

# Linux



## Shell Scripting

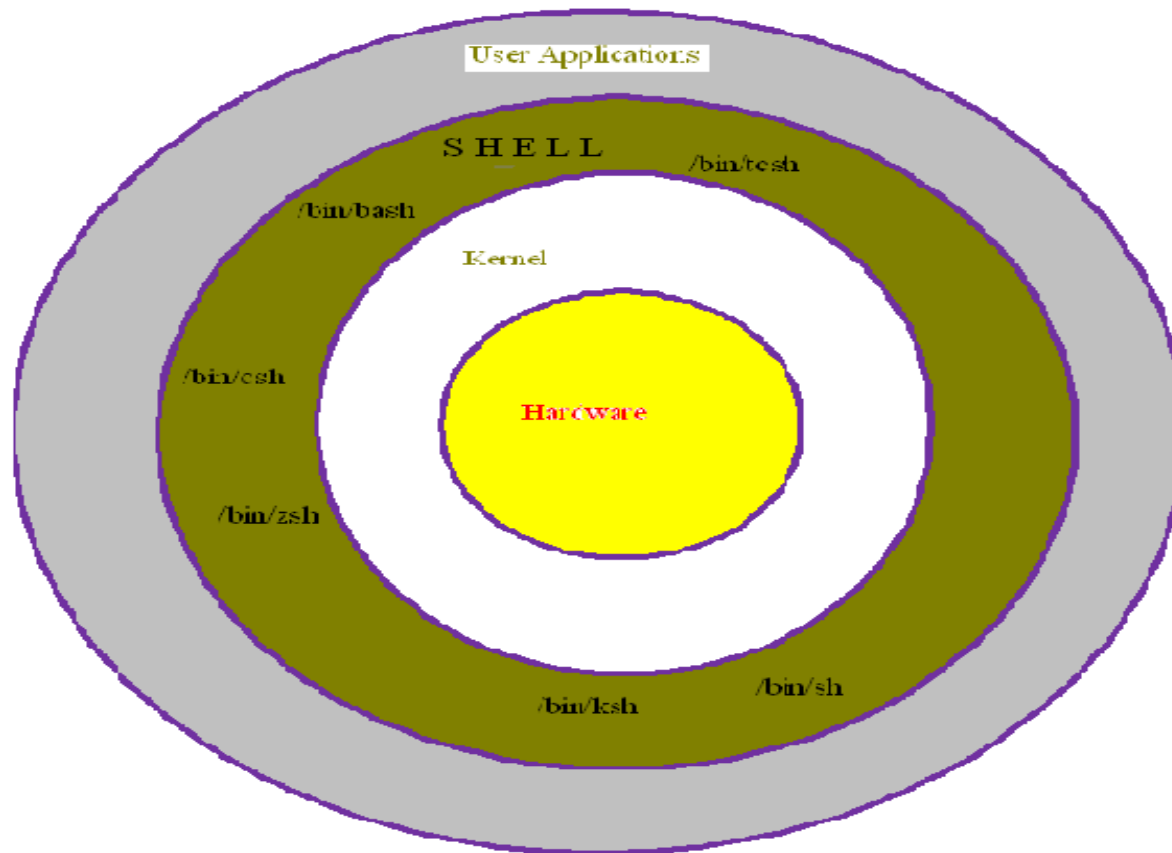
– Shamiyoddin Sayyad

# Agenda

---

- Architecture
- Basics
- Commands

# Unix Architecture



# Unix Architecture Continued .....

---

- **Kernel:**




- Controls hardware.
- Deals with Device, processor , memory , file, network management
- Is an interface between user applications and the hardware.

## **Shell:**

- Shell is an interface between the user and the kernel.
- Is a text only command interpreter.
- Sh( the bourne Shell) is the original Unix shell
- There are many others - Ksh, bash, ash, csh, tcsh, zsh


# Basics:

---

- Login:
- When you connect to Linux login prompt appears.
- login as: <User Name>
- E5252108@etaq517.tcc.etn.com's password<Password> 
- Enter Username and password.
- Unix takes you to shell prompt.
- etaq517:/home/E5252108><Type Command Here> 
- Changing Password:
- etaq517:/home/E5252108> passwd 

# Format of Unix Commands

---

- The General Syntax of Unix Command
- \$ command -options targets 
- The options sometimes can be listed together after a single dash.
- Unix Commands are case sensitive.
- A UNIX command's output may be redirected directly to another command's input by the pipe symbol |.

# Commands

- echo My First Command



- date



- hostname



- uname -a



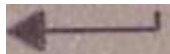
- uptime



- who am I



- who



- id



- last



- W



- echo \$SHELL



# Commands continued

- `echo {con,pre}{sent,fer}{s,ed}` 

- `man ls` 

----- Press q to quit.

- `man who` 

----- Press q to quit.

- `Clear` 

- `cal 2010` 

- `cal 9 1752` 

- `time sleep 5` 

- `history` 



# Commands continued.....

- yes please 


----- Press Ctrl + q to quit

- bc -l 

-----Press Ctrl + d to quit

- echo 5+4 | bc -l 

- echo {con,pre}{sent,fer}{s,ed} 

- top 

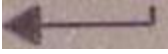
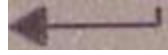

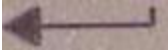
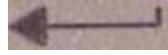

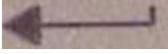
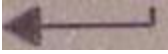

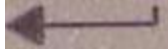

- arch 

- top 

- dmesg | more 










-----Press q to quit

# Directory Commands ...

- cd 
- pwd 
- ls -al 
- cd . 
- pwd 
- cd .. 
- pwd 
- ls -al 
- cd .. 
- pwd 
- ls -al 



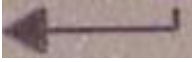
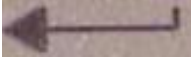
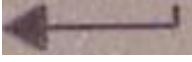
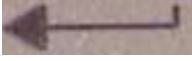
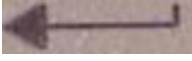
# Directory Commands ...

---

- `cd ..` 
- `Pwd` 
- `cd /etc` 
- `ls -al | more` 
- `cat passwd` 
- `cd -` 
- `pwd` 
- `cd ~<User Name>` 
- `pwd` 

# Directory Listing Commands ...

---

- `ls -d` 
- `ls -l` 
- `ls -R` 
- `ls -F` 
- `ls -lc` 
- `ls -al` 
- `ls -lrt` 

# Directory Commands

- `cd $HOME` 
- `mkdir iitc bicoe` 
- `cp /etc/passwd $HOME` 
- `cd bicoe` 
- `cd ../iitc` 
- `ls` 
- `echo hello linux > hello_file.txt` 
- `cat hello_file.txt` 
- `rmdir -r $HOME/bicoe` 

# The Unix File System

---

- **Four types of items are stored in a Unix Filesystem**

1. Ordinary Files

2. Directories

3. Devices and

4. Links

# The Unix File System

---

## 1. Ordinary Files

- Contain text, Data, Program.
- Files cannot contain other Files or Directories
- Unix file names are not broken into a name part and an extension part. However used to classify files.
- Can contain any keyboard character except '/' and be up to 256 character long.
- characters such as \*,?,# and & have special meaning in most shells and should not therefore be used in filenames

# The Unix File System Continued ...

---

## 2. Directories

- Directories are containers or folders that hold files, and other directories.

## 3. Devices

- UNIX allows devices to be used in much the same way as ordinary files.
- Hard Disks are block-oriented devices. They transfer data in blocks.
- modems and dumb terminals are **character-oriented** devices that transfer data on a byte-by-byte basis



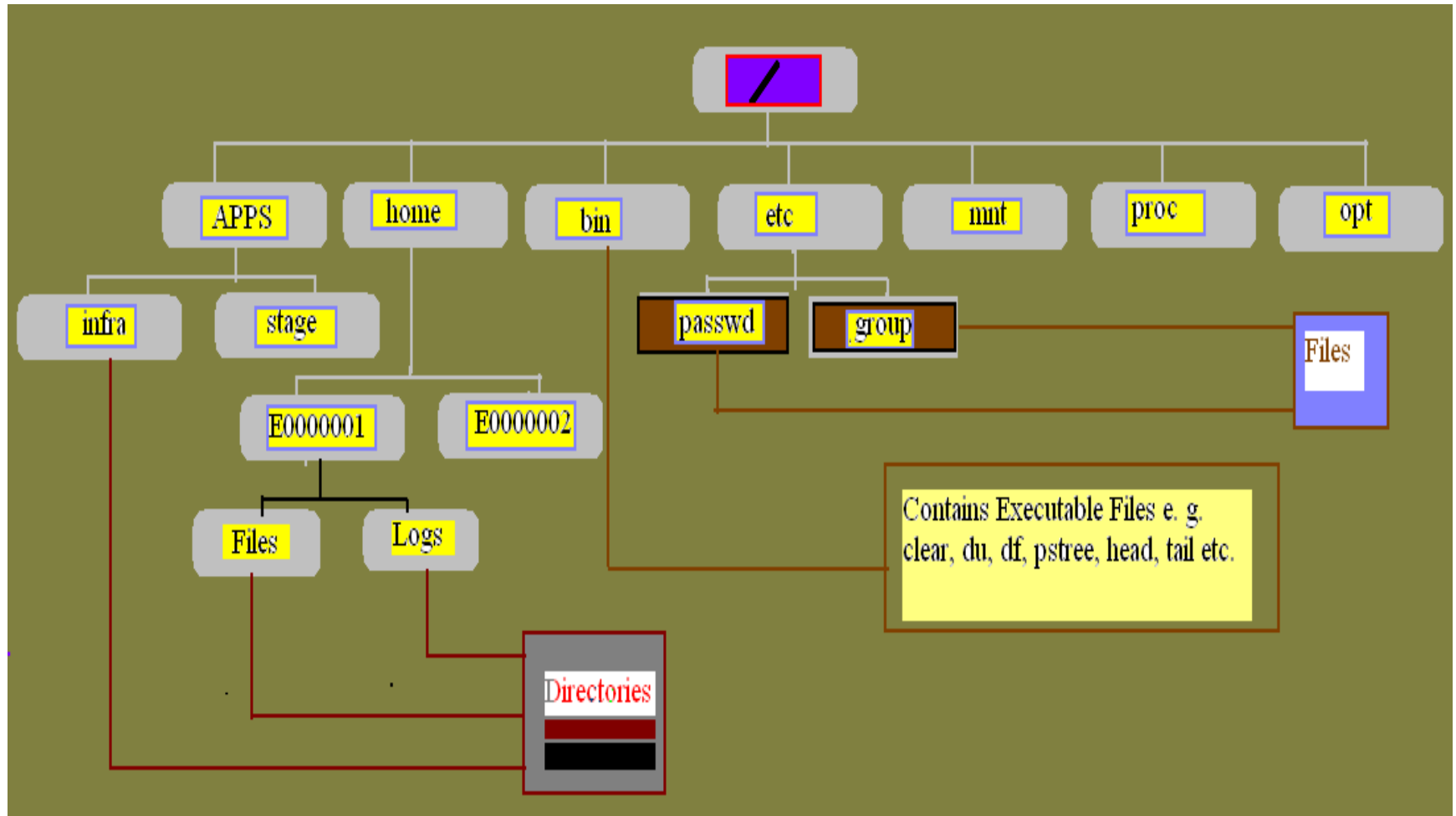
# The Unix File System Continued ...

---

## 4. Links:

- A link is a pointer to another file.
- There are two types of links - a **hard link** to a file is indistinguishable from the file itself.
- A soft link (or symbolic link) provides an indirect pointer or shortcut to a file.
- A soft link is implemented as a directory file entry containing a pathname.

# The Unix Directory Structure



# Try the following

---

- Look in /bin, /usr/bin, /sbin, /tmp and /boot.
- Explore /dev
- Explore /proc
- Display the contents of the files interrupts, devices, cpuinfo, meminfo and uptime using cat.
- Note : /proc is a pseudo-filesystem which allows access to Kernel

# File Paths

---

- Absolute Path starts from the root /.
- Relative Path starts from the Current Working Directory.
- Each Directory from source to the Destination should be included in the Path. Each Directory is separated by slash. e. g. /APPS/stage/E0000001/Logs
- . Represents current Directory and .. Represents parent Directory.

# Typical Directories

Directory	Typical Contents
/	The "root" directory
/bin	Essential low-level system utilities
/usr/bin	Higher-level system utilities and application programs
/sbin	Superuser system utilities (for performing system administration tasks)
/lib	Program libraries (collections of system calls that can be included in programs by a compiler) for low-level system utilities
/usr/lib	Program libraries for higher-level user programs
/tmp	Temporary file storage space (can be used by any user)
/home or /homes	User home directories containing personal file space for each user. Each directory is named after the login of the user.
/etc	UNIX system configuration and information files
/dev	Hardware devices
/proc	A pseudo-filesystem which is used as an interface to the kernel. Includes a sub-directory for each active program (or process).

# Directories..

- When you log into UNIX, your current working directory is your home directory. You can refer to your home Directory as ~
- When you list Directories using command `ls -al` Each line of the output looks like this.

The diagram shows a single line of output from the `ls -al` command: `drwxrwxr-x 3 pcenter autodw 4096 Apr 27 09:08 SIND`. Each field is enclosed in a red oval, and a vertical line connects each oval to a label above or below it. The labels are: Permissions (above the first oval), Owner (above the second oval), Size(Bytes) (above the fourth oval), Date (below the fifth oval), and Name (below the sixth oval). Additionally, there are labels below the first three ovals: Type (below the first oval), Links (below the second oval), and Group (below the third oval).

Permissions	Owner	Size(Bytes)	Date	Name
drwxrwxr-x	3 pcenter	4096	Apr 27 09:08	SIND

# Where

---

- Type - is a single character which is either 'd' (directory), '-' (ordinary file), 'l' (symbolic link), 'b' (block-oriented device) or 'c' (character-oriented device).
- permissions is a set of characters describing access rights.
- There are three access types 1) read('r') 2) write('w') and 3) execute('x').
- There are three user categories
  - 1) Owner – Owns the file.
  - 2) Users in the group that the File Belongs to and
  - 3) other users (the general public).

# Where

---

- An 'r', 'w' or 'x' character means the corresponding permission is present; a '-' means it is absent.
- links refers to the number of filesystem links pointing to the file/directory
- owner is usually the user who created the file or directory.
- group denotes a collection of users who are allowed to access the file according to the group access rights specified in the permissions field.
- size is the length of a file, or the number of bytes used by the operating system to store the list of files in a directory.



# Where

---

- date is the date when the file or directory was last modified (written to). The -u option display the time when the file was last accessed (read).
- name is the name of the file or directory.

# File and Directory Permissions

Permission	File	Directory
read	User can look at the contents of the file	User can list the files in the directory
write	User can modify the contents of the file	User can create new files and remove existing files in the directory
execute	User can use the filename as a UNIX command	User can change into the directory, but cannot list the files unless (s)he has read permission. User can read files if (s)he has read permission on them.

- The permissions are read(r), write(w) and execute(x).
- The three categories of users are user/owner(u), group(g) and others (o).
- Only owners or the super user (root) can modify File and Directory permissions using **chmod** system utility.

# File and Directory Permissions.....

---

- `chmod -options File[Directory] name`
- `chmod g+r sales.txt` --- Allow Group to Read.
- `chmod u+w sales.txt` – Allow user to write.
- `chmod a+x sales.txt` –Allow everyone(user, group and others ) to execute.
- `chmod g+w, o-r sales.txt` – Allow group to write Disallow others to write.
- `chmod u=rw sales.txt` – Set user permissions to read and write.
- `chmod -R go+r sales_Dir` – Set permissions to all Files in the Directory recursively.

# Octal Numerical Representation

- **r**  $\longrightarrow$  **4**
- **w**  $\longrightarrow$  **2**
- **x**  $\longrightarrow$  **1**

For Example :

- $7 = 4+2+1 = r+w+x$
- $5=4+1 = r+x$
- $0=$  no permissions
- `chmod 750 sales.txt` is equivalent to
- `chmod u=rwx sales.txt`
- `chmod g=rx sales.txt`

# The sticky bit

---

- Any user who has write permission to a Directory can delete files in that directory regardless of who owns it, even if they can't read or write to the file.
- With sticky bit 't' set, only the owner of a file can delete it.
- `chmod +t <Directory>` → Set the sticky bit on a Directory.
- `chmod -t <directory>` → Remove the sticky bit from a Directory

# umask – user file-creation mode mask

- It is a four digit octal number.
- Unix uses this number to set permissions for newly created files and directories.
- On Many Unix Systems, the default umask is 022.
- umask command displays umask value.

Common umask Settings			
umask	User Access	Group Access	Other
0000	all	all	all
0002	all	all	read, execute
0007	all	all	none
0022	all	read, execute	read, execute
0027	all	read, execute	none
0077	all	none	none

# umask

---

- Korn shell, ksh allows to set umask value symbolically.
- `$ umask u=rwx,g=x,o=`
- `$ umask 067`

## **To Change group of a File or Directory.**

`$ chgrp group files`

It also supports `-R` option.

- Example:

`$chgrp -R bicoe sales_Dir`

# Inspecting File Contents

---

- `$ file filename(s)`
- `file` command analyzes a file's contents and reports a high-level description of what type of file it appears to be
- `$ head filename`
- Displays the first few lines in the File.
- `$tail filename`
- Displays last few lines in a file.
- `$ tail -20 sales.txt --` Displays last 20 lines  
`$ head -5 sales.txt –` Displays first 5 lines
- `tail` supports `-f` option to continuously monitor the last few lines.



# Find Files

---

- find can find files by type (e.g. -type f for files, -type d for directories),
- by permissions (e.g. -perm o=r for all files
- and directories that can be read by others),
- by size (-size) etc.
- You can also execute commands on the files you find.
- `$ find . -name "*.txt" -exec wc -l '{}' ';'`
- counts the number of lines in every text file in and below the current directory. The '{}' is replaced by the name of each file found and the ';' ends the -exec clause.

# Find Files

---

- `$ find sales_Dir -name sales.txt -print`
- Will Find the file sales.txt in the directory tree starting at sales\_Dir Directory.
- `$ find /home -name "*.txt" -print 2>/dev/null`
- Outputs any matching files in home directory with a full absolute or relative path. Here quotes are necessary to avoid filename expansion.
- `2>/dev/null` suppresses error messages (arising from errors such as not being able to read the contents of directories for which the user does not have the right permissions)

# Find - Options

---

- -type Locates a certain type of file. The most typical options for -type are as following:
  - **d** -Directory
  - **f** - File
  - **l** - Link
- Example:
- `find . -type d -print`
- `find . -type f -print | xargs ls -l`
- `find . -size +10000c -size -32000c -print`
- `find `gen_dir` -type f -size 0 -ls`

# Find – iname and xargs

- Use the -iname predicate to run a case-insensitive search
- `$ find . -follow -iname '*.htm' -print0 | xargs -i -0 mv '{}'`  
`~/Target_Dir`
- -print0 prints files with spaces.
- `find . ( -name a.out -o -name *.o ) -print`
- To search for all files with group write permission, and remove the permission, you can use
- `find . -perm -20 -exec chmod g-w {} ;`  
or
- `find . -perm -20 -print | xargs chmod g-w`

# Find – wholename option

- File name matches simple regular expression.
- In simple regular expressions the metacharacters '/' and '.' do not exist;
- Example:
- `find / -wholename '/proc' -prune \`
- `-o -perm -2 ! -type l ! -type s \`
- `! \ ( -type d -perm -1000 \ ) -print`
- Here prune excludes /proc Directory and its subdirectories.
- No Directories with sticky bit set. No socket or symlink

# Difference between `-exec` and `-xargs`

- `-exec` executes a program once per file.
- `-xargs` handles several files with each process.
- `-xargs` cannot handle files with spaces well.
- Find within find
- **find** can even call **find**. If you wanted to list every symbolic link in every directory owned by group "staff" you could execute:
  - `find `pwd` -group staff -exec find {} -type l -print ;`
- Find and Remove Files
  - `find . -type f -name "salesday.txt" -exec rm -f {} \;`

# Find and cpio

---

- cpio allows us to archive a stream of directories and files in a single archive.
- cpio archive can be compressed using gzip.
- Example:
- Archive entire directory tree using cpio.
- `$ find . -depth -print | cpio -ov > tree.cpio`
- List contents of cpio archive.
- `$ cpio -t < tree.cpio`
- Extract cpio files.
- `$ cpio -id < tree.cpio`
- We can use different find options as desired.

# cpio using find and copy

- `$ find . -depth -print0 | cpio --null -pvd new-dir`
- Copies files from one directory tree to another.
- Combines copy-out and copy-in steps without archiving.
- `find . -type f -name salesday.txt -print`
- Find files named salesday.txt. Exclude files in bkp Directory.
- `find . -type f -name salesday.txt -print -o -type d -name bkp -prune`
- cpio example:
- `find . -type f -name salesday.txt -print -o -type d -name bkp -prune | cpio -ov > sales_tree.cpio`



# ctime, atime, and mtime

---

- ctime      —————>      Change Time
- atime      —————>      Access Time
- mtime      —————>      Modify Time
- ctime :
- In Unix it is not possible to tell actual creation time of a File. ctime is the time when changes were made to a file or Directories' inode(owner permissions etc.).
- ctime is also updated when contents of a file change.
- Command to view ctime
- `$ls -lc`

# ctime, atime, and mtime

---

- atime
- Is the Time when Data of the File was last accessed.
- Displaying contents of the file or executing a shell script changes file's atime.
- Command to view atime
- `$ls -lu`
- mtime is the time when actual contents of the file were last modified.
- It does not change with owner or permission changes
- Command to view mtime
- `ls -l`

# ctime, atime, and mtime

---

- Examples

- `$ cat filename`  
file's atime is updated.

`$ chmod g+w filename`  
file's ctime is updated.

`$ echo "File contents" >> filename`  
file's ctime and mtime are updated.

# ctime, atime, and mtime

---

- The command to view inode number is
- `ls -li filename`
- `Stat` command can be used to view inode number and its attributes such as atime, mtime, ctime etc.
- `$stat Filename`

# Find - -ctime, -mtime and -atime

- Time is counted from the current moment. It is measured in 24 hour periods.
- Integer n without sign means exactly n 24-hour periods(days) ago.
- 0 means today.
- +n means older than n.
- -n means younger than n.
- -1 and 0 are the same. Both mean today.
- Example:
- `$ find . -mmin +5 -mmin -10`
- Files modified between 6 and 9 minutes ago.

# Find - -newer, -anewer, -cnewer

---

- -newer/-anewer/-cnewer reference\_file
- The time of modification, access time or creation time are compared with the same timestamp in the baseline file.
- If file is a symbolic link and the -H option or the -L option is in effect, the modification time of the file it points to is always used.
- Example:
- `$ find /usr !-newer /usr/ref_file -print`

# Find - -perm

---

- `$find . -type d -perm 777 -print`
- `find . -user root -perm -4000 -print`
- `find . -group 10 -perm -2000 -print`

# Find – which, locate

---

- *which* command is used to find location of a command, system utility or an application program.
- \$ which ls
- locate \*.txt
- Locates all .txt files in their full path.
- Locate stores filenames in an index.
- This index is updated only once in a day.
- Cannot find recently created Files.
- Reports Files as present even if deleted.



# Inode

---

- An inode is a data structure on Unix file system.
- An inode stores basic information about a regular file, directory, or other file system object.
- An inumber is the identification number of the inode.
- The inode table contains a listing of all inode numbers for a file system.
- About 1% of the total Disk space is allocated to the inode table.
- Every Time we create a File an inode is allocated to the file.
- df command shows the number of inodes used.

# Inode

---

- Inode holds information about the following attributes.
- Inode number
- File type
- Number of links to the File
- UID of the owner
- Group ID (GID) of the owner
- Size of the file
- Actual number of blocks that file uses
- Time last modified
- Time last accessed

# Inode

---

- Time last changed
- Access Control List(ACLs)
- Directories and Files are not treated differently in UNIX File system.
- Directory is a File containing other Files.
- Directories have additional settings in their inodes.
- Command to list inode information for a mount point
- `$ istat /usr/bin/ksh`

# Remove Files with inode number

- We cannot remove files created with control characters or special characters such as \*, ?, ^ etc.
- We have to use inode number to remove such file.
- `$ls -li filename`
- Command to remove file with control characters.
- `$ find . -inum [inode-number] -exec rm -i {} \;`
- Command to rename such File.
- `# find . -inum [inode-number] -exec mv {} filename\;`
- We can also use add \ character before special character in filename to remove it directly.

# Command to Write Protect a File

---

- `$ chattr +i Filename`  
To Remove Protection
- `$ chattr -i Filename`
- = causes them to be the only attributes that the files have.

# Finding Text in Files

---

- grep (General Regular Expression Print)
- grep options pattern files
- grep searches the named files for lines that match a given pattern.
- By default grep prints out the matching lines.
- Options:
  - -c -- Print the count of the number of lines that match
  - -i -- Ignore case
  - -v -- Print out the lines that do not match the pattern
  - -n -- Print out the line number before printing the matching line.

# Finding Text in Files

---

- find and grep can be combined.
- `$ grep hello `find . -name "*.txt" -print``
- Searches all the files in the Directory tree starting at the current Directory for lines containing the word 'hello'
- `[0-9]` matches any digit and `[^0-9]` matches any character that is not a digit
- The caret ``^`` and the dollar sign ``$`` are special characters that match the beginning and end of a line respectively. The dot ``.`` matches any character.

# Finding Text in Files

---

- `$ grep ^..[p-z]$ hello.txt`
- matches any line in `hello.txt` that contains a three character sequence that ends with a lowercase letter from p to z.
- Searching a string in Files with tree traversal.
- `$ grep -r "search string" /tmp`
- **egrep(extended grep):**
- Allows to join two regular expressions by `|`.
- Matches any string matching either sub expression.
- A regular expression may be followed by repetition operators.



# Finding Text in Files

---

- ``?'` means the preceding item is optional (matched at most once).
- ``*'` means the preceding item will be matched zero or more times.
- ``+'` means the preceding item will be matched one or more times.
- ``{N}'` means the preceding item is matched exactly N times.
- ``{N,}'` means the preceding item is matched N or more times.

# Finding Text in Files

---

- `{N,M}` means the preceding item is matched at least N times, but not more than M times.

For example, if `egrep` was given the regular expression

```
'(^[0-9]{1,5}[a-zA-Z ]+$)|none'
```

it would match any line that either:

begins with a number up to five digits long, followed by a sequence of one or more letters or spaces, or contains the word `none`

# Sorting Files

---

- sort filenames
- Sort sorts lines in a group of files alphabetically.
- With `-n` option sorts numerically.
- `$ sort sales_emp.txt supp_chain_emp.txt > emp.txt`
- Outputs sorted concatenation of files to emp.txt
- uniq filename
- uniq removes duplicate adjacent lines from a file.
- `$ sort emp.txt | uniq > employees.txt`

# File Compression and backup

---

- Unix systems support many utilities for backup and compressing files.
- tar (tape archiver)
- tar backs up entire directories and files onto a tape device or into a single disk file called archive.
- An archive File contains:
  - a) Other files.
  - b) Information about the files such as filename, owner, timestamps and access permissions.
- tar does not perform any compression by default.

# File Compression and backup

---

- To create a disk file tar archive
- `$ tar -cvf archivename filenames`
- Where
  - a) archivename may have .tar extension.
  - b) c  $\longrightarrow$  Create
  - c) v  $\longrightarrow$  Verbose(output filenames as they are archived)
  - d) f  $\longrightarrow$  File

# File Compression and backup

---

- To list contents of a tar archive
- `$ tar -tvf archivename`
- To restore contents of a tar archive
- `$ tar -xvf archivename`
- **cpio**
- cpio is another utility for creating and reading archives.
- Unlike tar, cpio doesn't automatically archive the contents of directories.

# File Compression and backup

---

- Commonly cpio is combined with find to create archives.
- `$ find . -print -depth | cpio -ov -Htar > archivename`
- This command archives all the files in the current Directory and the Directories below into archivename file.
- -depth option controls the order in which the files are produced.
- This option is recommended to avoid issues related to Directory permissions while doing a restore.
- -o option creates the archive.

# File Compression and backup

---

- -v option prints the names of the files archived as they are added.
- -H option specifies an archive format type.
- In this case it creates a tar archive.
- Another common archive format is crc, a portable format with a checksum for error control.
- To list the contents of a cpio archive use
- `$ cpio -tv < archivename`
- To restore Files use
- `$ cpio -idv < archivename`



# File Compression and backup

---

- Here `-d` option creates directories as necessary.
- `-u` option forces `cpio` to extract files on top of files of the same name that already exist and have same or later modification time.
- **compress and gzip**
- These utilities compress and decompress individual files.
- The Files may or may not be archive files.
- To compress files.
- `$ compress filename`
- `$ gzip filename`

# File Compression and backup

---

- In either case, file name gets deleted and is replaced by a compressed file filename.Z or filename.gz.
- To reverse the compression.
- `$ compress -d filename`
- `$ gzip -d filename`

# Pipes

---

- '|' Operator is used to create concurrently executing processes.
- These pass data directly to one another.
- `$ who | sort | uniq`
- Creates three processes corresponding to who sort and uniq.
- These execute concurrently.
- The output of who process is passed on to sort process, sort process passes it onto uniq process.
- uniq displays the output on the screen.
- Sorted list of users with duplicate lines removed.

# Pipes

---

- `$ cat hello.txt | grep "dog" | grep v "cat"`
- Find the lines in `hello.txt` that contain the string `"dog"` but do not contain the string `"cat"`

# Redirecting input and output

---

- Processes write to standard output, the screen.
- Processes take their input from standard input, the keyboard.
- Processes write their error messages to standard error.
- Error messages are sent to screen by default.
- To redirect standard output to a file, we use the > operator.
- `$echo hello`
- `$echo hello > outputfile`
- `$ cat outputfile`

# Redirecting input and output

---

- Redirection using `>` operator destroys contents of the file if the file already exists.
- If we want to append output of `echo` command to the file we can use the `>>` operator.
- `$echo bye >> outputfile`
- `$ cat outputfile`
- To capture standard error, prefix `>` operator with `2`.
- `$cat nofile 2> errors`
- `$ cat errors`

# Redirecting input and output

---

- In UNIX the file numbers are assigned as below:
- 0 —————> standard input
- 1 —————> standard output
- 2 —————> standard error
- Standard Error and Standard output can be output to two different files.
- `find . -print 1>files 2>errors`
- Or the same file
- `find . -print 1>output 2>output` or
- `find . -print >& output`

# Redirecting input and output

---

- `<` Operator is used to redirect standard input.
- In this case input is read from file instead of the keyboard.
- `$cat > filename`
- We can combine input redirection and output redirection
- `$ cat < output_file > output_file`
- Avoid use of the same file name in both places.
- Will Destroy the contents of the file.



# Redirecting input and output

---

- When the shell sees the `>` operator, it creates an empty file ready for the output.
- We can pass output to system utilities that require filename as “-”.
- `$ cat all_files.tar.gz | gzip -d | tar tvf -`
- Here the output of the `gzip -d` command is used as the input file to the `tar` command.

# Connecting to Remote Machines

---

- C:> ping machinename
- C:> telnet machinename
- C:> ssh machinename
- ssh is not a standard system utility.
- rlogin and rsh are not secure utilities.
- ssh creates a secure encrypted communication with remote machine over an insecure network
- Putty is a commonly used ssh client on windows.

# Remote File transfer

---

- ftp machinenname (file transfer protocol)
- ftp is an insecure way of transferring files between computers.
- You can use your username and password to login to remote host.
- ftp also supports ftp user and anonymous user.
  - You can list Files. Example dir.
  - Receive Files get and mget
  - Send Files put and mput
  - binary file transfers Files preserving all 8 bits.

# Remote File transfer

---

- \$ prompt n → Do not confirm each file on multiple file transfer.
- \$ quit to leave ftp and return to the shell prompt.
- \$ scp sourcefiles destination (secure copy)
- scp is a secure way of transferring files between machines.
- It works like cp command.
- \$ scp greatuser@eta11.tcc.etn.com:~/hello.txt .
- Subject to correct authentication, copies the file hello.txt from the user account greatuser to current directory . on the local machine.

# User Information and Communication

---

- `$ finger`
- `$who`
- `finger` and `who` commands show the list of users logged into a machine, the terminal they are using and the date they logged on.
- `write scott pts/2 <Enter>`
- `Hello scott<Enter>`
- Lines are only transmitted when you press `<Enter>`
- To return to the shell prompt, press `ctrl + d` (the UNIX end of file marker).
- `talk` and `ytalk` work the similar way.

# Email Utilities

---

- mail is a standard UNIX utility for sending and receiving email.

\$ mail<Enter>

- We can send email directly from the command line.

- For Example:

- \$ mail -s <"Subject Here"> <Email Address> <Message File>
- \$ mail -s "Hi" xyz@abc.com < message.txt
- mutt, elm, pine are some other non standard email utilities.

# Vi (Vee Eye) Editor

---

- The current iteration of vi for Linux is called vim
- **Vi**l**m**proved

## Starting vi

- Type vi <filename> at the shell prompt.
- Press Enter.
- Command prompt disappears and you see tilde(~) characters on all the lines.
- Tilde character indicates that the line is blank.

# Vi Modes

---

➤ There are three modes in vi.

☐ Command Mode

☐ Input Mode and

☐ Last-line Mode

➤ By default vi opens in command mode.



# Vi Modes

---

- **Command Mode:**
  - You can navigate the file and use commands.
- **Insert Mode:**
  - You can type into the file. With vim you can still move around the file.
- **Last Line Mode:**
  - You can issue command on the last line of the editor.

# Vi Commands

- Editing

Command	Function
i	Insert
o	Open a new line (below)
O	Open a new line (above)
a	Append
A	Append at end of line
u	Undo
U	Undo All
.	repeat last command

# Vi Commands

- **Cutting and pasting**

Command	Function
yy	Yank (copy)
5yy	Yank 5 lines
dd	Delete current line
6dd	Delete six lines
p	Paste (below current line) (lower-case 'p')
P	Paste (above current line)(capital 'P')

# Vi Commands

- **Deleting**

Command	Function
x	Delete current character
10x	Delete 10 characters
dd	Delete current line
10dd	Delete 10 lines
d0	Delete to beginning of line
d\$	Delete to end of line

# Vi Commands

- Navigation – up, down

Command	Function
[Up]	Move up one line
5[Up]	Move up 5 lines
[Down]	Move down 1 line
9[Down]	Move down 9 lines
1G	Go to line 1
15G	Go to line 15
G	Go to last line
H	Go to top of screen (high)
M	Go to middle of screen
L	Go to bottom of screen (low)

# How to Exit from Vi(Command Mode)

- **Navigation – left, right**

Command	Function
w	Go to next word
7w	Move over 7 words
b	Back one word
0	Go to beginning of line
\$	Go to end of line

# How to Exit from Vi(Command Mode)

- **Searching**

Command	Function
/my_string	Search forward for "my_string"
?my_string	Search backward from "my_string"
n	Repeat the last search

# Vi Commands

- **Miscellaneous**

Command	Function
<code>:!ls</code>	Run "ls" command from editor
<code>:r file1</code>	Read file file1 into this file
<code>:10,20d</code>	Delete lines 8-15
<code>:1,\$s/old/new/g</code>	From the First line to the last line change all occurrences of "old" with "new"



# How to Exit from Vi(Command Mode)

---

- `:q <Enter>` Exit without saving changes. If you have not made any changes to the file.
- `:q! <Enter>` Force quit. Discard changes and quit.
- `:w<Enter>` Write contents to disk.
- `:wq<Enter>` Save and Exit.
- `:x<Enter>` Save and Exit
- `ZZ(Upper Case)` Save and Exit
- `!` Character forces over writes e. g. `wq!`

