---

- **Creating a Library**
- The bash shell allows you to create a library file for your functions, then reference that single library file in as many scripts as you need to.
- The first step in the process is to create a common library file that contains the functions you need in your scripts.

# Basic Script Functions

---

- **Using Functions on the Command Line**
- You can use script functions to create some pretty complex operations.
- Sometimes it would be nice to be able to use these functions directly on the command line interface prompt.
- Just as you can use a script function as a command in a shell script, you can also use a script function as a command in the command line interface.
- This is a nice feature, since once you define the function in the shell, you can use it from any directory on the system;
- You don't have to worry about a script being in your PATH environment variable

# Basic Script Functions

- **Using Functions on the Command Line**

```
eswaribala@DESKTOP-55AGI0I: ~
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ function divem { echo $[ $1 / $2 ]; }
eswaribala@DESKTOP-55AGI0I:~/shellprograms$ cd ..
eswaribala@DESKTOP-55AGI0I:~$ divem 67 67
1
eswaribala@DESKTOP-55AGI0I:~$ divem 24253 67
361
eswaribala@DESKTOP-55AGI0I:~$
```

**function bonus { echo `expr \$1\*0.05+\$2\*0.10|bc`; }**

# Basic Script Functions

- **Using Functions on the Command Line**

```
eswaribala@DESKTOP-55AGI0I:~$ eswaribala@DESKTOP-55AGI0I:~$ function doubleit { read -p "Enter value: " value; echo $[ $value * 2 ]; }  
eswaribala@DESKTOP-55AGI0I:~$ doubleit  
Enter value: 78687  
157374  
eswaribala@DESKTOP-55AGI0I:~$
```

```
function doubleit { read -p "Enter value: " value; echo $[  
$ value * 2 ]; }
```

# Basic Script Functions

- **Create the menu functions**
- **Adding Color (Refer colodename.sh)**

## The ANSI SGR Effect Control Codes

Code	Description
0	Reset to normal mode.
1	Set to bold intensity.
2	Set to faint intensity.
3	Use italic font.
4	Use single underline.
5	Use slow blink.
6	Use fast blink.
7	Reverse foreground/background colors.
8	Set foreground color to background color (invisible text).

# Basic Script Functions

- **Create the menu functions**
- **Adding Color**

## The ANSI Color Control Codes

Code	Description
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

# Text Manipulation

---

- **The sed editor**
- The sed editor is called a stream editor, as opposed to a normal interactive text editor.
- In an interactive text editor, such as vim, you interactively use keyboard commands to insert, delete, or replace text in the data.
- A stream editor edits a stream of data based on a set of rules you supply ahead of time, before the editor processes the data.

# Text Manipulation

---

- **The sed editor**
- The sed editor can manipulate data in a data stream based on commands you either enter into the command line or store in a command text file.
- It reads one line of data at a time from the input, matches that data with the supplied editor commands, changes data in the stream as specified in the commands, then outputs the new data to STDOUT.
- After the stream editor matches all of the commands against a line of data, it reads the next line of data and repeats the process.
- After the stream editor processes all of the lines of data in the stream, it terminates

# Text Manipulation

---

- **The sed editor**
- Since the commands are applied sequentially line by line, the sed editor only has to make one pass through the data stream to make the edits.
- This makes the sed editor much faster than an interactive editor, allowing you to quickly make changes to data in a file on the fly.
- The format for using the sed command is:
  - **sed options script file**
- The options parameters allow you to customize the behavior of the sed command

# Text Manipulation

- **The sed editor**

## The sed Command Options

Option	Description
<code>-e script</code>	Add commands specified in the script to the commands run while processing the input.
<code>-f file</code>	Add the commands specified in the file to the commands run while processing the input.
<code>-n</code>	Don't produce output for each command, but wait for the print command.

# Text Manipulation

---

- **The sed editor**
- How to add a header line say "Employee, Empld" to this file using sed?
- '1i' means to include the following before reading the first line and hence we got the header in the file.
- `sed '1i Employee, Empld' empfile`

# Text Manipulation

---

- **The sed editor**
- How to add a line '-----' after the header line or the 1st line?
- `sed -i '1a -----' empfile`
- '`1i`' is similar to '`1a`' except that '`i`' tells to include the content before reading the line, '`a`' tells to include the content after reading the line.
- `sed -i '$a -----' empfile` (trailing)

# Text Manipulation

---

- **The sed editor**
- How to add a record after a particular record?
- `sed -i '/Hilesh/a Bharti, 1002' empfile`
  
- How to add a record before a particular record?
- `sed -i '/Harshal/i Aparna, 1003' empFile`

# Text Manipulation

---

- **The sed editor**
- To add something to the beginning of a every line in a file, say to add a word Fruit:
- \$ sed 's/^/Fruit: /' fruits
- Similarly, to add something to the end of the file:
- \$ sed 's/\$/ Fruit/' fruits
- To replace or substitute a particular character, say to replace 'a' with 'A'.
- \$ sed 's/a/A/' fruits

# Text Manipulation

---

- **The sed editor**
- To replace or substitute all occurrences of 'a' with 'A'
- \$ sed 's/a/A/g' fruits
- Now, say to replace all occurrences from 2nd occurrence onwards:
- \$ sed 's/a/A/2g' fruits
- Say, you want to replace 'a' only in a specific line say 3rd line, not in the entire file:
- \$ sed '3s/a/A/g' fruits

# Text Manipulation

---

- **The sed editor**
- To replace or substitute 'a' on a range of lines, say from 1st to 3rd line:
- \$ sed '1,3s/a/A/g' fruits
- To replace the entire line with something. For example, to replace 'apple' with 'apple is a Fruit'.
- \$ sed 's/.\*/& is a Fruit/' fruits

# Text Manipulation

---

- **The sed editor**
- Using sed, we can also do multiple substitution. For example, say to replace all 'a' to 'A', and 'p' to 'P':
- `$ sed 's/a/A/g; s/p/P/g' fruits`

# Text Manipulation

---

- **The sed editor**
- Read the file2 after every line of file1.
- \$ sed 'r fruitnew' fruits
- we can also try to read a file contents on finding a pattern:
- \$ sed '/banana/r fruitnew' fruits

# Text Manipulation

---

- **The sed editor**
- Write the contents from the 3rd line onwards to a different file:
- `$ sudo sed -n '3,$w fruitnew' fruits`

# Text Manipulation

---

- **The sed editor**
- **Viewing a range of lines of a document**
  - `sed -n '5,10p' /home/eswaribala/EmployeeData.txt`
- **Viewing the entire file except a given range**
  - `sed '20,35d' /home/eswaribala/EmployeeData.txt`
- **Viewing non-consecutive lines and ranges**
  - `sed -n -e '5,7p' -e '10,13p'`  
`/home/eswaribala/EmployeeData.txt`

# Text Manipulation

---

- **The sed editor**
- **Replacing words or characters (basic substitution)**
- To replace every instance of the word old name with new name in EmployeeData.txt, do:
  - `sed 's/Martina/Madhu/g'`  
`/home/eswaribala/EmployeeData.txt`
  - Additionally, you may want to consider using gi instead of g in order to ignore character case:
  - `# sed 's/version/story/gi' myfile.txt`

# Text Manipulation

---

- **The sed editor**
  - **ip route show**
- To replace multiple blank spaces with a single space, we will use the output of ip route show and a pipeline:
  - **ip route show | sed 's/ \*/ /g'**
- Replacing words or characters inside a range If you're interested in replacing words only within a line range (30 through 40, for example), you can do:
  - **# sed '5,10 s/Martina/Madhu/g'  
/home/eswaribala/EmployeeData.txt**

# Text Manipulation

---

- **The sed editor**
  - **Using regular expressions**
  - To remove empty lines or those beginning with # from the Apache configuration file, do:
  - `sed '/^#\||^$|\ *#/d' apache2.conf`
  - The caret sign followed by the number sign (^#) indicates the beginning of a line, whereas ^\$ represents blank lines.
  - The vertical bars indicate boolean operations, whereas the backward slash is used to escape the vertical bars.

# Text Manipulation

---

- **The sed editor**
  - **Using regular expressions**
  - To replace a word beginning with uppercase or lowercase with another word, we can also use sed. To illustrate, let's replace the word zip or Zip with rar in myfile.txt:
  - `# sed 's/[Zz]ip/rar/g' myfile.txt`
  - **Viewing lines containing with a given pattern**
  - **In this case, the pattern to search for is nMadhu at the beginning of each line:**
  - `sed -n '/^LogFormat / p' apache2.conf`

# Text Manipulation

---

- **Inserting spaces in files**
- With sed, we can also insert spaces (blank lines) for each non-empty line in a file. To insert one blank line every other line in LICENSE, a plain text file, do:
  - `# sed G myfile.txt`
  - To insert two blank lines, do:
    - `# sed 'G;G' myfile.txt`

# Text Manipulation

---

- **Performing two or more substitutions at once**
- You can combine two or more substitutions one single sed command.
- Let's replace the words that and line in myfile.txt with This and verse, respectively.
- Note how this can be done by using an ordinary sed substitution command followed by a semicolon and a second substitution command:
- `# sed -i 's/that/this/gi;s/line/verse/gi' myfile.txt`

## gawk Editor

---

- While the sed editor is a handy tool for modifying text files on the fly, it has its limitations.
- Often you need a more advanced tool for manipulating data in a file, one that provides a more programming-like environment allowing you to modify and reorganize data in a file.
- This is where gawk comes in.
- The gawk program is the GNU version of the original awk program in Unix.
- The awk program takes stream editing one step further than the sed editor by providing a programming language instead of just editor commands.

## gawk Editor

---

- Within the programming language you can:
  - Define variables to store data.
  - Use arithmetic and string operators to operate on data.
  - Use structured programming concepts, such as if-then statements and loops, to add logic to your data processing.
  - Generate formatted reports by extracting data elements within the data file and repositioning them in another order or format.

## gawk Editor

---

- The gawk program's report-generating abilities are often used for extracting data elements from large bulky text files and formatting them into a readable report.
- The perfect example of this is formatting log files.
- Trying to pore through lines of errors in a log file can be difficult.
- The gawk program allows you to filter just the data elements you want to view from the log file, then format them in a manner that makes reading the important data easier.

# gawk Editor

---

- **The gawk command format**
- The basic format of the gawk program is:
  - gawk options program file
  - The command line options provide an easy way to customize features in the gawk program.
  - We'll be looking more closely at these as we explore using gawk.
  - The power of gawk is in the program script.
  - You can write scripts to read the data within a line of text, then manipulate and display the data to create any type of output report.

## gawk Editor

---

- To define an awk script, use braces surrounded by single quotation marks like this:
- \$ awk '{print "Welcome to awk Text Manipulator"}'
- **If you type anything, it returns the same welcome string we provide.**
- **To terminate the program, press The Ctrl+D.**
- **Looks tricky, don't panic, the best is yet to come.**

# gawk Editor

---

- **Built-in Variables #**
- Awk has a number of built-in variables that contain useful information and allows you to control how the program is processed.
- Below are some of the most common built-in Variables:
  - NF - The number of fields in the record.
  - NR - The number of the current record.
  - FILENAME - The name of the input file that is currently processed.
  - FS - Field separator.
  - RS - Record separator.
  - OFS - Output field separator.
  - ORS - Output record separator.

# gawk Editor

---

- **Built-in Variables #**
- To print the file name and the number of lines (records):
- `awk 'END { print "File", FILENAME, "contains", NR, "lines." }' teams.txt`.

## gawk Editor

---

- **Built-in Variables #**
- Changing the Field and Record Separator
- The default value of the field separator is any number of space or tab characters. It can be changed by setting in the FS variable.
- For example, to set the **field separator to .** you would use:
  - awk 'BEGIN { FS = "." } { print \$1 }' teams.txt
  - Or
  - awk -F "." '{ print \$1 }' teams.txt

# gawk Editor

---

- **Built-in Variables #**
- Here is an example showing how to change the record separator to .:  
• awk 'BEGIN { RS = "." } { print \$1 }' teams.txt

# gawk Editor

---

- **Using Variables**
- With awk, you can process text files. Awk assigns some variables for each data field found:
  - \$0 for the whole line.
  - \$1 for the first field.
  - \$2 for the second field.
  - \$n for the nth field.
- The whitespace character like space or tab is the default separator between fields in awk.

# gawk Editor

---

- **Using Variables**
- awk '{print \$1}' empfile
- echo "Hello Param" | awk '{\$2="Eswari"; print \$0}'
- awk '{print \$1,\$2,\$NF}' dennis\_ritchie.txt
- \$NF prints last field
- awk '{ print \$1, \$3, \$5 }' teams.txt
- awk '{ print "The first field:", \$1}' teams.txt
- awk 'BEGIN { print "First line\nSecond line\nThird line" }'

# gawk Editor

---

- **Using Variables**
- awk '{ printf "%3d. %s\n", NR, \$0 }' teams.txt
- awk '{ sum += \$3 } END { printf "%d\n", sum }' teams.txt
- awk 'BEGIN { i = 1; while (i < 6) { print "Square of", i, "is", i\*i; ++i } }'
- awk -v seed=\$RANDOM  
'BEGIN{i=1;srand(seed);while(i<100){print rand()\*10000,i++}}'

# gawk Editor

---

- **Prog.awk**
- **BEGIN {**
- **i = 1**
- **while (i < 6) {**
- **print "Square of", i, "is", i\*i;**
- **++i**
- **}**
- **}**

## gawk Editor

---

- **Run the program by passing the file name to the awk interpreter:**
- awk -f prog.awk
- chmod +x prog.awk
- ./prog.awk

# gawk Editor

---

- **Using Shell Variables in Awk Programs**
- num=51
- awk -v n="\$num" 'BEGIN {print n}'

## gawk Editor

---

- **Adding Output Field Separators**
- date
- date | awk '{print \$2,\$3,\$6}'
- date | awk 'OFS="/" {print\$2,\$3,\$6}'
- date | awk 'OFS="-" {print\$2,\$3,\$6}'

# gawk Editor

---

- **Awk Patterns**
- awk '{ print \$3 }' teams.txt
- awk '/0.5/ { print \$1 }' teams.txt
- awk '/^[0-9][0-9]/ { print \$1 }' teams.txt
- To print the first field of each record whose second field contains “ia” you would type:
- awk '\$2 ~ /ia/ { print \$1 }' teams.txt

# gawk Editor

---

- **Awk Patterns**
- To match fields that do not contain a given pattern use the `!~` operator:
- `awk '$2 !~ /ia/ { print $1 }' teams.txt`
- The following command prints the first field of all records whose third field is greater than 50:
- `awk '$3 > 50 { print $1 }' teams.txt`

# gawk Editor

---

- **Awk Patterns**
- Range patterns
- To print the first field of all records starting from the record including “Raptors” until the record including “Celtics”:
  - `awk '/Raptors/,/Celtics/ { print $1 }' teams.txt`

# gawk Editor

---

- **Special expression patterns**
- **The BEGIN and END Rules**
  - A BEGIN rule is executed once before any text processing starts.
  - In fact, it's executed before awk even reads any text.
  - An END rule is executed after all processing has completed.
  - You can have multiple BEGIN and END rules, and they'll execute in order.

## gawk Editor

---

- **The BEGIN and END Rules**
- The BEGIN pattern is generally used to set variables and the END pattern to process data from the records such as calculation.
- To print “Start Processing.”, then print the third field of each record and finally “End Processing.”:
- `awk 'BEGIN { print "Start Processing." }; { print $3 }; END { print "End Processing." }' teams.txt`

## gawk Editor

---

- **The BEGIN and END Rules**
- **Combining patterns #**
- Awk allows you to combine two or more patterns using the logical AND operator (`&&`) and logical OR operator (`||`).
- Here is an example that uses the `&&` operator to print the first field of those record whose third field is greater than 50 and the fourth field is less than 30:
- `awk '$3 > 50 && $4 < 30 { print $1 }' teams.txt`

# gawk Editor

## The gawk Options

Option	Description
<code>-F <i>fs</i></code>	Specify a file separator for delineating data fields in a line.
<code>-f <i>file</i></code>	Specify a filename to read the program from.
<code>-v <i>var=value</i></code>	Define a variable and default value used in the gawk program.
<code>-mf <i>N</i></code>	Specify the maximum number of fields to process in the data file.
<code>-mr <i>N</i></code>	Specify the maximum record size in the data file.
<code>-W <i>keyword</i></code>	Specify the compatibility mode or warning level for gawk.

## gawk Editor

---

- -F: It uses FS for the input field separator (the value of the FS predefined variable).
  - **gawk -F: '{print \$1}' /etc/passwd**

# The Unix File System

---

- The Unix file system is a methodology for logically organizing and storing large quantities of data such that the system is easy to manage.
- 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.

# The Unix File System

---

- By contrast, a file system consists of files, relationships to other files, as well as the attributes of each file.
- File attributes are information relating to the file, but do not include the data contained within a file.
- File attributes for a generic operating system might include (but are not limited to):
  - a file type (i.e. what kind of data is in the file)
  - a file name (which may or may not include an extension)
  - a physical file size
  - a file owner
  - file protection/privacy capability
  - file time stamp (time and date created/modified)

# The Unix File System

---

- Additionally, file systems provide tools which allow the manipulation of files, provide a logical organization as well as provide services which map the logical organization of files to physical devices.
- From the beginners perspective, the Unix file system is essentially composed of files and directories.
- Directories are special files that may contain other files.

# The Unix File System

---

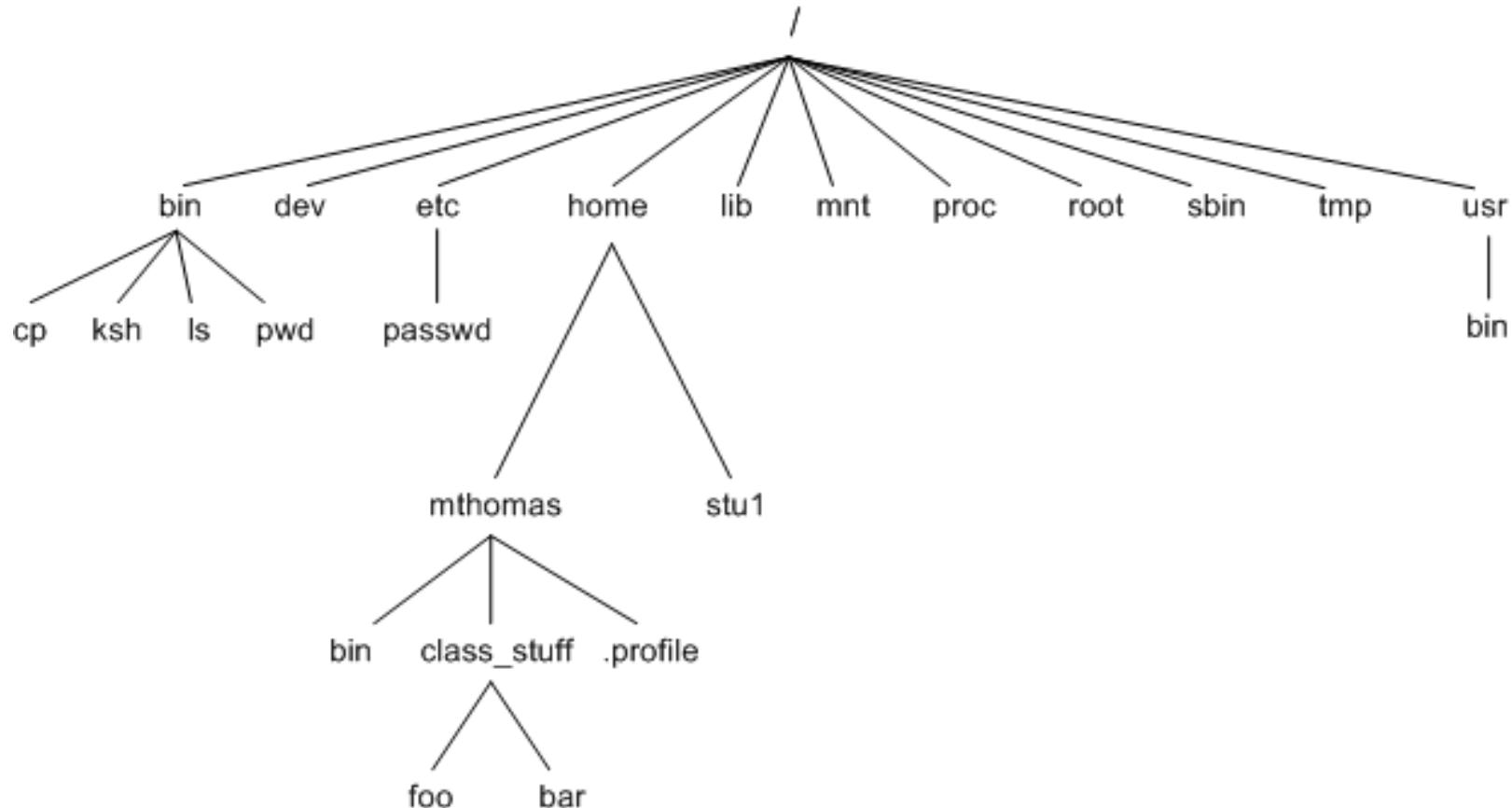
- The Unix file system has a hierarchical (or tree-like) structure with its highest level directory called root (denoted by /, pronounced slash).
- Immediately below the root level directory are several subdirectories, most of which contain system files.
- Below this can exist system files, application files, and/or user data files.
- Similar to the concept of the process parent-child relationship, all files on a Unix system are related to one another.
- That is, files also have a parent-child existence.
- Thus, all files (except one) share a common parental link, the top-most file (i.e. /) being the exception.

# The Unix File System

---

- Below is a diagram (slice) of a "typical" Unix file system.
- The top-most directory is / (slash), with the directories directly beneath being system directories.
- Note that as Unix implementations and vendors vary, so will this file system hierarchy.
- However, the organization of most file systems is similar.

# The Unix File System



# The Unix File System

---

- bin - short for binaries, this is the directory where many commonly used executable commands reside
- dev - contains device specific files
- etc - contains system configuration files
- home - contains user directories and files
- lib - contains all library files
- mnt - contains device files related to mounted devices
- proc - contains files related to system processes
- root - the root users' home directory (note this is different than /)

# The Unix File System

---

- sbin - system binary files reside here. If there is no sbin directory on your system, these files most likely reside in etc
- tmp - storage for temporary files which are periodically removed from the filesystem
- usr - also contains executable commands

# The Unix File System

---

- **du COMMAND**
- du command (short for disk usage) is useful command which is used to find disk usage for files & directories.
- du command when used with various options provides results in many formats.
- 1- To find out summary of disk usage for a directory with all its sub-directories
- \$ sudo du /home

# The Unix File System

---

- **2- Disk usage with file/directory sizes in human readable format i.e. in kb, mb etc**
- **\$ du -h /home**
- **3- Total disk size of a directory**
- **\$ du -s /home**

# The Unix File System

---

- **df COMMAND**
- df command (short for disk filesystem) is used to show disk utilization for a Linux system.
- Some examples are shared below.
- 1- To display information of device name, total blocks, total disk space, used disk space, available disk space and mount points on a file system.
- **\$ df**

# The Unix File System

---

- **df COMMAND**
- 2- Information in human readable format
- \$ df –h
- Display information of a particular partition
- \$ df -hT /etc

# The Unix File System

---

- On Linux and UNIX operating systems, you can use the mount command to attach (mount) file systems and removable devices such as USB flash drives at a particular mount point in the directory tree.
- The umount command detaches (unmounts) the mounted file system from the directory tree.

# The Unix File System

---

- **How to List Mounted File Systems #**
- When used without any argument, the mount command will display all currently attached file systems:
- \$mount
- device\_name on directory type filesystem\_type (options)

# The Unix File System

---

- **How to List Mounted File Systems #**
- To display only certain file systems use the -t option.
- For example, to print only the ext4 partitions you would use:
- `mount -t ext4`
- For example, to mount the `/dev/sdb1` file system to the `/mnt/media` directory you would use:
- `sudo mount /dev/sdb1 /mnt/media`

# The Unix File System

---

- **Mounting USB Drive**
- On most modern Linux distribution like Ubuntu, USB drives will auto mount when you insert it, but sometimes you may need to manually mount the drive.
- To manually mount a USB device, perform the following steps:

# The Unix File System

---

- **Mounting USB Drive**
- Create the mount point:
- sudo mkdir -p /media/usb
- Assuming that the USB drive uses the /dev/sdd1 device you can mount it to /media/usb directory by typing:
- sudo mount -t tmpfs /dev/sdd1 /media/usb
- ls /media

# The Unix File System

---

- **Mounting USB Drive**
- To find the device and filesystem type, you can use any of the following commands:
  - `fdisk -l`
  - `ls -l /dev/disk/by-id/usb*`
  - `dmesg`
  - `lsblk`

# The Unix File System

---

- **Mounting ISO Files #**
- You can mount an ISO file using the loop device which is a special pseudo-device that makes a file accessible as a block device.
- Start by creating the mount point, it can be any location you want:
- `sudo mkdir /media/iso`
- Mount the ISO file to the mount point by typing the following command:
- `sudo mount /path/to/image.iso /media/iso -o loop`

# The Unix File System

---

- **Unmounting a File System**
- To detach a mounted file system, use the umount command followed by either the directory where it has been mounted (mount point) or the device name:
  - umount DIRECTORY
  - umount DEVICE\_NAME
  - sudo umount /media/usb
- If the file system is in use the umount command will fail to detach the file system.
- In those situations, you can use the fuser command to find out which processes are accessing the file system:
  - fuser -m DIRECTORY

# The Unix File System

---

- **Lazy unmount**
- Use the `-l` (`--lazy`) option to unmount a busy file system as soon as it is not busy anymore.
- `umount -l DIRECTORY`
- Force unmount
- Use the `-f` (`--force`) option to force an unmount. This option is usually used to unmount an unreachable NFS system.
- `umount -f DIRECTORY`

# User and Group Quota

---

- On Linux, you can setup disk quota using one of the following methods:
  - **File system base disk quota allocation**
  - **User or group based disk quota allocation**

# User and Group Quota

---

- **On the user or group based quota**, following are three important factors to consider:
- Hard limit – For example, if you specify 2GB as hard limit, user will not be able to create new files after 2GB
- Soft limit – For example, if you specify 1GB as soft limit, user will get a warning message “disk quota exceeded”, once they reach 1GB limit. But, they’ll still be able to create new files until they reach the hard limit
- Grace Period – For example, if you specify 10 days as a grace period, after user reach their hard limit, they would be allowed additional 10 days to create new files. In that time period, they should try to get back to the quota limit.

# User and Group Quota

---

- **Enable quota check on filesystem**
- First, you should specify which filesystem are allowed for quota check.
  - Modify the /etc/fstab, and add the keyword usrquota and grpquota to the corresponding filesystem that you would like to monitor.
  - The following example indicates that both user and group quota check is enabled on /home filesystem
  - **\$ cat /etc/fstab**
  - ```
LABEL=cloudimg-rootfs   /      ext4  defaults    0 0
```

# User and Group Quota

---

- **Initial quota check on Linux filesystem using quotacheck**
- Once you've enabled disk quota check on the filesystem, collect all quota information initially as shown below.
- \$ repquota /dev/sda3
- quotacheck: Scanning /dev/sda3 [/home] done
- quotacheck: Checked 5182 directories and 31566 files
- quotacheck: Old file not found.
- quotacheck: Old file not found.

# Creating groups and adding users

---

- `sudo groupadd editorial`
- `sudo usermod -a -G editorial olivia`
- The `-a` option tells `usermod` we are appending and the `-G` option tells `usermod` we are appending to the group name that follows the option.
- `grep editorial /etc/group`
- Below command will change the group `group_old` to `group_new` using `-n` option.
- `groupmod -n group_new group_old`

## Creating groups and adding users

---

- Giving groups permissions to directories
- Let's say you have the directory /READERS and you need to allow all members of the readers group access to that directory. First, change the group of the folder with the command:
- `sudo chown -R :readers /READERS`
- Next, remove write permission from the group with the command:
- `sudo chmod -R g-w /READERS`

## Creating groups and adding users

---

- Now we remove the others x bit from the /READERS directory (to prevent any user not in the readers group from accessing any file within) with the command:
- **sudo chmod -R o-x /READERS**

## Creating groups and adding users

---

- Let's say you have the directory /EDITORS and you need to give members of the editors group read and write permission to its contents. To do that, the following command would be necessary:
- sudo chown -R :editors /EDITORS
- sudo chmod -R g+w /EDITORS
- sudo chmod -R o-x /EDITORS

## Creating groups and adding users

---

- To delete a group named staff, enter:
- `sudo groupdel staff`

# Command Line Tools to Monitor Linux Performance



- Top – Linux Process Monitoring
- Glances
- VmStat – Virtual Memory Statistics
- Lsof – List Open Files
- Tcpdump – Network Packet Analyzer
- `tcpdump -i -eth0`
- `netstat -a | more`
- Htop – Linux Process Monitoring

# Command Line Tools to Monitor Linux Performance



- iotop – Monitor Linux Disk I/O
- iostat – Input/Output Statistics
  - sudo apt install sysstat
  - iostat
- iptraf – Real Time IP LAN Monitoring
  - sudo apt install iptraf
  - Iptraf-ng
- Psacct or Acct – Monitor User Activity
- Monit – Linux Process and Services Monitoring
- NetHogs – Monitor Per Process Network Bandwidth
- iftop – Network Bandwidth Monitoring
- Monitorix – System and Network Monitoring

# Command Line Tools to Monitor Linux Performance



- Arpwatch – Ethernet Activity Monitor
- . Suricata – Network Security Monitoring
- VnStat PHP – Monitoring Network Bandwidth
- Nagios – Network/Server Monitoring
- Nmon: Monitor Linux Performance
- Collectl: All-in-One Performance Monitoring Tool

# Linux Logs

---

- Linux logs provide a timeline of events for the Linux operating system, applications, and system, and are a valuable troubleshooting tool when you encounter issues.
- Essentially, analyzing log files is the first thing an administrator needs to do when an issue is discovered.

## Linux Logs

---

- For desktop app-specific issues, log files are written to different locations.
- For example, Chrome writes crash reports to ‘`~/.chrome/Crash Reports`’).
- Where a desktop application writes logs depends on the developer, and if the app allows for custom log configuration.

# Linux Logs

---

- Files are stored in plain-text and can be found in the /var/log directory and subdirectory.
- There are Linux logs for everything: system, kernel, package managers, boot processes, Xorg, Apache, MySQL.

# How to View Linux Logs

---

- Linux logs can be viewed with the command cd/var/log, then by typing the command ls to see the logs stored under this directory.
- One of the most important logs to view is the syslog, which logs everything but auth-related messages.
- Issue the command var/log/syslog to view everything under the syslog, but zooming in on a specific issue will take a while, since this file tends to be long.
- You can use Shift+G to get to the end of the file, denoted by “END.”

# How to View Linux Logs

---

- You can also view logs via dmesg, which prints the kernel ring buffer.
- It prints everything and sends you to the end of the file.
- From there, you can use the command `dmesg | less` to scroll through the output.
- If you want to view log entries for the user facility, you need to issue the command `dmesg –facility=user`.

# How to View Linux Logs

---

- Lastly, you can use the tail command to view log files.
- It is one of the handiest tools you can use, since it only shows the last part of the logs, where the problem usually lies.
- For this, use the command `tail /var/log/syslog` or `tail -f /var/log/syslog`.
- `tail` will continue watching the log file, and print out the next line written to the file, allowing you to follow what is written to `syslog` as it happens

# How to View Linux Logs

---

- Most Important Linux Logs
- Most directories can be grouped into one of four categories:
- Application Logs
- Event Logs
- Service Logs
- System Logs

# How to View Linux Logs

---

- /var/log/syslog or /var/log/messages: general messages, as well as system-related information.
- Essentially, this log stores all activity data across the global system.
- Note that activity for Redhat-based systems, such as CentOS or Rhel, are stored in messages, while Ubuntu and other Debian-based systems are stored in Syslog.

# How to View Linux Logs

- syslog is a protocol for tracking and logging system messages in Linux.
- Applications use syslog to export all their error and status messages to the files in the /var/log directory.
- syslog uses the client-server model; a client transmits a text message to the server (receiver).
- The server is commonly called syslogd, syslog daemon, or syslog server. syslog uses the User Datagram Protocol (UDP) port 514 for communication.
- The messages are sent in cleartext, although an SSL wrapper can be used to provide encryption.
- Each message sent to the syslog server has two labels associated with it that make the message easier to handle.

# How to View Linux Logs

---

- The first label describes the function (facility) of the application that generated it.
- For example, mail servers typically log using the mail facility.
- The second label specifies the severity level. After these two labels, the action is specified.
- The action is usually a filename in the /var/log directory tree, in which the messages will be stored:

# How to View Linux Logs

---

- /var/log/auth.log or /var/log/secure: store authentication logs, including both successful and failed logins and authentication methods.
- Again, the system type dictates where authentication logs are stored; Debian/Ubuntu information is stored in /var/log/auth.log, while Redhat/CentroOS is stored in /var/log/secure.

# How to View Linux Logs

---

- /var/log/boot.log
- /var/log/maillog or var/log/mail.log
- /var/log/kern
- /var/log/dmesg
- /var/log/faillog
- /var/log/cron
- /var/log/yum.log
- /var/log/httpd/
- /var/log/mysqld.log or /var/log/mysql.log

# How To Manage Logfiles with Logrotate on Ubuntu

---

- Logrotate is a system utility that manages the automatic rotation and compression of log files.
- If log files were not rotated, compressed, and periodically pruned, they could eventually consume all available disk space on a system.
- Logrotate is installed by default on Ubuntu, and is set up to handle the log rotation needs of all installed packages, including rsyslog, the default system log processor.

# How To Manage Logfiles with Logrotate on Ubuntu

---

- Logrotate is available on many other Linux distributions as well, but the default configuration may be quite different.
- `logrotate --version`
- `cat /etc/logrotate.d/apt`

# Signals and Trap

---

- Signals are software interrupts sent to a program to indicate that an important event has occurred.
- The events can vary from user requests to illegal memory access errors.
- Some signals, such as the interrupt signal, indicate that a user has asked the program to do something that is not in the usual flow of control.

# Signals and Trap

| Signal Name | Signal Number | Description                                                                                         |
|-------------|---------------|-----------------------------------------------------------------------------------------------------|
| SIGHUP      | 1             | Hang up detected on controlling terminal or death of controlling process                            |
| SIGINT      | 2             | Issued if the user sends an interrupt signal (Ctrl + C)                                             |
| SIGQUIT     | 3             | Issued if the user sends a quit signal (Ctrl + D)                                                   |
| SIGFPE      | 8             | Issued if an illegal mathematical operation is attempted                                            |
| SIGKILL     | 9             | If a process gets this signal it must quit immediately and will not perform any clean-up operations |
| SIGALRM     | 14            | Alarm clock signal (used for timers)                                                                |
| SIGTERM     | 15            | Software termination signal (sent by kill by default)                                               |

# Signals and Trap

- List of Signals
- There is an easy way to list down all the signals supported by your system.
- Just issue the kill -l command and it would display all the supported signals.

```
eswaribala@DESKTOP-55AGI0I: /$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
eswaribala@DESKTOP-55AGI0I: /$
```

# Signals and Trap

---

- **Sending Signals**
- There are several methods of delivering signals to a program or script.
- One of the most common is for a user to type CONTROL-C or the INTERRUPT key while a script is executing.
- When you press the Ctrl+C key, a SIGINT is sent to the script and as per defined default action script terminates.
- The other common method for delivering signals is to use the kill command, the syntax of which is as follows –
- `$ kill -signal pid`

# Signals and Trap

---

- Here signal is either the number or name of the signal to deliver and pid is the process ID that the signal should be sent to.
- For Example –
- \$ kill -1 1001
- The above command sends the HUP or hang-up signal to the program that is running with process ID 1001.
- To send a kill signal to the same process, use the following command –
- \$ kill -9 1001
- This kills the process running with process ID 1001.

# Trapping Signals

---

- When you press the Ctrl+C or Break key at your terminal during execution of a shell program, normally that program is immediately terminated, and your command prompt returns.
- This may not always be desirable.
- For instance, you may end up leaving a bunch of temporary files that won't get cleaned up.
- Trapping these signals is quite easy, and the trap command has the following syntax –
- `$ trap commands signals`

# Trapping Signals

---

- Here command can be any valid Unix command, or even a user-defined function, and signal can be a list of any number of signals you want to trap.
- There are two common uses for trap in shell scripts –
- Clean up temporary files
- Ignore signals

# Trapping Signals

---

- Trap command without any option and arg
- Run the following command from the terminal to display the list of all commands associated with each condition. If any `trap` command is not set before then the following command will not display any information.
- \$ trap
- Trap command with -l option
- Run the following command from the terminal to display the list of all signal names with number.
- \$ trap -l
- The output of the above command will show the list of 64 signals with numbers.

# Questions



# Module Summary

---

- Unix Architecture
- Directories and Files
- File Management
- Editors
- Shell Scripting
- Utilities

