# Unit 1

Overview of Linux

## History Of Linux

- In 1969 ken Thompson and Dennis Ritchie developed small general Purpose OS called UNIX.
- First it was developed using Assembly language,In 1973 it was rewritten using C by the same people.

## Ken Thompson & Dennis Ritchie

## History of Linux contd…

- In 1974 UNIX was licensed to universities for educational purposes and made commercially available.
- Many vendors like IBM,Sun Purchased the source code of UNIX developed their own version of unix.The souce code was not freely available to others.

# History of Linux con .

- In 1984 Richard Stallman’s free software Foundation(FSF) began GNU(GNU’s Not UNIX).Their symbol is given above.
- The aim was to create free version of UNIX(General Public license: License to modify source code).Many tools were developed by GNU. Kernal was not developed.

## History of Linux contd…

- In 1991 Linus Tovalds developed Linux kernal.
- In 1992 Linux kernal was combined with Incomplete GNU to form a complete OS.
- This Os is known as GNU/Linux Operating System.

# Linux History

- 1969 Ken Thompson and Dennis Ritchie developed a small general purpose os called UNIX. Written in assembly language. 1973 rewrote the UNIX in C.
- UNIX was licensed to universities for educational purposes and made commercially available later.
- 1984 Richard Stallmans Free Software Foundation began the GNU project to create a free versions of the UNIX os.
- 1991 Linus Torvalds developed a kernel and called it Linux.
- 1992 Linux kernel was combined with incomplete GNU system to form a completely free os.

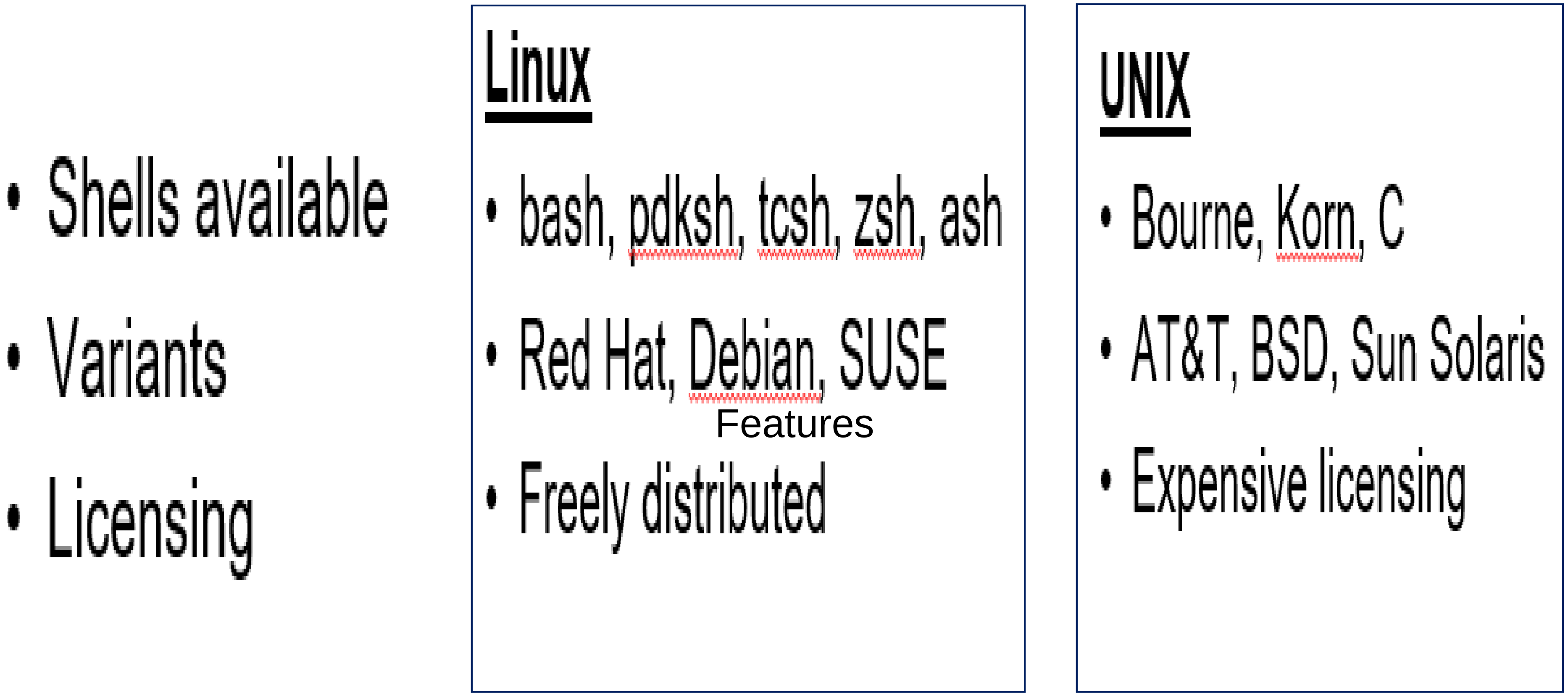
## What is Linux

- Linux is a freely distributed implementation of UNIX-like kernel.
- Linux & UNIX programs are very similar
- Almost all programs written for UNIX can be compiled & run on Linux.

- linux is kernel , Unix is complete os
- linux-free , unix-costly
- linux-optional CUI, unix permanent CUI

character user interface CUI.
 CUI stands for character user interface and can also be referred to as the command prompt, C prompt or command line. In the direct operating system called DOS this is generally displayed on the screen by a "C:>" or "C>".

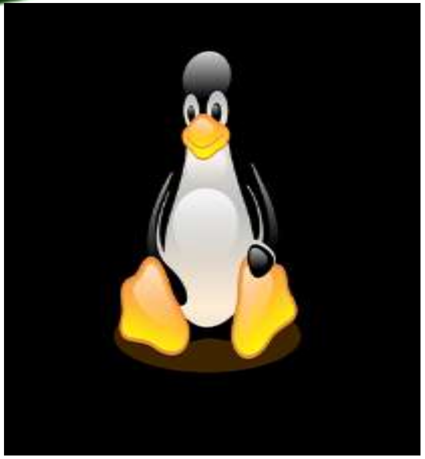
## Features



|  | Linux | UNIX |
|---|---|---|
| Shells available | bash, pdksh, tcsh, zsh, ash | Bourne, Korn, C |
| Variants | Red Hat, Debian, SUSE | AT&T, BSD, Sun Solaris |
| Licensing | Freely distributed | Expensive licensing |

Features

## LINUX IS THE STAR



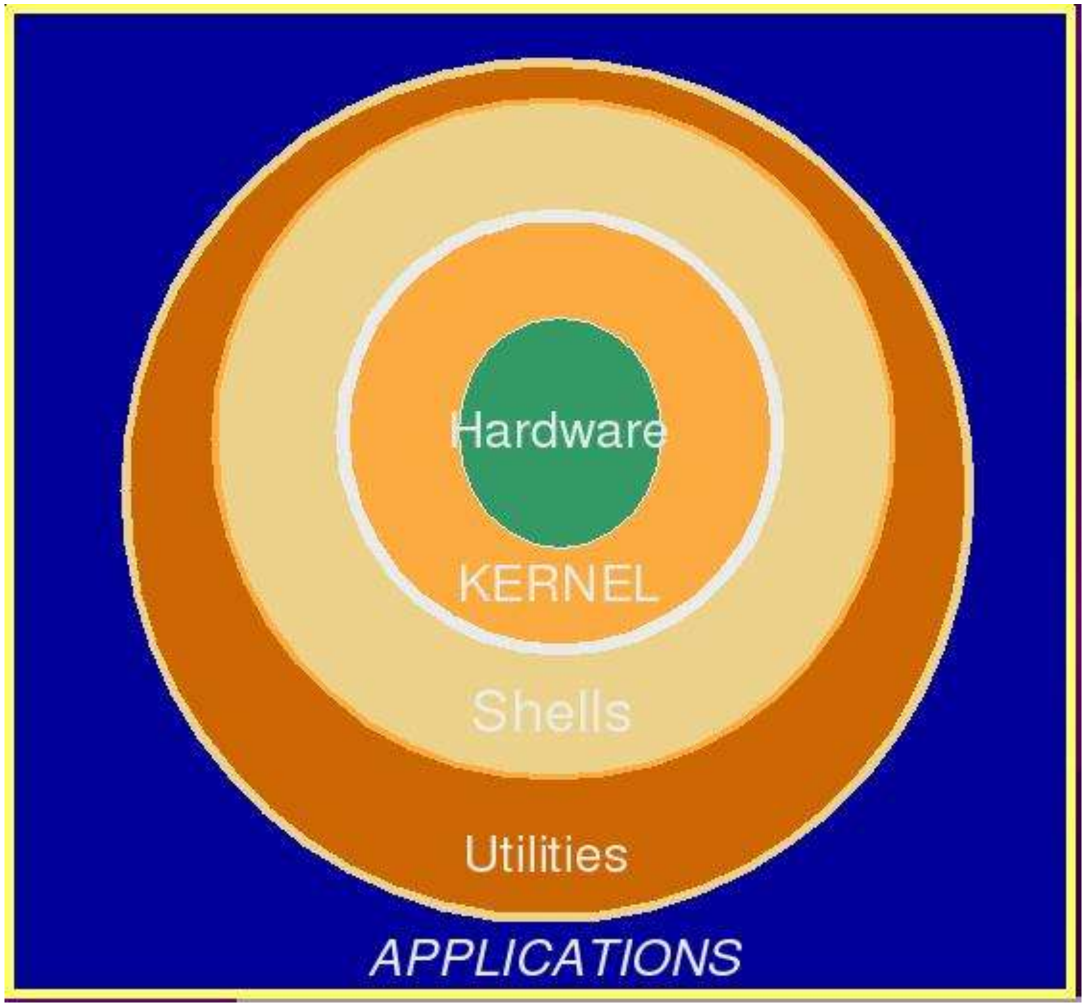
- 94 % of worlds super computers are  powered by linux.
- *Most of servers powering INTERNET*
- *Majority of financial traders*   Uses linux
- *A billion android devices*
- Many different architectures from mini → mainframe → server → desktop → mobile

Ie  linux is everywhere today

# Features and Utilities in Linux

- Multiprogramming
- Time-Sharing
- Multitasking
- Virtual Memory
- Multiuser
- Samba
  - Server Message Block or AMB. SMB is the protocol used by Microsoft os to share files and printers.
  - Samba is a suite of programs that implement the SMB protocol in Linux.
- Cron Scheduler
- Licensing
- Web Server

# Architecture of Linux



- **Kernel**

- The core of the Linux system is the *kernel, which* is also the operating system program.
- The kernel controls the resources of a computer, allocating them to different users and tasks. It interacts directly with the hardware, making programs easy to write and portable across different hardware platforms.
- However, the user does not interact directly with a kernel. Instead, the logon process initiates a separate, interactive program called the *shell* for each.
- The kernel is the heart of the system. It manages the communication between the underlying hardware and the peripherals.
- The kernel also makes sure that processes and daemons (server processes) are started and stopped at the exact right times.

- **Shell**

  Linux has a simple user interface called the shell. The shell provides services for a user. A user interacts with the computer by using the shell. The user need not know about the details of the hardware. Some of the common shells in linux are bash, sh, tch, csh and ksh.

- **System Utilities**
  - Linux utilities or commands are a collection of programs that service processing requirements. These programs can be started by using the shell. This includes commands such as ls, cp, grep, awk, sed, bc, wc, more, and so on.
  - These system utilities are designed to be powerful tools that do a single task extremely well (e.g. grep finds text inside files while wc counts the number of words, lines and bytes inside a file).
  - Users can often solve problems by interconnecting these tools instead of writing a large monolithic application.
  - Linux's system utilities also include server programs called **daemons** which provide remote network and administration services. A daemon is usually spawned automatically at system startup and spends most of its time lying dormant waiting for some event to occur.

- **Application programs**

  Linux distributions typically come with several useful application programs as standard. Examples include the emacseditor, xv (an image viewer), gcc (a C compiler), g++ (a C++ compiler), xfig (a drawing package), latex (a powerful typesetting language) and soffice (StarOffice, which is an MS-Office style clone that can read and write Word, Excel and PowerPoint files). Redhat Linux also comes with rpm, the Redhat Package Manager which makes it easy to install and uninstall application programs.

## Linux File System

- A file system is a combination of methods and data structures, which is also useful for any operating system to keep track of files on a disk or partition and states the way the files are organized on the disk. The central concepts of a file system includes:
  - Boot block
  - Super block
  - Inode block
  - Data blok

- **Boot Block** – This represents the beginning of the file system. It contains a program called ‘bootstrap loader’. This program is executed when we boot the host machine. Although only one boot block is needed to start up the system, all file systems contain one (possibly empty) boot block.
- **Super Block** – The super block describes the state of the file system – how large it is, how many maximum files can it accommodate, how m It contains info about

  1. type of file system (ext2, ext3...)

  2. the block size

  3 pointers to a list of free blocks

  4. the inode number of the root directory

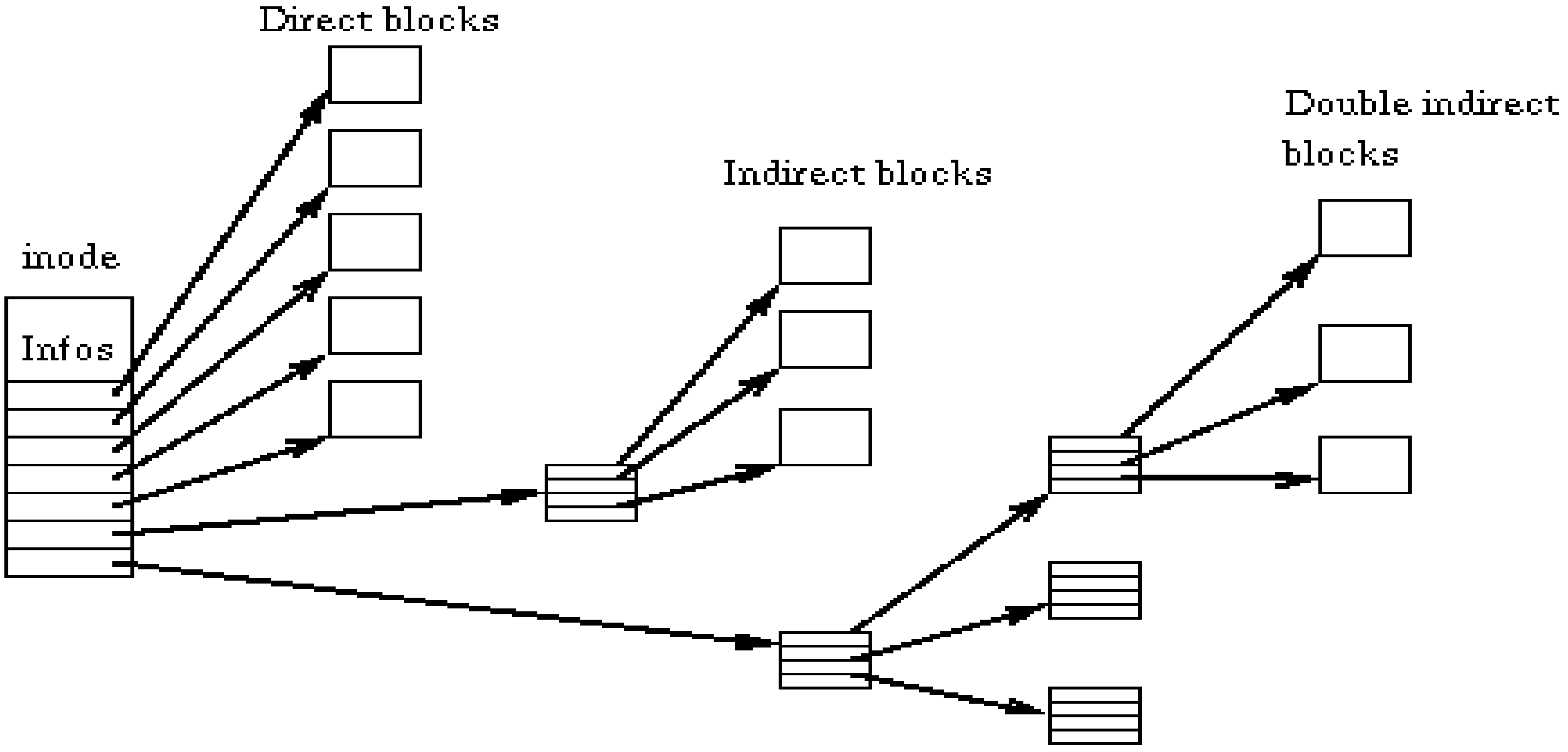
- Inode Table –
- All the entities in linux are treated as files. The information related to all these files (not the contents) is stored in an Inode Table on the disk.
- An inode contains all the information about a file, except its name. The name is stored in the directory, together with the number of the inode.
- A directory entry consists of a file name and the number of the inode which represents the file. The inode contains the numbers of the data blocks that are used to store the data in the file. It contains:

- For each file, there is an inode entry in the table. Each entry is made up of 64 bytes and contains the relevant details for that file. These details are
  - Owner of the file
  - Group to which the owner belongs
  - Type of the file
  - File access permissions
  - Date and time of last access
  - Date and time of last modification
  - Number of links to the file
  - Size of the file
  - Address of blocks where the file is physically present

- Data Blocks – These contain the actual file contents. An allocated block can belong to only one file in the file system. This block cannot be used for storing any other file’s contents unless the file to which it originally belonged is deleted.

## Data Blocks

data blocks are part of inodes, which contain the information stored in the individual files.

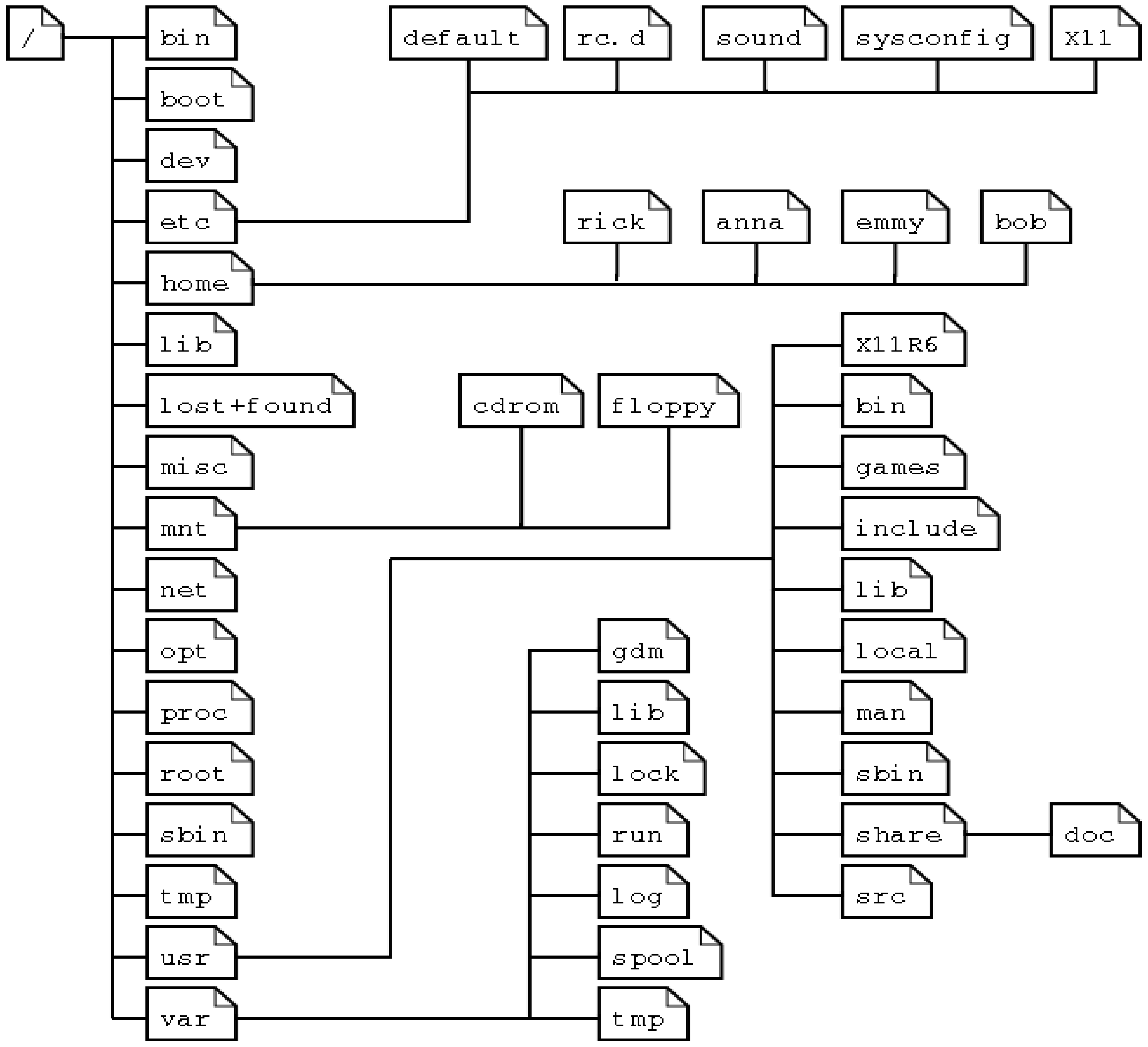


## Surrogate super block and Inode Table

- Whenever we use any file or change its permissions corresponding changes should be made in super block and inode table.
- Making changes on the disk will take a lot of precious cpu time. To remedy this a copy of super block and node table gets loaded into memory (RAM) at start-up time.
- Since memory access is faster than disk access, a lot less time is consumed in recording the changes in the RAM copies of Super Block and Inode Table every time some modification occurs.
- The original Super Block and Inode Table on the disk are updated after a  fixed interval of time by a command called sync.

## Linux Standard Directories

- Linux disks are organised into a logical tree structure formed of directories. Each directory is a container which can hold files (programs or data) and other directories. The base of the tree is known as the root. This is represented by the / symbol. Linux file system layout is given below.


/
bin
boot
dev
etc
home
lib
lost+found
misc
mnt
net
opt
proc
root
sbin
tmp
usr
var
default
rc.d
sound
sysconfig
X11
rick
anna
emmy
bob
cdrom
floppy
X11R6
bin
games
include
lib
local
man
sbin
share
doc
src
gdm
lib
lock
run
log
spool
tmp

Some of the root level directories found on all linux systems are:

- **/** - this is referred as root directory. It is the baseline for the directory structure, which is important when navigating from one directory to another.(not get confused with /root directory which is the root users directory).
- **/bin** – contains executable programs(binary files) which make up the commands of the operating system.
- **/boot** – contains the boot information for linux, including kernel.
- **/dev** – in linux, devices are treated in the same way as files, so they can be read from and written to in the same way. So when a device is attached, it will show up in this folder. It includes every device in the system, including motherboard, mouse etc.

- **/etc** – the default configuration files for linux applications are stored in this directory. These text files can be easily edited and sometimes they can be overridden by copies of the files elsewhere on the system.
- **/home** – contains the home directories of users on the system. That is, the directory a user finds himself in when he logs on, and in which he has full ownership permissions.
- **/lib** – contains shared libraries(shared program files) for use by applications running on linux, simalr to windows DLL files.

- **/lost+found** – this is linux's rescue directory, so that in the event of a crash or other serious event files are stashed(hide or store in a secret place) here to enable recovery.
- **/misc** – for miscellaneous purposes.
- **/mnt** – in linux, every storage device is treated as just another directory. This included floppy disks, hard drives, CD/DVD ROMs and USB card devices. This is the directory in which storage devices are mounted.
- **/net –** standard mount point for entire remote file systems.
- **/opt –** contains extra and third party softwares.

- **/proc** – this is not a real directory but it is a virtual directory simulated by the operating system. It shows many pieces of information about the hardware and software of the system.
- **/root** – the home directory of the superuser “root”. This is not the same as root directory of the system(/).
- **/sbin** – contains programs for use by the system & system administrator. Unlike the applications in the /bin folder, the root user is usually the only user who can run these.

- **/tmp** – directory available to all users and programs for storing temporary files and data.
- **/usr** – contains user’s application like executables, source code along with any images and documentation.
- **/var** –storage for all variable files and temporary files created by users like log files, print spool files, temporary storage of file downloaded from internet etc. this can be viewed in the event of a system error to help in diagnosing the problem.

## Installing requirements of Linux

- Before installation always check your hardware as all Linux flavours come with a basic package and later can be upgraded as per the system requirements using the Linux Kernel. You only need to ensure that your flavour will run on your hardware.
- The list of configuration details needed to know before installation are:
  - ➢ **CPU –** Find out the model(Intel, AMD etc), processor speed, type of cpu.
  - ➢ **Hard disk** – Find out the details like make, manufacturer, size. It is always helpful if you know the size of the windows partitions present and the amount of the space you want to give to Linux.

- **CD-ROM drive** – This is one of the first requirements before installation. If you have a relatively new drive, chances are you may not face any problem. However, if the drive is older, it will not be auto-detected. So, you will have to get the port number on which it works.
- **Display card** – This is required for X-Windows, the graphical feature of Linux.

# Linux requirements

| Requirement | Minimum | Recomme nded |
|---|---|---|
| Version | Linux 2.2 and 2.4 | |
| CPU | 400 MHz | 800+ MHz |
| RAM | 32 MB<br>96 MB (live broadcasting) | 256 MB |
| Hard Disk space (software) | 20 MB | |
| Hard Disk space (data) | 500 MB | 1 GB |

## Installation Methods

- **CD-ROM –** Linux is available in CD-ROM and is the best medium to install.
- **Hard Drive** – Linux is free to download from the internet and if you have downloaded the Linux ISO(International Organization for Standardization) images to your local hard drive, you can also install from that.
- **NFS image** – Linux is so hardware driven that you can also install from an NFS server using ISO images or a mirror image of Linux.
- **FTP** – Linux also allows you to install directly from an FTP server, just downloaded it and install as per your requirement.

- **An ISO is an Exact Image of a CD, that can be burned again into a disc to get a cloned copy of the original disk**
  **usually ISO images can be anything between 1 to 700MB**
  **so don't consider downloading the ISO if you have a slow connection, unless you are planning to stay online for 2 weeks downloading, especially if you are on a dial-up connection!**
- **In order to burn an ISO Image onto a CD, you need a CD Writer, and CD Writing Software.**
- **Network File System (NFS)**
- NFS allows a system to share directories and files with others over a network. By using NFS, users and programs can access files on remote systems almost as if they were local files.

## File processing commands

- **pwd**  - will display the current directory

  Eg : $ pwd  o/p : /home/bca08b/magtest

- **cd** – used to change a directory. This command will take you to the location you specify as argument.

  **Synatx** :  $ cd <location>

  **Eg** : cd hello – this will change to the directory named hello located inside the current directory

  $ cd /home/games – this will change to the directory called games within the home directory.

$ cd ~ - this will take directly to home directory

$ man cd – cd command details

- **cp –** copies files and directories. You can copy a file within the current directory to another directory.

  **syntax :** $ cp <source> <destination>

  **Eg :** $ cp myfile.txt   /home/help/newfile.txt – this will copy myfile.txt in the current directory to the directory /home/help/ and will rename it as newfile.txt (if u don't want to rename then no need to specify any filename(/home/help/)

$cp –R  /home/user1/mydir   /home/user1/backup/ – this command is used to copy mydir directory from /home/user1 to/home/user1/backup

Or

$cp –R  ~/mydir   ~/backup/

- **rm** – used to delete files

  syntax : rm <filename>

  eg :
  - ➢ rm myfile.txt – will delete myfile.txt and will ask your permission before deleting
  - ➢ rm –f myfile.txt – file will be deleted forcefully without asking permission
  - ➢ rm –rf mydir – deletes the directory structure fully

  (confusion in rm switches not asking permission)

- **mkdir –** used to create a new directory

  Syntax : $ mkdir <name of directory>

  Eg: $ mkdir newdir – creates new directory newdir in the current directory.

- **rmdir –** used to delete an empty directory

  **Syntax :** rmdir <dir name>

  Eg : rmdir newdir

- **file** – used to get more details about a file like file name, type, size etc

  Eg : file img_small.png

  o/p : img_small.png : png image data, 128X151, 8 bit/color RGB, non-interlaced

- **more** – this command allows you to scroll through a file one screen at a time, allowing you to read the file’s contents more conveniently. Some files are too big, and with this command you can view the contents of large files more efficiently. To move one screen ahead, use the spacebar, and to go one screen back, use B key.

  **Syntax** : more <filename>

- **less** – this command allows you to scroll through a file, a page at a time. The less command is similar to more command, only it is more advanced and has more enhanced features.

  **Syntax** : less <filename>

- **mv** – is used to move files or directories from one location to another or rename a file or directory.

  **Syntax** : $ mv <source> <destination>

  Eg : $ mv /home/usr1/newfile   /home/usr1/mydoc

- **find** - is used to locate files or folders within a Linux System.

  Eg : find /usr/bin –name <filename> - this will search inside the /usr/bin directory (and sub directories) for the file named filename

- **cat** – allows you to create and manipulate files. we can create, view, edit and append the file content. This command is mainly used in input/output redirection.

cat file1.txt – to view the contents of file1.txt. This command will display the contents if file1.txt to the screen

- Cat can be used to concatenate the contents of two files and store them in a third file.

  Eg:- cat sample1 sample2 > newsample

- If newsample contains something it would be overwritten.
- If we want that it should remain intact then use ‘append output redirection operator’ >>.

  Eg:- cat sample1 sample 2 >> newsample

- touch – used to create an empty file

  Syntax : $ touch <filename>

  Eg : touch myfile

  touch file1,file2,file3 – to create multiple files

- ls – used to see the contents of the current directory.

  syntax : $ ls

This command can also give listings of other directories without going to those directories.

  Syntax : $ ls /dev/bin – this will list the contents dev/bin directory

Some of the common switches in ls command are :

- ls -a : this will list all the files including those beginning with '.' that will be hidden files
- Ls -l : long listing with file attributes and permissions

```
-rwxr-x—x   1   user1   group   24 Jun 06 10:12   carribeans
```

- ls -s : listing showing the size rounded up to the nearest kilobyte
- ls -S : list the files according to the file size
- ls -C : listing displayed in columns
- ls -F : this gives a symbol next to each file in the listing showing the file type. The / means it is a directory, the * implies an executable file and the @ implies a symbolic link
- ls -r : listing in reverse order
- ls -R : recursive listing of all directories
- ls -t : lists directory according to time stamps
- ls -la : lists all the file in long listing format including hidden file

- ln
  - ln poem newpoem creating new link to poem.
- When we create directory will create 2 links.

Eg:- if we create a directory dir1 in a directory aa1 the directory file aa1 would have an entry dir1 where as the directory file dir1 itself would also have an entry dir1.

- 3 types of permissions to a file.

1. Read(r)
2. Write(w)
3. Execute(x)

rwxr-x—x :-

rwx:- the owner can read, write as well as execute.

r-x:- the members of the group can read and execute the file but cannot write.

--x:- all others can only execute the file

- These permissions can be encoded numerically.
  - Read(r) 4
  - Write(w) 2
  - Execute(x) 1
- Owner:- 4+2+1=7
- Others in group:- 4+0+1=5
- Others:-0+0+1=1
- We can say file has the permission 751.
- When everybody has full permission, they would amount to 777.

- The existing file permissions can be changed by the owner of the file or by the super user.
  - chmod 700 myfile then permission becomes rwx------.
  - This is absolute mode.
  - chmod [who] [=/-/=] [permissions] file
  - Who means to whom the permissions are to be assigned. It may be the user or owner(u), the group(g) or others(o).
  - If none is specified all are assumed.
  - + refers to add permission, - refers to remove permission and = instructs chmod to add the specified permission and take away all others, if present.

## File related commands

- Wc
  - wc, or "word count," prints a count of newlines, words, and bytes for each input file.
  - wc [OPTION]... [FILE]...
  - **wc** prints newline, word, and byte counts for each *FILE*, and a total if more than one*FILE* is specified.

## Options

| | |
|---|---|
| **-c**, **--bytes** | print the byte counts. |
| **-m**, **--chars** | print the character counts. |
| **-l**, **--lines** | print the newline counts. |
| **-L**, **--max-line-length** | print the length of the longest line. |
| **-w**, **--words** | print the word counts. |
| **--help** | display a help message, and exit. |
| **--version** | output version information, and exit. |

- wc myfile.txt
- Displays information about the file myfile.txt. Output will resemble the following:
- 5 13 57 myfile.txt
- Where 5 is the number of lines, 13 is the number of words, and 57 is the number of characters.

# sort

- **sort** sorts the contents of a text file, line by line.
- sort is a simple and very useful command which will rearrange the lines in a text file so that they are sorted, numerically and alphabetically. By default, the rules for sorting are:
- lines starting with a number will appear before lines starting with a letter;
- lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet;
- lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.
- The rules for sorting can be changed according to the options you provide to the sort command; these are listed below.

| | |
|---|---|
| **-b**, **--ignore-leading-blanks** | Ignore leading blanks. |
| **-d**, **--dictionary-order** | Sorts in dictionary order |
| **-f**, **--ignore-case** | Fold lower case to upper case characters. |
| **-g**, **--general-numeric-sort** | Compare according to general numerical value. |
| **-c** | Checks if files are already sorted. If they are, sort nothing |
| **-M**, **--month-sort** | Compare (unknown) < `**JAN**' < ... < `**DEC**'. |
| **-m** | Merges files that have already been sorted |
| **-n**, **--numeric-sort** | Compare according to string numerical value. |
| **-R**, **--random-sort** | Sort by random hash of keys. |
| **-tc** | Separates fields with character(default is tab) |
| **-r**, **--reverse** | Reverse the result of comparisons. |
| **-u** | Unique output: if merge creates identical lines uses only the first |
| **-V**, **--version-sort** | Natural sort of (version) numbers within text. |

# cut

- Linux command cut is used for text processing. You can use this command to extract portion of text from a file by selecting columns.
- cut [options] [filename/s]
- When invoking **cut**, use the **-b**, **-c**, or **-f** option, but only one of them.
- -b, --bytes=LIST Select only the bytes from each line as specified in LIST. LIST specifies a byte, a set of bytes, or a range of bytes; see Specifying LIST below.
- -c, --characters=LIST Select only the characters from each line as specified in LIST. LIST specifies a character, a set of characters, or a range of characters; see Specifying LIST below.
- -f, --fields=LIST select only these fields on each line; also print any line that contains no delimiter character, unless the -s option is specified. LIST specifies a field, a set of fields, or a range of fields; see Specifying LIST below.
- -d, <clumn_delimiter> Specifies the column delimiter.

- For example, let's say you have a file named data.txt which contains the following text:

```
one	two	three	four	five
Alpha	beta	gamma	delta	epsilon
```

- In this example, each of these words is separated by a tab character, not spaces. The tab character is the default delimiter of cut, so it will by default consider a field to be anything delimited by a tab.
- To "cut" only the third field of each line, use the command:
- cut -f 3 data.txt
- ...which will output the following:

```
three
gamma
```

- cut -f 2-4 data.txt
- ...which will output the following:

```
two	three	four
beta	gamma		delta
```

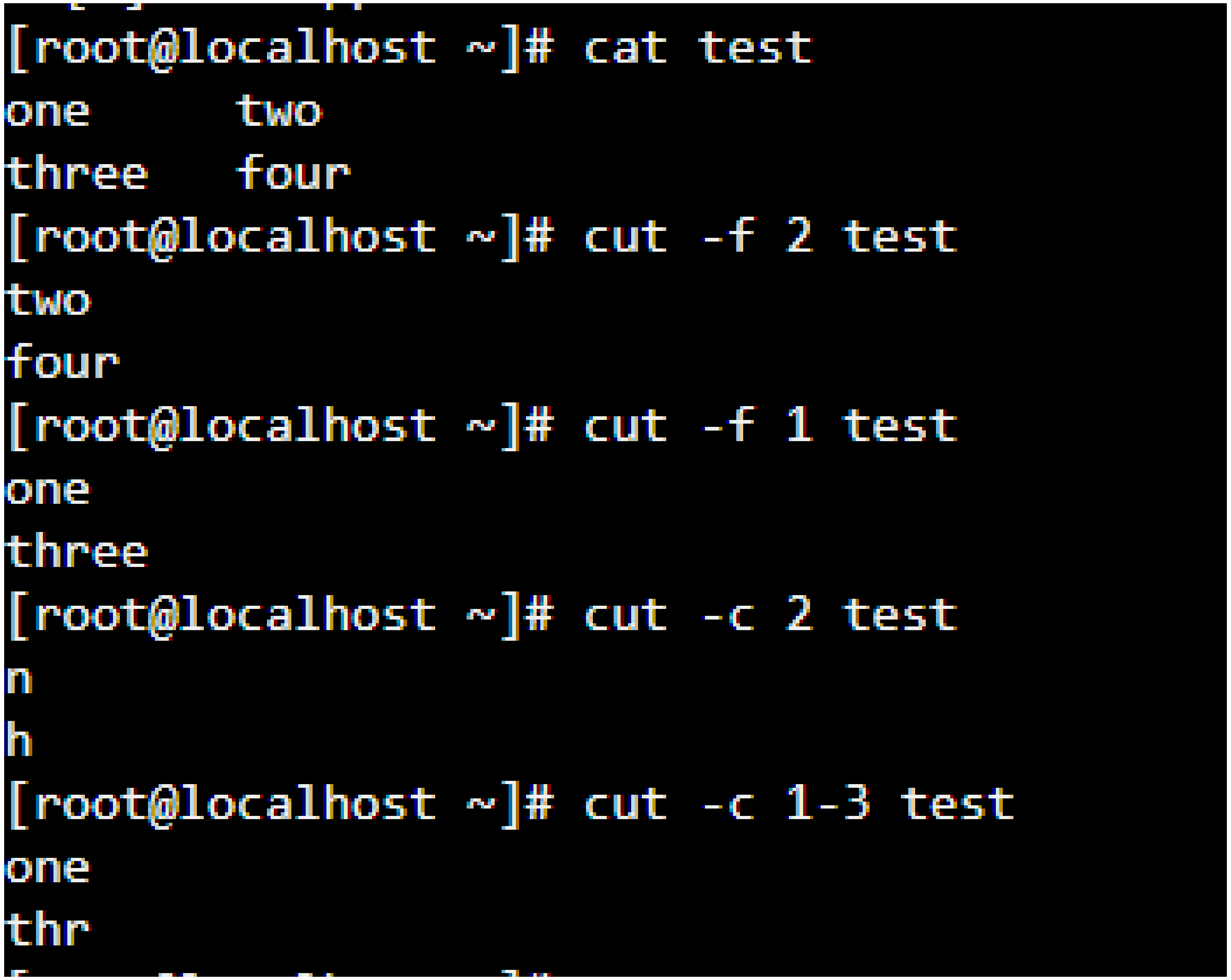
- cut -f 1-2,4-5 data.txt
- ...which will output the following:

```
one	two	four	five
alpha	beta	delta	epsilon
```

- Specifying a range with LIST also applies to cutting characters (-c) or bytes (-b) from a line.
- cut -c 3-12 data.txt
- ...which will output the following:

```
e	two	thre
pha	beta	g
```

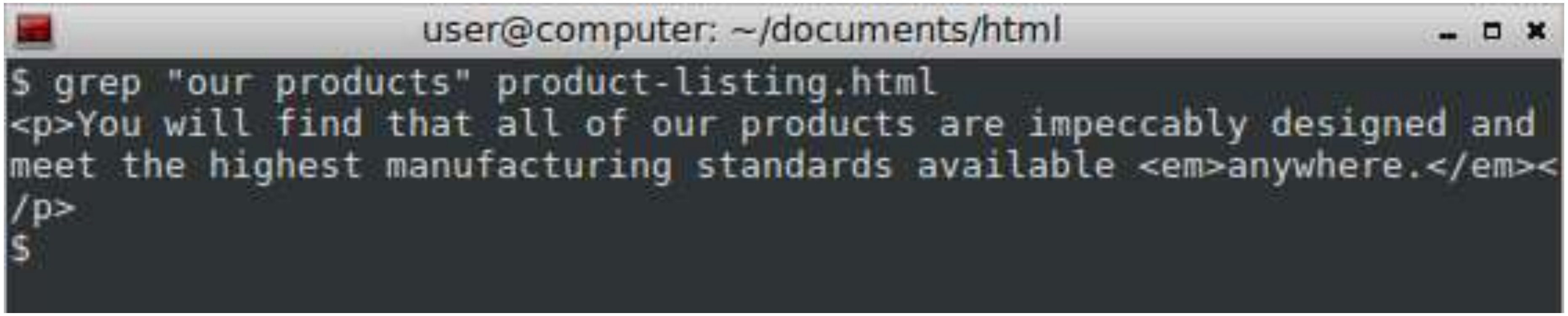
- Remember that the "space" in between each word is actually a single tab character



```
[root@localhost ~]# cat test
one     two
three   four
[root@localhost ~]# cut -f 2 test
two
four
[root@localhost ~]# cut -f 1 test
one
three
[root@localhost ~]# cut -c 2 test
n
h
[root@localhost ~]# cut -c 1-3 test
one
thr
```

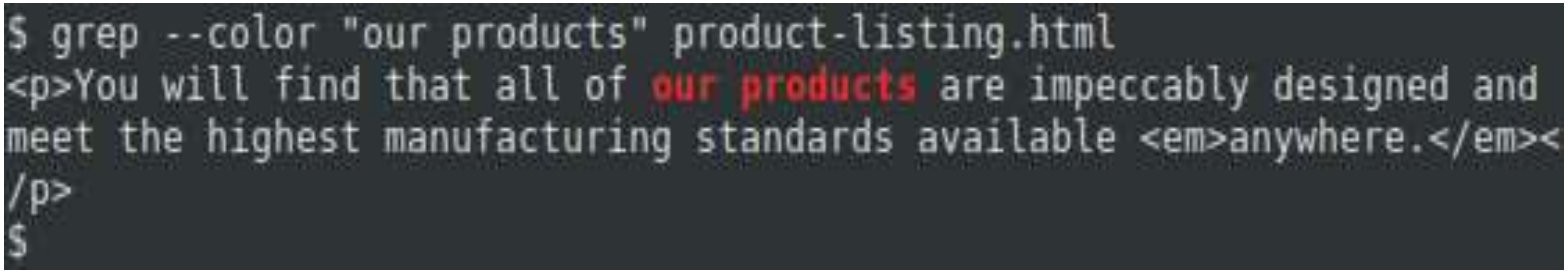
# grep

- grep, which stands for "global regular expression print," processes text line by line and prints any lines which match a specified pattern.
- Grep is a powerful tool for matching a regular expression against text in a file, multiple files, or a stream of input. It searches for the PATTERN of text that you specify on the command line, and outputs the results for you.



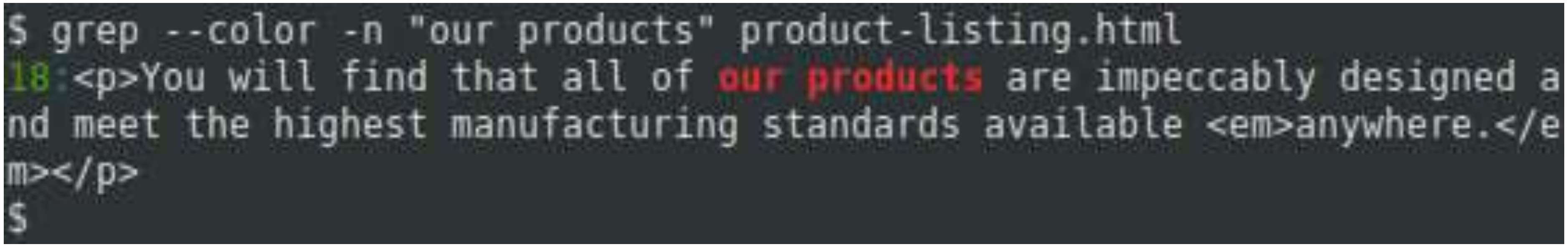
```
$ grep "our products" product-listing.html
<p>You will find that all of our products are impeccably designed and
meet the highest manufacturing standards available <em>anywhere.</em><
/p>
$
```

Successful match highlight for us



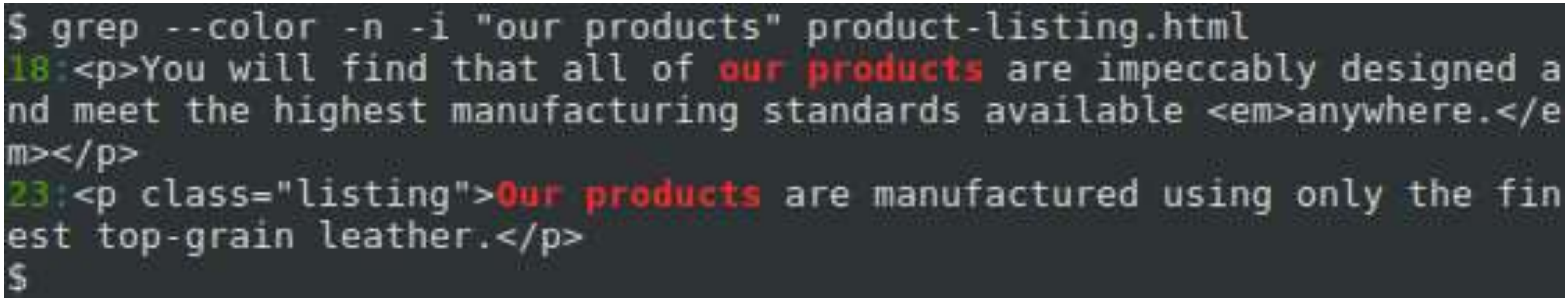
```
$ grep --color "our products" product-listing.html
<p>You will find that all of our products are impeccably designed and
meet the highest manufacturing standards available <em>anywhere.</em><
/p>
$
```

If we specify the **-n** option, **grep** will prefix each matching line with the line number:



```
$ grep --color -n "our products" product-listing.html
18:<p>You will find that all of our products are impeccably designed a
nd meet the highest manufacturing standards available <em>anywhere.</e
m></p>
$
```

We can specify the **-i** option to perform a *case-insensitive* match:



```
$ grep --color -n -i "our products" product-listing.html
18:<p>You will find that all of our products are impeccably designed a
nd meet the highest manufacturing standards available <em>anywhere.</e
m></p>
23:<p class="listing">Our products are manufactured using only the fin
est top-grain leather.</p>
$
```

- We can use grep to search pattern in several files:
- grep picture newfile storyfile
- If it found in both files, the lines containing it would be displayed along with the name of the file where it occurred.
- grep [Rr]ebecca myfile
- Search for Rebecca as well as rebecca
- Grep b??k myfile
- Will display all four letter words whosefirst letter is a ‘b’ and last letter, a ‘k’.
- Grep –v a* myfile
- All those lines that do not contain words starting with ‘a’ are displayed.

## dd

- The dd command copies a file, converting the format of the data in the process, according to the options specified.
- dd if=report of =document conv=ebcdic, ucase
- Converts the input file report from ascii to ebcdic, and while doing this conversion map alphabetic characters in uppercase and copy them to the output file called document.

| **Option** | **meaning** |
|---|---|
| Count=n | copy only n input records. |
| Conv=ascii | convert ebcdic to ascii |
| Conv=lcase | convert to lower case |
| Conv=noerror | continue processing if no error |
| Conv=…,… | several comma separated conversions |
| Skip=n | skip n input records before starting copy |

## View files

- Cat
- Head
- Tail
- More
- Less
- Text editor(vi, joe)

## head

- head makes it easy to output the first part of files.
- head, by default, prints the first 10 lines of each FILE to standard output.
- -c, --bytes=[-]numprint the first num bytes of each file;
- -n, --lines=[-]num print the first num lines instead of the first 10;
- head myfile.txt
- Display the first ten lines of myfile.txt.
- head -15 myfile.txt
- Display the first fifteen lines of myfile.txt.
- head myfile.txt myfile2.txt
- Display the first ten lines of both myfile.txt and myfile2.txt, with a header before each that indicates the filename.

- head -n 5 myfile.txt myfile2.txt
- Displays only the first 5 lines of both files.
- head -c 20 myfile.txt
- Will output only the first twenty bytes (characters) of myfile.txt.

## pg

- pg is a pager: it allows you to view text files one page at a time.
- pg displays a text file on a CRT one screenful at once. After each page, a prompt is displayed. The user may then either press the newline key to view the next page
- pg myfile.txt
- Displays the first screenful of the contents of text file myfile.txt

## Disk related commands

1) df - Report how much free disk space is available for each mount you have.

**Syntax :** df [OPTION]... [FILE]...

| | |
|---|---|
| -a, --all | include dummy file systems |
| -B, --block-size=SIZE | use SIZE-byte blocks |
| -h, --human-readable | print sizes in human readable format (e.g., 1K 234M 2G) |
| -H, --si | likewise, but use powers of 1000 not 1024 |

In the above example when performing just the df command with no additional switches or specification of the file or directory you would get a listing of all file systems and their used and available space.

**df –h -** The above command is one of the most commonly used commands as it displays the sizes in an easy to read format as shown in the below example.

```
Filesystem   Size   Used   Avail   Use%      Mounted on

/dev/hda2    28G    7.6G   19G     29%       /
/dev/hda1    464M   37M    403M    9%        /boot
/dev/hda3    8.3G   429M   7.5G    6%        /var
```

# du

- du estimates and displays the disk space used by specified files and directories.
- $du – reporting the number of blocks used by the current directory (denoted by ‘.’) and those used by sub directories within the current directory.
- $du /dev – du descends down this directory locating any sub directories lying in it and reports the blocks used by the directory and the sub directories.

- To find out the disk usage summary of a /home/tecmint directory
- [root@tecmint]# du  /home/tecmint
- 40       /home/tecmint/downloads
- 4        /home/tecmint/.mozilla/plugins
- 4        /home/tecmint/.mozilla/extensions
- 12       /home/tecmint/.mozilla
- 12       /home/tecmint/.ssh
- 689112  /home/tecmint/Ubuntu-12.10
- 689360  /home/tecmint

- Using “-h” option with “du” command provides results in “Human Readable Format
- [root@tecmint]# du -h /home/tecmint
- 40K     /home/tecmint/downloads
- 4.0K    /home/tecmint/.mozilla/plugins
- 4.0K    /home/tecmint/.mozilla/extensions
- 12K     /home/tecmint/.mozilla
- 12K     /home/tecmint/.ssh
- 673M    /home/tecmint/Ubuntu-12.10
- 674M    /home/tecmint

- to get the summary of a grand total disk usage size of an directory use the option “-s”
- [root@tecmint]# du -s /home/tecmint
- 674M    /home/tecmint

- Using “-a” flag with “du” command displays the disk usage of all the files and directories.
- [root@tecmint]# du -a /home/tecmint
- 4       /home/tecmint/.bash_logout
- 12      /home/tecmint/downloads/uploadprogress-1.0.3.1.tgz
- 24      /home/tecmint/downloads/Phpfiles-org.tar.bz2
- 40      /home/tecmint/downloads
- 12      /home/tecmint/uploadprogress-1.0.3.1.tgz
- 4       /home/tecmint/.mozilla/plugins
- 4       /home/tecmint/.mozilla/extensions
- 12      /home/tecmint/.mozilla
- 4       /home/tecmint/.bashrc
- 689108  /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-i386.iso
- 689112  /home/tecmint/Ubuntu-12.10
- 689360  /home/tecmint

- Find out the disk usage of a directory tree with its subtress in Kilobyte blocks. Use the “-k” (displays size in 1024 bytes units).
- [root@tecmint]# du -k /home/tecmint
- 40      /home/tecmint/downloads
- 4       /home/tecmint/.mozilla/plugins
- 4       /home/tecmint/.mozilla/extensions
- 12      /home/tecmint/.mozilla
- 12      /home/tecmint/.ssh
- 689112  /home/tecmint/Ubuntu-12.10
- 689360  /home/tecmint

- To get the summary of disk usage of directory tree along with its subtrees in Megabytes (MB) only. Use the option “-mh” as follows. The “-m” flag counts the blocks in MB units and “-h” stands for human readable format.

```
• [root@tecmint]# du -mh /home/tecmint
• 40K     /home/tecmint/downloads
• 4.0K    /home/tecmint/.mozilla/plugins
• 4.0K    /home/tecmint/.mozilla/extensions
• 12K     /home/tecmint/.mozilla
• 12K     /home/tecmint/.ssh
• 673M    /home/tecmint/Ubuntu-12.10
• 674M    /home/tecmint
```

- The below command calculates and displays the disk usage of all files and directories, but excludes the files that matches given pattern. The below command excludes the “.txt” files while calculating the total size of diretory. So, this way you can exclude any file formats by using flag “-–exclude“.
- [root@tecmint]# du -ah --exclude="*.txt" /home/tecmint
- 4.0K    /home/tecmint/.bash_logout
- 12K     /home/tecmint/downloads/uploadprogress-1.0.3.1.tgz
- 24K     /home/tecmint/downloads/Phpfiles-org.tar.bz2
- 40K     /home/tecmint/downloads
- 12K     /home/tecmint/uploadprogress-1.0.3.1.tgz
- 4.0K    /home/tecmint/.bash_history
- 4.0K    /home/tecmint/.bash_profile
- 4.0K    /home/tecmint/.mozilla/plugins
- 4.0K    /home/tecmint/.mozilla/extensions
- 12K     /home/tecmint/.mozilla
- 4.0K    /home/tecmint/.bashrc
- 24K     /home/tecmint/Phpfiles-org.tar.bz2
- 4.0K    /home/tecmint/geoipupdate.sh
- 4.0K    /home/tecmint/.zshrc
- 120K    /home/tecmint/goaccess-0.4.2.tar.gz.1
- 673M    /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-i386.iso
- 673M    /home/tecmint/Ubuntu-12.10
- 674M    /home/tecmint

- Display the disk usage based on modification of time, use the flag “–time” as shown below.
- [root@tecmint]# du -ha --time /home/tecmint
- 4.0K    2012-10-12 22:32        /home/tecmint/.bash_logout
- 12K     2013-01-19 18:48        /home/tecmint/downloads/uploadprogress-1.0.3.1.tgz
- 24K     2013-01-19 18:48        /home/tecmint/downloads/Phpfiles-org.tar.bz2
- 40K     2013-01-19 18:48        /home/tecmint/downloads
- 12K     2013-01-19 18:32        /home/tecmint/uploadprogress-1.0.3.1.tgz
- 4.0K    2012-10-13 00:11        /home/tecmint/.bash_history
- 4.0K    2012-10-12 22:32        /home/tecmint/.bash_profile
- 0       2013-01-19 18:32        /home/tecmint/xyz.txt
- 0       2013-01-19 18:32        /home/tecmint/abc.txt
- 4.0K    2012-10-12 22:32        /home/tecmint/.mozilla/plugins
- 4.0K    2012-10-12 22:32        /home/tecmint/.mozilla/extensions
- 12K     2012-10-12 22:32        /home/tecmint/.mozilla
- 4.0K    2012-10-12 22:32        /home/tecmint/.bashrc
- 24K     2013-01-19 18:32        /home/tecmint/Phpfiles-org.tar.bz2
- 4.0K    2013-01-19 18:32        /home/tecmint/geoipupdate.sh
- 4.0K    2012-10-12 22:32        /home/tecmint/.zshrc
- 120K    2013-01-19 18:32        /home/tecmint/goaccess-0.4.2.tar.gz.1
- 673M    2013-01-19 18:51        /home/tecmint/Ubuntu-12.10/ubuntu-12.10-server-i386.iso
- 673M    2013-01-19 18:51        /home/tecmint/Ubuntu-12.10
- 674M    2013-01-19 18:52        /home/tecmint

## Diff command

- **diff** analyzes two files and prints the lines that are different. Essentially, it outputs a set of instructions for *how to change one file in order to make it identical to the second file.*

- If **file1.txt** contains the following four lines of text:
  - I need to buy apples.
  - I need to run the laundry.
  - I need to wash the dog.
  - I need to get the car detailed....
- and **file2.txt** contains these four lines:
  - I need to buy apples.
  - I need to do the laundry.
  - I need to wash the car.
  - I need to get the dog detailed.

- diff file1.txt file2.txt
- 2,4c2,4
- < I need to run the laundry.
- < I need to wash the dog.
- < I need to get the car detailed.
- ---
- > I need to do the laundry.
- > I need to wash the car.
- > I need to get the dog detailed.
- it's telling you how to change the first file to make it match the second file.

- line numbers corresponding to the first file,
- a letter (**a** for *add*, **c** for *change*, or **d** for *delete*), and line numbers corresponding to the second file.
- In our output above, "**2,4c2,4**" means:
  "Lines **2** through **4** in the first file need to be **c**hanged in order to match lines **2** through **4** in the second file."
  It then tells us what those lines are in each file:
- Lines preceded by a **<** are lines from the first file;
- lines preceded by **>** are lines from the second file.
- The three dashes ("**---**") merely separate the lines of file 1 and file 2.

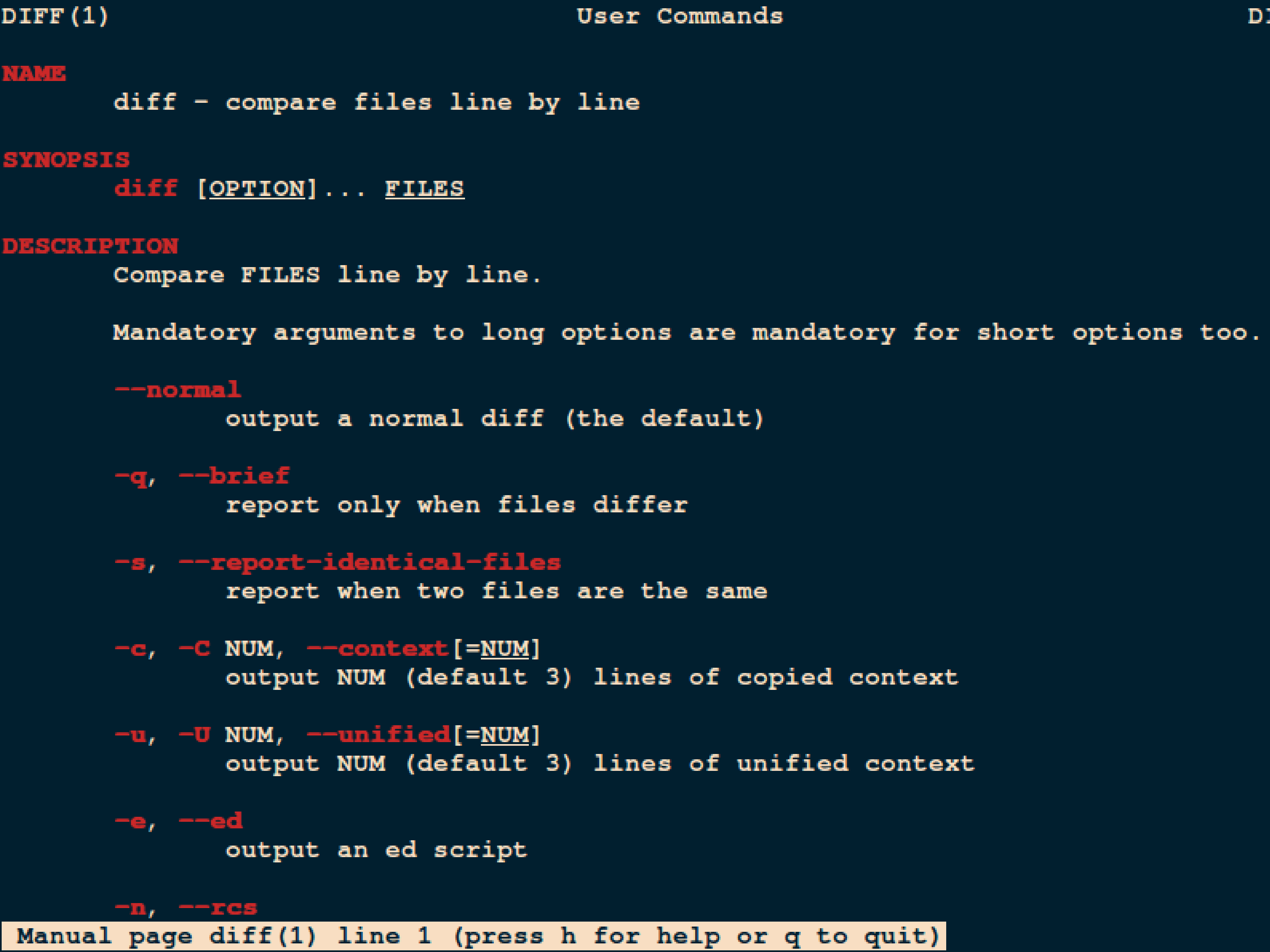
- **file1.txt**:
  - I need to go to the store.
  - I need to buy some apples.
  - When I get home, I'll wash the dog.
- **file2.txt**:
  - I need to go to the store.
  - I need to buy some apples.
  - Oh yeah, I also need to buy grated cheese.
  - When I get home, I'll wash the dog.

- diff file1.txt file2.txt
- 2a3
- > Oh yeah, I also need to buy grated cheese.
- Here, the output is telling us "After line **2** in the first file, a line needs to be **a**dded: line**3** from the second file." It then shows us what that line is.

- **file1**:
  - I need to go to the store.
  - I need to buy some apples.
  - When I get home, I'll wash the dog.
  - I promise.
- **file2**:
  - I need to go to the store.
  - I need to buy some apples.
  - When I get home, I'll wash the dog.

- 4d3
- < I promise.
- Here, the output is telling us "You need to **d**elete line **4** in the first file so that both files sync up at line **3**." It then shows us the contents of the line that needs to be deleted.

| | |
|---|---|
| **-b** | Ignore any changes which only change the amount of whitespace (such as spaces or tabs). |
| **-w** | Ignore whitespace entirely. |
| **-B** | Ignore blank lines when calculating differences. |
| **-y** | Display output in two columns. |



```
DIFF(1)                        User Commands                        DI

NAME
       diff - compare files line by line

SYNOPSIS
       diff [OPTION]... FILES

DESCRIPTION
       Compare FILES line by line.

       Mandatory arguments to long options are mandatory for short options too.

       --normal
              output a normal diff (the default)

       -q, --brief
              report only when files differ

       -s, --report-identical-files
              report when two files are the same

       -c, -C NUM, --context[=NUM]
              output NUM (default 3) lines of copied context

       -u, -U NUM, --unified[=NUM]
              output NUM (default 3) lines of unified context

       -e, --ed
              output an ed script

       -n, --rcs
 Manual page diff(1) line 1 (press h for help or q to quit)
```

## Linux Partitions

- Linux uses more than one partition on the same disk, even when using the standard installation procedure. You should partition a disk (or disks) according to your needs.
  - ➢One of the goals of having different partitions is to achieve **higher data security** in case of disaster. By dividing the hard disk in partitions, data can be grouped and separated. When an accident occurs, only the data in the partition that got the hit will be damaged, while the data on the other partitions will most likely survive.

A simple example: a user creates a script, a program or a web application that starts filling up the disk. If the disk contains only one big partition, the entire system will stop functioning if the disk is full. If the user stores the data on a separate partition, then only that (data) partition will be affected, while the system partitions and possible other data partitions keep functioning.

- it may **reduce the time required to perform file system checks**, because these checks can be done in parallel.
- The division of hard disks into partitions is determined by the system administrator.

Linux allows to partition the disk in 2 ways. They are

1. Automatic partitioning
2. Manual partitioning

Manual partitioning – manual partitioning of a disk in linux is done by the utility *Disk Druid.*

Automatic partitioning – it is done by the system and no manual interference is required. It is preferred as it gives control over the data on the disk. this type of partition has 3 options

- Remove all Linux partitions on this system: This will remove only the linux partitions that are created from a previous linux installation. In this case all other partitions will be safe.
- Remove all partitions on this system : this will remove all partitions on your hard drives and but be careful as there can be data loss.
- Keep all partitions and use existing free space : this retains your current data and partitions and we presume that there is enough space for the next partition.

- **Partition layout and types**

A partition can be classified as

- Primary partition - a partition from where an operating system can boot.
- Extended partition – used to store data or program files. It can be divided into a number of logical drives.

Partitions needed while installation are

- A Swap Partition – The partition of the hard disk that can be used as artificial or virtual memory. This partition is used when the physical memory is not sufficient.
- A /boot partition – The /boot partition contains the operating system kernel, along with few other files used during the boot process

The standard root partition (indicated with a single forward slash, */*) is about 100-500 MB, and contains the system configuration files, most basic commands and server programs, system libraries, some temporary space and the home directory of the administrative user. A standard installation requires about 250 MB for the root partition.

Swap space (indicated with *swap*) is only accessible for the system itself, and is hidden from view during normal operation. Swap is the system that ensures that you can keep on working, whatever happens. On Linux, you will virtually never see irritating messages like *Out of memory, please close some applications first and try again*, because of this extra memory.

- The kernel is on a separate partition and the rest of the hard disk(s) is generally divided in data partitions. it usually happens following a set pattern:
- a partition for user programs (*/usr*)
- a partition containing the users' personal data (*/home*)
- a partition to store temporary data like print- and mail-queues (*/var*)
- a partition for third party and extra software (*/opt*)

Once the partitions are made, you can only add more. Changing sizes or properties of existing partitions is possible but not advisable.

- **Mount points**

All partitions are attached to the system via a mount point. The mount point defines the place of a particular data set in the file system. Usually, all partitions are connected through the *root* partition. On this partition, which is indicated with the slash (/), directories are created. These empty directories will be the starting point of the partitions that are attached to them. An example: given a partition that holds the following directories:

videos/ cd-images/ pictures/

We want to attach this partition in the filesystem in a directory called /opt/media. In order to do this, the system administrator has to make sure that the directory/opt/media exists on the system. Preferably, it should be an empty directory. Then, using the **mount** command, the administrator can attach the partition to the system. When you look at the content of the formerly empty directory /opt/media, it will contain the files and directories that are on the mounted medium (hard disk or partition of a hard disk, CD, DVD, flash card, USB or other storage device)

## System startup and shutdown process

- Two common bootloaders in Linux are Lilo and grub. Bio-Linux currently uses grub which is  easier to use and more flexible.
- When an x86 computer is booted, the processor looks at the end of the system memory for the BIOS (Basic Input/Output System) and runs it. The BIOS program is written into permanent read-only memory and is always available for use. The BIOS provides the lowest level interface to peripheral devices and controls the first step of the boot process.

- The BIOS tests the system, looks for and checks peripherals, and then looks for a drive to use to boot the system. Usually it checks the floppy drive (or CD-ROM drive on many newer systems) for bootable media, if present, and then it looks to the hard drive. The order of the drives used for booting is usually controlled by a particular BIOS setting on the system. Once Linux is installed on the hard drive of a system, the BIOS looks for a Master Boot Record (MBR) and loads its contents into memory, then passes control to it.

- This MBR contains instructions on how to load the GRUB (or LILO) boot-loader, using a pre-selected operating system. The MBR then loads the boot-loader, which takes over the process (if the boot-loader is installed in the MBR). In the default Red Hat Linux configuration, GRUB uses the settings in the MBR to display boot options in a menu. Once GRUB has received the correct instructions for the operating system to start, either from its command line or configuration file, it finds the necessary boot file and hands off control of the machine to that operating system.

- **Init**

When **init** starts, it becomes the parent or grandparent of all of the processes that start up automatically on your Linux system. The first thing **init** does, is reading its initialization file, /etc/inittab and takes care of everything that your system needs to have done at system initialization like setting the clock, initializing serial ports and so forth.

- After having determined the default run level for your system, **init** starts all of the background processes necessary for the system to run by looking in the appropriaterc directory for that run level. **init** runs each of the kill scripts (their file names start with a K) with a stop parameter. It then runs all of the start scripts (their file names start with an S) in the appropriate run level directory so that all services and applications are started correctly. In fact, you can execute these same scripts manually after the system is finished booting with a command like **/etc/init.d/httpd *stop*** or **service *httpd stop*** logged in as *root*, in this case stopping the web server.

- **Shutdown**

  Linux was not made to be shut down, but if you really must, use the **shutdown** command. After completing the shutdown procedure, the -h option will halt the system,while -r will reboot it.The **reboot** and **halt** commands are now able to invoke **shutdown** if run when the system is in run levels 1-5, and thus ensure proper shutdown of the system, but it is a bad habit to get into, as not all UNIX/Linux versions have this feature. If your computer does not power itself down, you should not turn off the computer until you see a message indicating that the system is halted or finished shutting down, in order to give the system the time to unmount all partitions. Being impatient may cause data loss.

- The kernel is on a separate partition as well in many distributions, because it is the most important file of your system. If this is the case, you will find that you also have a */boot* partition, holding your kernel(s) and accompanying data files.
- The rest of the hard disk(s) is generally divided in data partitions, although it may be that all of the non-system critical data resides on one partition, for example when you perform a standard workstation installation. When non-critical data is separated on different partitions, it usually happens following a set pattern:

- The division of hard disks into partitions is determined by the system administrator. On larger systems, he or she may even spread one partition over several hard disks, using the appropriate software. During the installation process you can define your own partition layout using either your distribution specific tool, which is usually a straight forward graphical interface, or **fdisk**, a text-based tool for creating partitions and setting their properties.
- On a server, system data tends to be separate from user data. Programs that offer services are kept in a different place than the data handled by this service. Different partitions will be created on such systems:

- a partition with all data necessary to boot the machine
- a partition with configuration data and server programs
- one or more partitions containing the server data such as database tables, user mails, an ftp archive etc.
- a partition with user programs and applications
- one or more partitions for the user specific files (home directories)
- one or more swap partitions (virtual memory)
- Servers usually have more memory and thus more swap space. Certain server processes, such as databases, may require more swap space than usual

- **Partition layout and types**

  There are two kinds of major partitions on a Linux system:

  - ➢ *data partition*: normal Linux system data, including the *root partition* containing all the data to start up and run the system; and
  - ➢ *swap partition*: expansion of the computer's physical memory, extra memory on hard disk.

  Most Linux systems use **fdisk** at installation time to set the partition type, this usually happens automatically. The standard Linux partitions have number 82 for swap and 83 for data, which can be journaled (ext3) or normal (ext2, on older systems).

- Then **init** continues to read the /etc/inittab file, which describes how the system should be set up in each run level and sets the default *run level*. A run level is a configuration of processes. In run level 1 or run level S (or s), only the system administrator can connect to the system. It is used to perform maintenance tasks without risks of damaging the system or user data. Naturally, in this configuration we don't need to offer user services, so they will all be disabled. Another run level is the reboot run level, or run level 6, which shuts down all running services according to the appropriate procedures and then restarts the system. Use the **who** to check what your current run level is: Eg : who -r

- **Init**

When **init** starts, it becomes the parent or grandparent of all of the processes that start up automatically on your Linux system. The first thing **init** does, is reading its initialization file, /etc/inittab. This instructs **init** to read an initial configuration script for the environment, which sets the path, starts swapping, checks the file systems, and so on. Basically, this step takes care of everything that your system needs to have done at system initialization: setting the clock, initializing serial ports and so forth.

## Administration & Administrator

- The key person in an organization is the system administrator.
- System administrators are engaged in planning activities as per the organizational technical environment.
- They are also in charge of supporting and maintaining servers and desktops and monitoring the network.
- They should have good knowledge of programming also as sometimes scripting or the programming of systems is also required.
- The prime role is user administrator.
- To maintain smooth running of the system.
- To check that the peripherals are working.
- To manage hardware and arrange timely repair of hardware in case of any failure.

- To monitor system performance to the utmost.
- To create file systems as per the organizational requirements.
- To install software as per the organizational requirements.
- To formulate and administer the backup and recover policy to counter any crisis or data loss.
- To manage network communication for better coordination among users.
- To keep updating systems as soon as a new version of OS or application software is released.
- To implemented security policies for use of the computer system and network.
- To formulate security policies for users. A system administrator should a strong grasp of computer security (Eg., firewalls and intrusion detection systems).

## Qualities of a good administrator

- Should be able to update themselves about the latest developments in the field, such that new technologies that will save time and money are implemented.
- Should take conscious efforts for making security of the network and data as a primary objective.
- Should aware of the various tools to identify and monitor potential problems, which even include purchasing of expensive equipment with remote monitoring ability for efficient network operations.
- Should be capable of dealing with requests from users and explaining problems to them in non-technical manner.

- Should be capable enough to adopt a sensible network policy to control network use, keeping in mind that it should not impinge users in operation.
- Should be competent enough with multi-environment.
- Should have adequate understanding of new technologies, regarding how to implement them into the existing network and to design a capable solution for the betterment of the organization.

## Administrative Commands

- /bin – Contains commands for modifying our disk partitions (such as fdisk), changing boot procedure (grub), and changing system states (init).
- /usr/sbin – contains commands for managing user accounts (such as useradd) and configuring our mouse (mouseconfig) or keyboard (kbdconfig). Many commands that are run as daemon processes are also contained in this directory.
- Many administrative commands are contained in regular directories (such as /bin and /usr/bin).

## Configuration Files

- Configuration files are another mainstay of Linux administration.
- Almost everything we set up for our particular computer – user accounts, network address, or GUI preferences is stored in plain text files.
- Configuration files are stored in these locations:
- $HOME – All users store information in their home directories that directs how their login accounts behave. Most configuration files starts with dot(.), so they don’t appear as a users directory when u use ls command. There are dot files that define how each users shell behaves, the look and feel of the desktop, and what options are used with our text editor.

- /etc – Contains most of the basic Linux system configuration files. Following are the major:
  - adjtime – Holds to data to adjust the hardware clock.
  - aliases – contain distribution lists used by the Linux mail service.
  - Bashrc – Sets system wide defaults for bash shell users.
  - Cdrecord.conf – contains defaults used for recording CDs.
  - Crontab – sets cron environment and times for running automated tasks.
  - Csh.cshrc, exports, fdprm, fstab, ftp*, group, gshadow, host.conf, hosts, hosts.allow, hosts.deny, info-dir etc

- /etc/x11 – contains subdirectories that each contain system wide configuration files used by X and different X window managers available with Linux.
- /etc/alternative
- /etc/Amanda
- /etc/cipe
- /etc/cron*
- /etc/cups
- /etc/default
- /etc/httpd
- /etc/init.d
- /etc/mail
- /etc/pcmcia
- /etc/ppp
- /etc/sysconfig etc

# Log Files

- Linux keeps track of it self using log files.
- General system logging is done by syslogd
- Logging is done according to the information in the/etc/syslog.conf.
- Messages are directed to log files in /var/log directory.
- To View Log files
- Go to /var/logs directory: for that use cd
- # cd /var/logs

- /var/log/message: General message and system related stuff
- /var/log/auth.log: Authenication logs
- /var/log/kern.log: Kernel logs
- /var/log/cron.log: Crond logs (cron job)
- /var/log/maillog: Mail server logs
- /var/log/qmail/ : Qmail log directory (more files inside this directory)
- /var/log/httpd/: Apache access and error logs directory
- /var/log/lighttpd: Lighttpd access and error logs directory
- /var/log/boot.log : System boot log
- /var/log/mysqld.log: MySQL database server log file
- /var/log/secure: Authentication log
- /var/log/utmp or /var/log/wtmp : Login records file
- /var/log/yum.log: Yum log files

# Log Files

- One integral part of any UNIX system are the logging facilities.
- The majority of logging in Linux is provided by two main programs, **syslogd** and **klogd**, the first providing logging services to programs and applications, the second providing logging capability to the Linux kernel.
- **Klogd** actually sends most messages to the syslogd facility but will on occasion pop up messages at the console (i.e. kernel panics).
- **Syslogd** actually handles the task of processing most messages and sending them to the appropriate file or device, this is configured from within **/etc/syslog.conf**.
- By default most logging to files takes place in **/var/log/.**

## Managing User Accounts

## Adding users

- Straight method for creating a new user from the shell is with the useradd command.
- After opening terminal with root permission, we simply invoke the useradd command at the command prompt, passing the details of the new account as parameters.
- The only parameter required to useradd is the login name of the user.

## The passwd File

- Every account on the system has an entry in the file */etc/passwd*. This file contains entries, one line per user, which specifies several attributes for each account, such as the username, real name, and other informations. This file contains 7 semicolon-delimited fields.
- Each entry in this file is of the format:

  Username;password;uid;gid;gecos;homedir;shell

- *Username* - A unique character string, identifying the account. For personal accounts, this is the name the user logs in with. On most systems it is limited to eight alphanumeric characters--for example, larry or kirsten.

- Password - An encrypted representation of the user's password. This field is set using the passwd program to set the account's password; it uses a one-way encryption scheme that is difficult (but not impossible) to break.
- Uid - The user ID, a unique integer the system uses to identify the account. The system uses the uid field internally when dealing with process and file permissions; it's easier and more compact to deal with integers than byte strings. Therefore, both the uid and the username identify a particular account: the uid is more important to the system, while username is more convenient for humans.

- Gid - The group ID, an integer referring to the user's default group, found in the file /etc/group.
- Gecos - Miscellaneous information about the user, such as the user's real name, and optional "location information" such as the user's office address or phone number. Such programs as mail and finger use this information to identify users on the system.
- Homedir - The user's home directory, for his personal use. When the user first logs in, her shell finds its current working directory in the named homedir.
- *Shell* - The name of the program to run when the user logs in; in most cases, this is the full pathname of a shell, such as*/bin/bash* or */bin/tcsh*.

- Many of these fields are optional; the only required fields are *username*, *uid*, *gid*, and *homedir*. Most user accounts have all fields filled in, but "imaginary" or administrative accounts may use only a few.
- Note that as the system administrator, it's not usually necessary to modify the */etc/passwd* file directly. There are several programs available that can help you create and maintain user accounts;

## Shadow Passwords

- Any user on the system may read the password file. To some extent, it is a security risk to let everybody with access to the system view the encrypted passwords in/etc/passwd.
- To overcome this potential security risk, shadow passwords have been invented.
- When shadow passwords are used, the password field in /etc/passwd contains only an x or a *, which can never occur in the encrypted version of a password.
- Instead, a second file called /etc/shadow is used. This file contains entries that look very similar to those in /etc/passwd, but contain the real encrypted password in the password field.
- /etc/shadow is readable only by root, so that normal users do not have access to the encrypted passwords.

## The Group File

- Each file on the system has both a user and a group owner associated with it.
- Every user is assigned to at least one group, which you specify in the gid field of the /etc/passwd file.
- However, a user can be a member of multiple groups.
- The file /etc/group contains a one-line entry for each group on the system, very similar in nature to /etc/passwd. The format of this file is:

```
groupname:password:gid:members
```

- Here, groupname is a character string identifying the group; it is the group name printed when using commands such as ls -l.

- password is an optional password associated with the group, which allows users not in this group to access the group with the newgrp command.
- gid is the group ID used by the system to refer to the group; it is the number used in the gid field of /etc/passwd to specify a user's default group.
- members is a comma-separated list of usernames (with no whitespace in between), identifying those users who are members of this group

## Creating Accounts

- Creating a user account manually requires the following steps:
  - Edit /etc/passwd with ***vipw*** and add a new line for the new account
  - Edit /etc/group with ***vigr***
  - Create the home directory of the user with ***mkdir*** command.
  - Copy the files from /etc/skel to the new home directory
  - Fix ownerships and permissions with chown and chmod command
  - Set the password with ***passwd*** command
- The password setting is done as the last step because the user may log in simply while we are still copying the files. After this is done, the new account is ready log in

# Deleting and Disabling Accounts

- To delete an account, you must remove the user's entry in /etc/passwd, remove any references to the user in /etc/group, and delete the user's home directory, as well as any additional files created or owned by the user.
- For example, if the user has an incoming mailbox in /var/spool/mail, it must be deleted as well.
- The command userdel deletes an account and the account's home directory. For example:

```
userdel -r Norbert
```

- Will remove the account for norbert. The -r option forces the home directory to be removed as well. Other files associated with the user--for example, the incoming mailbox, crontab files, and so forth--must be removed by hand.

# Temporarily disabling a user account

- Temporarily (or not-so-temporarily) disabling a user account, is even simpler. You can either remove the user's entry in */etc/passwd* (leaving the home directory and other files intact), or add an asterisk to the first character of the *password* field of the */etc/passwd* entry, as so:

  aclark:*BjDf5hBysDsii:104:50:Anna Clark:
/home/aclark: /bin/bash

- Chsh –s /usr/local/lib/no-login/security anuj
- chsh changes a user's login shell.
- Su – tester
- This account has been closed due to a security breach.

# Modifying User Accounts

- Modifying attributes of user accounts and groups is usually a simple matter of editing /etc/passwd and /etc/group. Many systems provide commands such as usermod and groupmod to do  this; it's often easier to edit the files by hand.
- There are a few commands for changing various properties of an account. They are:
  - chfn – change the full name field
  - chsh – change the login shell
  - passwd – change the password
- To change a user's password, use the passwd command, which will prompt for a password, encrypt it, and store the encrypted password in the /etc/passwd file.
- If you need to change the user ID of an existing account, you can do this by editing the uid field of /etc/passwd directly. However, you should also chown the files owned by the user to that of the new uid.

## Changing Permissions and Ownerships

- -bash: cd: /root/: Permission denied
- That was one demonstration of Linux's security features. Linux, like UNIX, is a multi-user system and file permissions are one way the system protects against malicious tampering.
- All files and directories are "owned" by the person who created them. You created the file foo.txt in your login directory, so foo.txt belongs to you.
- Reading, writing, and executing are the three main settings in permissions.
- Chmod command

## Creating and Mounting File system

- A file system is a combination of methods and data structures, which is also useful for any operating system to keep track of files on a disk or partition and states the way the files are organized on the disk.
- The file system is a process which starts much before a partition or disk can be used as a file system, prior to this the book keeping data structures need to be written to the disk.
- Central concept of file system.

# Creating File System

- mkfs command is used to create file system.
- mkfs [-c] [-t fstype] device [size]
- mkfs is used to build a Linux filesystem on a device, usually a hard disk partition. The device argument is either the device name (e.g. /dev/hda1, /dev/sdb2), or a regular file that will contain the filesystem. The size argument is the number of blocks to be used for the file system.
- -c forces a check for bad blocks.
- -t fstype specifies the file system types.
- Ex:

  mkfs –t ext2 /dev/fd0 – make a ext2 file system on a floppy.

## Mounting a File System

- The file system built on the floppy disk can be linked to the existing file system on the hard disk using the mount command.
- Once mounted we can create files and directories in the new file system and treat it as a normal directory existing in a file system.
- Like all file system, each mounted file system too has a root directory and all its directories fan out from the root.
- Mounting a file system , we are simply attaching its root directory to a particular point in the existing system.
- This point of attachment is called ‘mount point’ for that file system.

- Linux provides a default mount point called /mnt.
- #/etc/mount  /dev/fd096ds15  /mnt
- /mnt is an empty directory in the (/) root directory, with root (superuser) as its owner.
- All users have the permission to access this directory.
- The **mount** command mounts a storage device or file system, making it accessible and attaching it to an existing directory structure.
- All files accessible in Linux, are arranged in one big tree: the file hierarchy, rooted at /. These files can be spread out over several devices. The mount command attaches a files system, located on some device or other, to the file tree.

- mount -t type device dir
- This tells the kernel to attach the file system found on **device** (which is of type **type**) at the directory **dir**.
- If only **directory** or **device** is given, for example:

  mount  /dir

- then **mount** looks for a corresponding mountpoint entry in the **/etc/fstab** file, and attempts to mount it.

# Unmounting a File System

- Unmount command can be used to unmounts any existing file system.
- On execution unmounts delinks the new file system from the root directory of the existing file system.
- The **umount** command "unmounts" a mounted filesystem, informing the system to complete any pending read or write operations, and safely detaching it.
- umount [-t vfstype] [-O options]
- /etc/unmount  /mntsss

## Checking and Monitoring System Performance

## Checking System Space

- Running out of disk space on our computer is not a good situation.
- Using tools that comes with Linux we can keep track of how much disk space has been used on our computer.

## Displaying system space with df

- We can display the space available in our file system using df command.
- $df
- $df –h
- -t , -x, -a, -l

## Checking disk usage with du

- To find how much space is being consumed by a particular directory, we can use du command.

## Finding Disk consumption with find

- The find command is used to find file consumption of our hard disk using a variety of criteria.
- We need to be root user to run this command.
- Eg:

```
find / -user jake –print  | ls –lds > /tmp/jake
```

- Find command searches the root file system(/) for any files owned by the user named jake and prints the filenames. The output of the find command is then listed with a long listing in size order. Finally that output is sent to the file /tmp/jake.
- #find / -size 100k –print | ls –lds > /tmp/size
- Instead of looking for a users files this command like looks for files that are larger than 100 kilobytes.

## Monitoring CPU power

- CPU utilization statistics are:
- User versus System – the percentage of time spent performing user-level processing versus system level processing can point out whether a systems load is primarily due to running applications or due to operating system overheads.
- Context Switches – A context switch happens when the CPU stops running one process and starts running another. Since each context switch requires the os to take control of the cpu.
- Interrupts – interrupts are situations where the processing being performed by cpu abruptly changed. Interrupts are due to hardware or software. High interrupts rates lead to higher system level cpu consumption.

- Runnable processes – processes may be in different states:
  - Waiting for an I/O operation to complete
  - Waiting for the memory management subsystem to handle a page fault.

  Number of Runnable processes.

  Problems with I/o subsystem and memory

  utilization statistics.

## Monitoring CPU usage with top

- The **top** program provides a dynamic real-time view of a running system. It can display system summary information, as well as a list of processes or threads currently being managed by the kernel.
- Top command displays information about the system such as:
  - Uptime : The top command displays the time at which the system was started.
  - Processes: displays the number of users connected to the server. It also provides information on the load on the server.

- Load average: Three load average numbers are the average number of processes, which were ready to run and waiting their turn on the processor in the past 1, 5 and 15 minutes. The load average is a measure of the load on the system. Higher load averages means that the processor is unable to handle the demands of the tasks running on the system.
- CPU States: It displays the time allocated to users and system. It displays tasks whose nice levels have been changed. Also displays the idle time of the system.
- Mem: Displays statistics on memory usage, including total available memory, free memory, used memory, shared memory, and memory used for buffers.
- Swap: Displays statistics on swap space, including total swap space, available swap space, and used swap space.

- PID: Displays unique Process ID.
- PPID: Displays the parent process ID.
- UID: user Id of the tasks owner.
- USER: user name of the tasks owner
- PRI: Priority of the task.
- NI: Nice value of the task.
- STAT: Information about the status of the process. It can be following values
  - R: Runnable
  - S: Sleeping
  - D: Uninterruptible sleep
  - T: Stopped or traced
  - Z: Zombie process

- Commonly used commands with the top command
- Space bar – Updates value on the screen
- h or ? – displays help about various commands
- K – Kills a process
- I – ignores idle and zombie processes
- N – sets the number of the processes that should be displayed on the screen.
- Q – Quits the top command
- S – Modifies the interval in which the values of the top command are updated.

## Monitoring memory utilization

- Page Ins/Page Outs – flow of pages from system memory to attached mass storage devices.
- Active/Inactive pages
- Free, shared, buffered, and catched pages
- Swap Ins/Swap Outs

## File Security

- On a Linux system, every file is owned by a user and a group user.
- For each category of users, read, write and execute permissions can be granted

```
 marise:~> ls -l To_Do
-rw-rw-r--    1 marise  users      5 Jan 15 12:39 To_Do
marise:~> ls -l /bin/ls
-rwxr-xr-x    1 root    root   45948 Aug  9 15:01 /bin/ls*
```

- the first three characters in this series of nine display access rights for the actual user that owns the file. The next three are for the group owner of the file, the last three for other users. The permissions are always in the same order: read, write, execute for the user, the group and the others.

| Code | Meaning |
|---|---|
| 0 or - | The access right that is supposed to be on this place is not granted. |
| 4 or r | read access is granted to the user category defined in this place |
| 2 or w | write permission is granted to the user category defined in this place |
| 1 or x | execute permission is granted to the user category defined in this place |

Access mode codes

**User group codes**

| Code | Meaning |
|---|---|
| u | user permissions |
| g | group permissions |
| o | permissions for others |

- Chmod command

## Su command

- The **su** command, which is short for *substitute user* or *switch user*, is used to become another user during a login session.
- If no username is specified, **su** defaults to becoming the superuser (root).
- To become superuser type su command then Enter key; when prompted for password supply root user password.

- $su
- Password:
- #
- To exit superuser status, type exit or press CTRL+D
- #exit
- >
- $

# Getting system information using uname

- Print information about the current system.
- Print certain system information. If no *OPTION* is specified, **uname** assumes

| | |
|---|---|
| **-a**, **--all** | Prints all information, omitting **-p** and **-i** if the information is unknown. |

|  |  |
|---|---|
| **-s**, **--kernel-name** | Print the kernel name. |
| **-n**, **--nodename** | Print the network node hostname. |
| **-r**, **--kernel-release** | Print the kernel release. |
| **-v**, **--kernel-version** | Print the kernel version. |
| **-m**, **--machine** | Print the machine hardware name. |
| **-p**, **--processor** | Print the processor type |
| **-i**, **--hardware-platform** | Print the hardware platform, or. |
| **-o**, **--operating-system** | Print the operating system. |
| **--help** | Display a help message, and exit. |
| **--version** | Display version information, and exit. |

- chown – change file owner and group
- Usage: chown [OPTION]... OWNER[:[GROUP]] FILE...
- eg. chown remo myfile.txt

## Hostname command

- The **hostname** command shows or sets the system hostname.
- **hostname** is used to display the system's DNS name, and to display or set its hostname or NIS (Network Information Services) domain name.
- When called without any arguments, **hostname** will display the name of the system.
- When called with one argument or with the **--file** option, **hostname** will set the system's host name.
- he host name is usually set once at system startup in the script **/etc/init.d/hostname.sh**
- hostname

  Displays the hostname of the system.

# fdisk command

- **fdisk** is a partition table manipulator for Linux.
- **fdisk** is a menu-driven program for creation and manipulation of partition tables.
- Hard disks can be divided into one or more logical disks called partitions. This division is recorded in the partition table, found in sector 0 of the disk.
- fdisk [-uc] [-b sectorsize] [-C cyls] [-H heads] [-S sects] device
- fdisk -l [-u] [device...]

| | |
|---|---|
| **-b***sectorsize* | Specify the sector size of the disk. Valid values are **512**, **1024**, **2048** or **4096**. Recent kernels know the sector size. |
| **-c**[**=***mode*] | Specify the compatiblity mode, '**dos**' or '**nondos**'. The default is non-DOS mode. |
| **-C** *cyls* | Specify the number of cylinders of the disk. This would be a very strange thing to want to do, but if so desired, this option will get it done. |
| **-H** *heads* | Specify the number of heads of the disk. Not the physical number, but the number used for partition tables. Reasonable values are **255** and **16**. |
| **-S** *sects* | Specify the number of sectors per track of the disk. Not the physical number, but the number used for partition tables. A reasonable value is **63**. |
| **-h** | Print help and then exit. |
| **-l** | List the partition tables for the specified devices and then exit. If no devices are given, those mentioned in **/proc/partitions** (if that exists) are used. |
| **-u**[**=***unit*] | When listing partition tables, show sizes in '**sectors**' or in '**cylinders**'. The default is to show sizes in sectors. |
| **-v** | Print version information, and exit. |

## Red Hat Package Manager(RPM)

- RPM, the Red Hat Package Manager, is a tool that automates the installation of software binaries and remembers what files are needed so that you can be assured the software will run properly.
- An RPM package consists of an archive of files and package information, including name, version, and the description.

# Working with RPM

- rpm [options]
- Depending on the options that we select, we might have to specify the package name, the source RPM name, or a particular package file.
- In the basic level we will perform following activities with RPM:
  - Installing
  - Upgrading
  - Uninstalling
  - Querying
  - Verifying

## Installing Packages

- RPM packages have file names typically as some_package-2.0-1.i386.rpm.
- Some_package is the name of package, 2.0 is the version, 1 is the release, and i386 is the architecture of the package.
- General form:
- Rpm –i[install options] <package_file>
- -i option used to install new packages.
- -v option used to verify files that are being installed or uninstalled.
- -h option used to print hash marks as the package is being installed.

- Example:
- #rpm –ivh taper-6.9b-3.i386.rpm
- Preparing…      #################### [100%]
- I:taper      #################### [100%]
- The taper package version 6.9b, release 3 for Intel platforms, is being installed.

- Before installing the packages, we can test if the package can be installed without any problem using –test option.
- # rpm –ivh taper-6.9b-3.i386.rpm –test
- Above command would display the # prompt with no error message if the package can be successfully installed.
- If package cannot be installed, an error message will be reported.

- If our machine contains insufficient space to install a particular package the rpm command will display an error message stating the amount of space that it expects in the file system on which it installs the package.
- # rpm –ivh taper-6.9b-3.i386.rpm
- Preparing…     ##################### [100 %]
- Installing package taper-6.9b-3 needs 1024Kb on the filesystem.

- # rpm –ivh taper-6.9b-3.i386.rpm
- Preparing…      ##################### [100 %]
- Package taper-6.9b-3 is already installed.
- We can install package even it is already installed use –replacepkgs option
- # rpm –ivh taper-6.9b-3.i386.rpm -replacepkgs

- Some packages depends on other packages, which means that before install the required package, we must install all the packages depends.
- If we try to install
- #rpm –ivh mysql-server-3.23.36-1.i386.rpm
- Error: failed

  mysql=3.23.36 is needed by mysql-server-3.23.36-1

  libmysqlclient.so.10 is needed by mysql-server-3.23.36-1
- If we want to force the installation of the packages even though it depends on other packages, use the –nodeps option.
- #rpm –ivh mysql-server-3.23.36-1.i386.rpm -nodeps

# Upgrading packages

- -U option used to upgrade.
- rpm -U[install-options]<package file name>
- Rpm automatically uninstalls the previous version of the software and installs the new versions.
- #rpm –Uvh telnet-0.7-18.i386.rpm
- We can use –U option to install new package.
- When upgrading packages, we may receive the following message:
- Saving /etc/some_package.conf as /etc/some_package.conf.rpmsave
- Configuration file formats are different.
- If we try to upgrade a package and RPM finds a newer version of the software installed on the system, error
- #rpm –Uvh-0.17-10.i386.rpm
- Package telnet-0.17-18 is already installed

## Uninstalling packages

- The –e(erase) option is used with the rpm command to do uninstalling.
- #rpm –e <package name>
- Don’t need to specify the full package name.
- #rpm –e netscape-communicator

- When we uninstall packages, an error can occur, if some other package depends on it.
- Ex:
- #rpm –e tar
- Error: removing these packages would break dependencies:

  tar is needed by mkinittrd-3.0.10-1

  tar is needed by Amanda-2.4.2p2-1
- To ignore this error and uninstall the package use the –nodeps options

# tr Command

- The **tr** command automatically translates (substitutes, or maps) one set of characters to another.
- The **tr** utility copies the standard input to the standard output with substitution or deletion of selected characters.
- tr [-Ccsu] string1 string2
- In this form, the characters in the string *string1* are translated into the characters in *string2* where the first character in *string1* is translated into the first character in *string2* and so on. If *string1* is longer than *string2*, the last character found in *string2* is duplicated until string1 is exhausted.

- tr [-Ccu] -d string1
- In this form, the characters in *string1* are deleted from the input.

| | |
|---|---|
| **-C** | Complement the set of characters in *string1*, that is "**-C ab**" includes every character except for '**a**' and '**b**'. |
| **-c** | Same as **-C** but complement the set of values in *string1*. |
| **-d** | Delete characters in *string1* from the input. |
| **-s** | Squeeze multiple occurrences of the characters listed in the last operand (either *string1* or *string2*) in the input into a single instance of the character. This occurs after all deletion and translation is completed. |
| **-u** | Guarantee that any output is unbuffered. |

# Uniq Command

- **uniq** reports or filters out repeated lines in a file.
- **uniq** filters out adjacent, matching lines from input file *INPUT*, writing the filtered data to output file *OUTPUT*.
- If *INPUT* is not specified, **uniq** reads from the standard input.
- If *OUTPUT* is not specified, **uniq** writes to the standard output.
- uniq [OPTION]... [INPUT [OUTPUT]]

| | |
|---|---|
| **-c**, **--count** | Prefix lines with a number representing how many times they occurred. |
| **-d**, **--repeated** | Only print duplicated lines. |
| **-D**, **--all-repeated**[**=***delimit-method*] | Print all duplicate lines. *delimit-method* may be one of the following: |
| **-i**, **--ignore-case** | Normally, comparisons are case-sensitive. This option performs case-insensitive comparisons instead. |
| **-s** *N*, **--skip-chars=***N* | Avoid comparing the first *N* characters of each line when determining uniqueness. This is like the **-f** option, but it skips individual characters rather than fields. |
| **-u**, **--unique** | Only print unique lines. |
| **-w**, **--check-chars=***N* | Compare no more than *N* characters in lines. |
| **--help** | Display a help message and exit. |
| **--version** | Output version information and exit. |

# egrep Command

- egrep is an acronym for "Extended Global Regular Expressions Print".
- It is a program which scans a specified file line by line, returning lines that contain a pattern matching a given regular expression.
- egrep <flags> '<regular expression>' <filename>
- -c for counting the number of successful matches and not printing the actual matches
- -i to make the search case insensitive
- -n to print the line number before each match printout
- -v to take the complement of the regular expression (i.e. return the lines which don't match)
- -l to print the filenames of files with lines which match the expression.
- matches are allowed to occur anywhere within the string

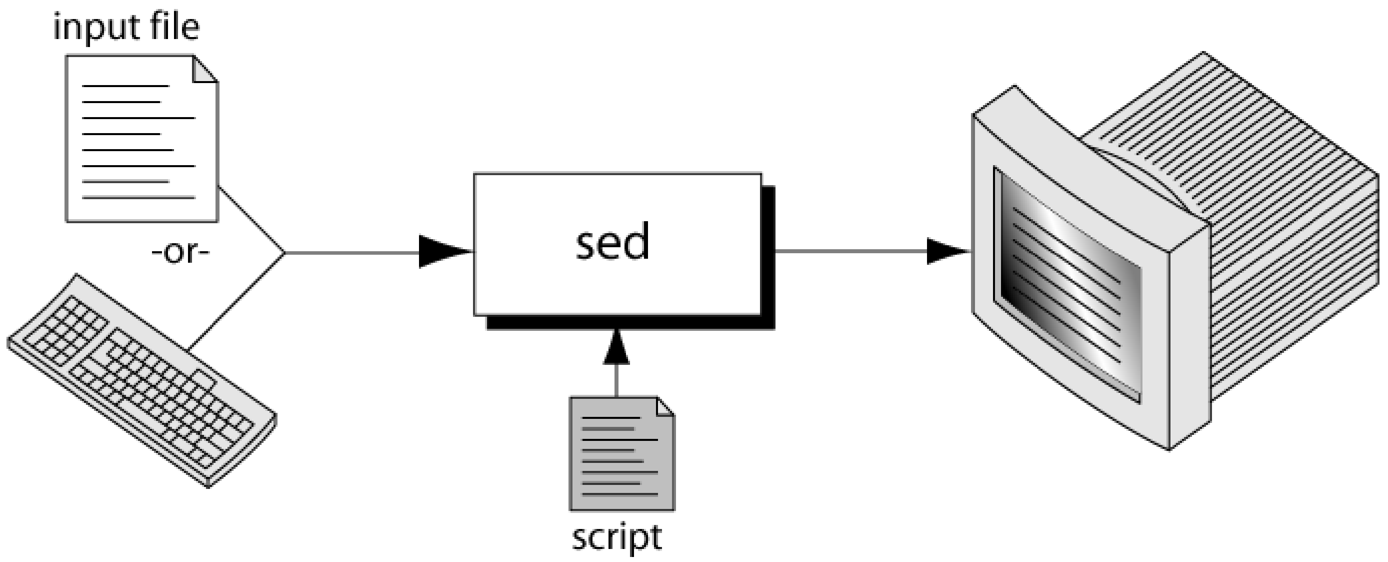
## Sed command

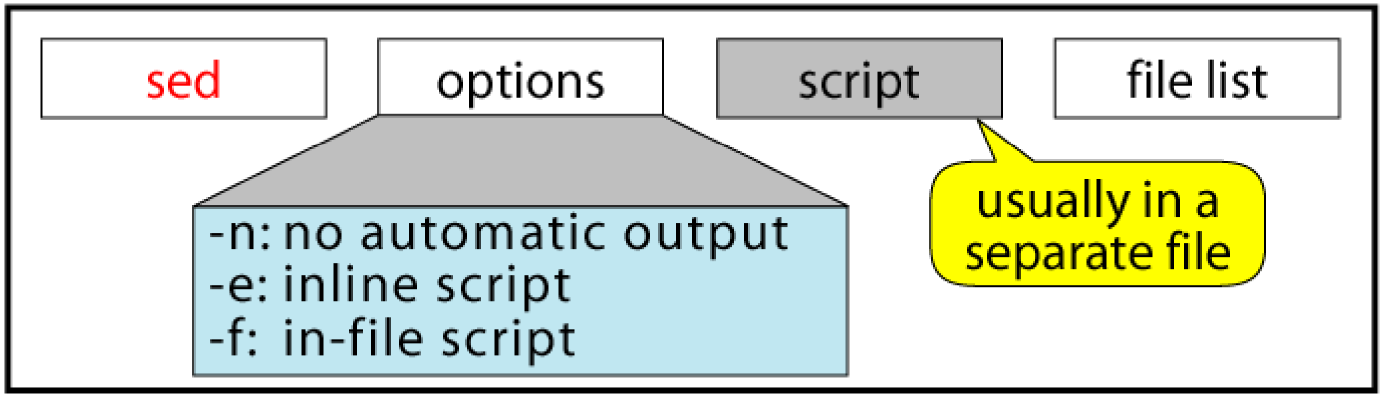
- **sed**, short for "stream editor", allows you to filter and transform text.
- sed OPTIONS...  [INPUTFILE...]
- **sed** is a *stream editor*. A stream editor is used to perform basic text transformations on an input stream (a file, or input from a pipeline).

## SED: <u>S</u>tream-oriented, Non-Interactive, Text <u>Ed</u>itor

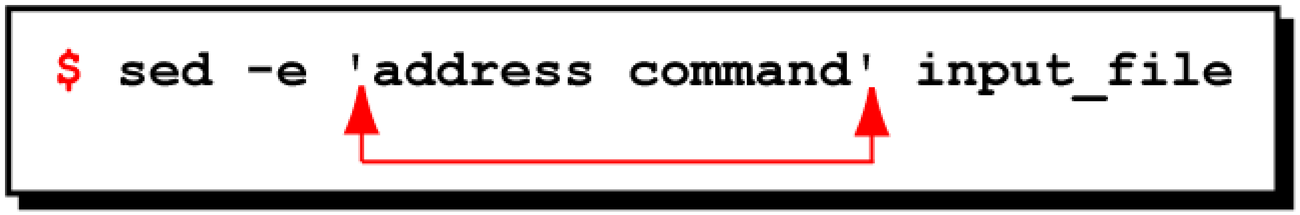
- **Typical Usage**:
  - edit files too large for interactive editing
  - edit any size files where editing sequence is too complicated to type in interactive mode
  - perform "multiple global" editing functions efficiently in one pass through the input
  - edit multiples files automatically
  - good tool for writing conversion programs

# The sed command



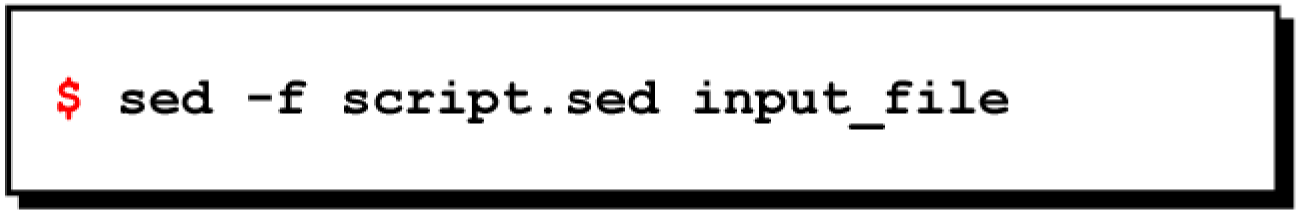


# sed command syntax



```
$ sed -e 'address command' input_file
```

(a) Inline Script



```
$ sed -f script.sed input_file
```

(b) Script File

# SED Usage

- `sed  [-n] -e 'command' [file]*`
- `sed [-n] -f scriptfile [file]*`

**-n** suppresses default output (except for lines specified with the **p** command, or pflag of the **s** (substitute) command.

# SED: Overall Operation

References: Unix In a Nutshell (o'reilly)

- input file is unchanged
- processes one line at the time
- copies standard input to standard output, perhaps performing one or more editing commands on each input line

# How Does sed Work?

- sed reads line of input
  - line of input is copied into a temporary buffer called pattern space
  - editing commands are applied
    - subsequent commands are applied to line in the pattern space, not the original input line
    - once finished, line is sent to output

      (unless –n option was used)
  - line is removed from pattern space
- sed reads next line of input, until end of file

## SED: REGULAR EXPRESSIONS

**c:** ordinary character, matches that character

**^** matches the beginning of the line

**$** matches the end of the line

**'\n'** matches an embedded newline character, nut not the newline at the end of a pattern space.

**.** period matches any single character , but not newline

r* matches any number (zero or more) of the regular expression preceding it.

# SED: Regular Expressions
(cont')

**[…]** matches any character in the …

**[^…]** matches any character not in …

**r1r2** matches the concatenation of r1r2

**\(..\)** is a tagged regular expression

**'\d'** means the same string of characters matched by an expression enclosed in '\(' and '\)' earlier in the same pattern; **d** is a single digit

// null regular expression is equivalent to the last regular expression compiled.

# Processes in Linux

- Linux as we know is a multiuser, multi tasking os.
- Scheduler which decides which process should get CPU attention and when.
- Ps command is used to see which processes are running at any instant.
- The output of ps shows the PIDs for the two processes being run by the user when ps was executed, terminal from which the processes were launched, the time that has elapsed since the processes were launched and the names of the processes.
- $ps

```
    PID         TTY         TIME        COMMAND
    2269        3a          0:05        sh
    2396        3a          0:00        ps
```

- Simple ps display only the process that are running at users terminal.

- To find out which processes are running for the other users who have logged in execute the ps command with –a option. –a stands for all users.
- If we want to see what a particular user vis doing just type ps –u user1.
- If we want to find out the processes that have been launched from a particular terminal use ps –t tty3d.
- If we want additional information then use –f option.
- If we want to display all processes running at that instant use –e option.

## Process Types

- Interactive Process(Foreground process & Background process)
- Automatic process
- Daemons

## Interactive Process

- Interactive processes are initialized and controlled through a terminal session. In other words, there has to be someone connected to the system to start these processes; they are not started automatically as part of the system functions.
- These processes can run in the foreground, occupying the terminal that started the program, and you can't start other applications as long as this process is running in the foreground.
- Alternatively, they can run in the background, so that the terminal in which you started the program can accept new commands while the program is running.

## Background Process

Most of the system process run in the background, while the users execute their processes in the foreground. Using this facility of linux the user can run time-consuming  tasks like sorting a huge file and storing the sorted output in a file in the background. In this case, there is no need to wait until the sorting is over to run the next process. As soon as the sorting process is submitted to run in the background , the terminal session can be used for other tasks.

To run a process in the background, use the ampersand(&) symbol. While executing a command if this symbol is placed at the end of the command then this command will be executed in the background.

```
$  sort   emp.dat  >  emp.out  &
     1765
```

1765 is the PID(process ID )of the process that we have just executed in the background.

**Limitations of background process are :**

- On termination of a background process no success or failure is reported on the screen. Here PID is used pd command to get the status of each process.

- The o/p of a background process should always be redirected to a file. Otherwise we will get a screen mix up with the o/p of background process with whatever we are doing in foreground.
- With too many processes running in the background the overall system performance is likely to degrade.
- If u logout while some of your processes are running in background all these processes would be abandoned halfway through.

## nohup Command

- We want to ensure that the processes that have executed should not die even when we logout, use nohup command.
- Using this command we can submit the time consuming commands in the background, log out and leave the terminal and return other time to see our output is ready.
- $nohup sort employee.dat > output.dat &

  17695
- If we do not redirect the output of background process the command acts intelligently and stores the output in the file ‘nohup.out’.

## Killing a Process

- $kill 6173

  6173 Terminated
- When invoked it sends a termination signal to the process being killed.
- These signals have been given numbers.

## Multiple Process

- The shell offers a feature called *job control* which allows easy handling of multiple processes. This mechanism switches processes between the foreground and the background. Using this system, programs can also be started in the background immediately.
- One approach is to open many terminal windows
- Another is to run programs in the background in one or a few terminals, and save the output of each program t files. Is bash and most other Linux shells, '>' redirects a program's output. '2>' redirects it's error log. '2>&1' merges those two for that process.
- Linux is what is referred to as a preemptive multitasking operating system. Preemptive multitasking systems rely on a scheduler. The function of the scheduler is to control the process that is currently using the CPU.
- A **killall** process never kills itself (but may kill other **killall** processes)s

## Changing process priorities

- Processes in linux can differentiated using their priorities. Process with higher priority will get the cpu faster than other processes in the queue. Priority of the process is determined by the number associated with it. This number is called ‘nice’ value of the process. Higher the nice value of a process, lower the priority. The nice value of a process can range from 0-39, with 20 as the default nice value of a process. Thus a process with nice value 20 will execute more faster than a process with nice value 25.

By default nice value of the below process have value 20. then execute this

Eg : nice cat emp.txt            This will increase the nice value of the above process from 20 to 30. as we havnt specified any increment, by default an increment of 10 will occur.

We can specify the value of increment like

```
$ nice -15 cat emp.txt
```

Now cat process will have a nice value or priority 35(20+15). The increment can range from 0-19.

Only the superuser can put a process on a higher priority by reducing its nice value. The superuser can reduce the nice value as below

```
$ nice - -10 sort emp.dat > out.dat
```

Priority of a process can be changed at the time of firing the process at command prompt. Once the process has been submitted to the process queue its priority cannot be changed.

Nice values of various processes can be seen by

```
$ ps –l
```

**Eg to check the time taken for executing a process before and after increasing the nice value**

```
time  ls -laR > dir.txt
real 0.1s
User 0.0s
Sys 0.1s

Time nice -25  ls -laR > dir.txt
real 0.1s
User 0.2s
Sys 0.1s
```

## Scheduling of processes

- The shell provides two simple utilities to schedule the utilities that we want to run at specific times. They are **cron** and **at** utilities.

## Crontab Utility

- The crontab utility instructs cron to execute the commands on a specific date and time.
- It supplies the crontab file to the cron daemon.
- Syntax:

```
crontab [-u user] file
crontab [-u user] [options]
```

- Example:

```
$crontab cronfile
```

- This command invokes the crontab utility with the crontab file as the parameter.
- Command send the crontab file to the cron daemon.
- Will replicate file in the /var/spool/cron directory.
- Will create one file for each user with the user login name.

## The crontab file

- List the tasks that need to be scheduled in a file.
- The file can bear any name.
- It has a special format and has one line for each task that want to automate.
- Format:

Minute  hour  day-of-month  month-of-year day-of-week  command

| Column | Meaning | Valid Values |
|---|---|---|
| 1 | The minute of the hour | 0-59 |
| 2 | The hour of the day | 0-23 |
| 3 | The day of the month | 1-31 |
| 4 | The month of the year | 1-12 |
| 5 | The day of the week | (Sunday=0, Monday=1,……,Saturday=6) |
| 6 | Command/shell script name | Any utility to be executed at that point of time |

Each column could also have an asterisk(*), which denotes that it can be any of the valid values.

## * * * * *  echo “This is an example of cron command”

- After saving the file we need to execute the following command to assign the task to the cron daemon.
- $crontab cronfile
- We can list the tasks that have been assigned to the cron daemon by executing the following command
- $crontab –l
- The echo command is executed after every minute.
- The output of the command will be send by e-mail to the user.

- The –e option with crontab command allows we to directly modify the crontab file that resides with cron daemon.
- User need not resubmit crontab file.
- The –r option with the crontab command to delete the scheduled task.
- Eg 2: 0,20,40 20-21 * 7 fri-sat /home/ian/mycrontest.sh

- In this example, the command is executed at the 0th, 20th, and 40th minutes (every 20 minutes), for the hours between 8 pm and 9 p.m on Fridays and Saturdays during July(7).

## Cron Utility

- The cron daemon starts automatically when the system starts and executes the tasks that we have scheduled at regular intervals.
- The crontab utility is used to submit the tasks to the cron daemon. The tasks to be automated have to be specified in a file called the *crontab* file.
- The cron daemon checks the each entry of the crontab file. Each time the date, time and day of an entry in the file matches with the system date, cron executes the command.
- For each task that the scheduler executes,  a mail message will be sent to the user by the cron daemon.
- Each user of the linux system has a separate crontab file that he or she can submit to the cron daemon.

## ‘at’ Utility

- Another utility that is used to schedule the tasks is ‘at’ utility. It can schedule the tasks to be run only once but the cron daemon is used to execute the tasks at regular intervals.
- ie,. Crontab utility is used to run the tasks repetitively and at utility is used to schedule a task which has to be execute only once as a particular specified time.
- The number of users who can use this at utility can be limited by using **at.allow** file.
- There is an **at.allow** and **at.deny** file. Only login names specified in the at.allow file can use the at utility and an empty at.deny file indicated that all users can use the at utility which is the default configuration.

- List of commands used to work with at utility are:
- at – accepts commands or shell scripts to be executed later by using the bash shell
- atq – display the list of jobs pending for the current user. If super user is logged on, the pending jobs of all users are displayed
- atrm – removes a job from list of pending jobs
- batch – executes the scheduled tasks only when the system load level is below a specific level.
- Eg:

```
$ at 6:23pm Jan 01     echo “checking at command here"
```

## batch Command

- Instead of specifying that our commands be executed at a precise moment in time batch executes commands in best time for executing our commands.
- When we submit our jobs using this command linux executes our job when it is relatively free and the system load is light.
- Time of execution of command is left for the system to decide.
- $batch

```
sort employee.dat | grep Nagpur > addresses.out
```

## Printing commands

- Linux is a multiuser operating system that allows users to share hardware resources, such as printers. So all users can print files at a centrally located printer in the linux os. The printer commands of linux are :
- lpr
- lpq
- lprm
- lpc
- pr

## lpr command

lpr command is used to print files to a local or a network printer.

**Syntax : lpr  [options]   [filename1 filename2]**

Options in lpr command is used to specify the settings like no of copies, page width etc. multiple documents can be specified for printing using this command

Eg : lpr file1.txt file2.txt – here the two files file1.txt and file2.txt will be send to the default printer for printing.

The various options of lpr command are :

- -p<printername> - to specify the printer name. if not specified the file will be send to default printer
- -#<num> - to set the no of copies to be printed
- -w<num> - to set the page width
- -s – to print large files without copying them.
- -r – to delete the file after printing (-r and –s options cannot be used together)

## lpq command

Multiple users can print files using the same printer. So print jobs are sent to the print queue. the lpq command is used to

- To check the status of print queue
- View the entries in the print queue

**Synatx : lpq [options]**

The different options are :

**lpq [-l] [-P printer] [job ID] [user/s]**

- ➢ -p<printername> - to view the status of a specified printer. If printername is not specified the status of default printer is shown
- ➢ -l – to display a single line description of each file in the print queue
- ➢ Job ID – to display the status of a print job with the specified jobID
- ➢ User <username> - to display the files that the specified user is printing

The o/p of lpq command is as follows

```
lp is ready and printing
Rank     owner     job          Files
   Total Size
Active    steve             3         file1.txt
   4567 bytes
```

The above o/p displays the name of owner, the job ID and the size of the files that are in the print queue for the printer.

## lpc command

The lpc command is used by system administrators to control the operations of the print services. The /etc/printcap file stores information about the printers configured on a computer. For each printer specified in this file, the lpc command can be used to

- Disable or enable a printer
- Disable or enable the print queue of a printer
- Rearrange the jobs in a print queue
- Check the status of printers, their print queues and the printer daemons

**Synatx : lpc [subcomands] [arguments]**

The subcommands are used to perform operations such as disabling or enabling print queues and the arguments are used to specify the settings such as printer name for which we need to disable the print queue.

Some of the subcommands of lpc command are:

- abort – is used to kill the active job and if we use this option the lpr command cannot spool the print jobs in the print queue
- disable – to disable the print queue for the specified printer
- enable - to enable the print queue for the specified printer
- status – to display the status of daemons and printer queues on the local printer.

## pr command

Pr command is used to format a file and specify attributes such as page size, the number of lines in a page, borders, header and footer etc and print the file. Typically pr command is sued to format large text files.

**Syntax : pr [options] [filename/s]**

With pr command multiple filenames can be specified by a space. If we do not specify a filename, it expects input from the standard input file(keyboard or pipe).

Some options of pr command are :

- + <pageno> - to print a file starting from the specified page number.
- - <no of column> - to display data in the specified no of columns
- -w <page width> - used to set the page width to the specified number

- ➢ -o <left margin> - used to set the left margin space for each line
- ➢ -l <page length> - used to set the page length to a specified no
- ➢ -h <header string> - to set the header to the specified string
- ➢ -d – to print double spaces between the lines in a file
- ➢ -s <column separator> -to specify that the columns should be separated by the specified column separator
- ➢ -t - to specify that the printer should not print the header and the footer on each page

Eg : `$ pr  +1  –h  ‘SALES REPORT’  -l 15  -o 5  -w 40  /home/steve/file.txt`

The above command means the file file.txt should be printed with the following settings

- Should be printed from first page
- Page header should be SALES REPORT
- Length of page should be 15 lines
- Left margin should be 5 spaces
- Width of page should be 40 columns

## Filter commands

### Simple Filter Commands

- cut –
- Paste –
- uniq –
- tr –
- wc-
- pr -
- head –
- tail –

## Filter commands using Regular Expressions

grep

Egrep-Search for a pattern using extended regular expressions. **egrep** is essentially the same as running **grep** with the **-E** option.

sed

awk

sort

# Uniq Command

- **uniq** reports or filters out repeated lines in a file.
- uniq [OPTION]... [INPUT [OUTPUT]]
- **uniq** filters out adjacent, matching lines from input file *INPUT*, writing the filtered data to output file *OUTPUT*.

  If *INPUT* is not specified, **uniq** reads from the standard input.

  If *OUTPUT* is not specified, **uniq** writes to the standard output.

  If no options are specified, matching lines are merged to the first occurrence.
- -c, --count     Prefix lines with a number representing how many times they occurred.
- -d, --repeated          Only print duplicated lines.
- -i, --ignore-case        Normally, comparisons are case-sensitive. This option performs case-insensitive comparisons instead

## tr command

- The **tr** command automatically translates (substitutes) sets of characters.
- tr  string1 string2
- In this form, the characters in the string *string1* are translated into the characters in*string2* where the first character in *string1* is translated into the first character in*string2* and so on. If *string1* is longer than *string2*, the last character found in *string2* is duplicated until string1 is exhausted.
- tr -d string1
- In this form, the characters in *string1* are deleted from the input.

|  |  |
|---|---|
| **-C** | Complement the set of characters in *string1*, that is "**-C ab**" includes every character except for '**a**' and '**b**'. |
| **-c** | Same as **-C** but complement the set of values in *string1*. |
| **-d** | Delete characters in *string1* from the input. |
| **-s** | Squeeze multiple occurrences of the characters listed in the last operand (either *string1* or *string2*) in the input into a single instance of the character. This occurs after all deletion and translation is completed. |
| **-u** | Guarantee that any output is unbuffered. |

- tr -cs "[:alpha:]" "\n" < file1

Create a list of the words in **file1**, one per line, where a word is taken to be a maximal string of letters.

- tr "[:lower:]" "[:upper:]" < file1

Translate the contents of **file1** to uppercase.

- tr -cd "[:print:]" < file1

Remove all non-printable characters from **file1**.

## Mathematical Commands

## bc Command

- The calculator called bc, short for base conversion or best calculator.
- Once we type bc at the shell prompt we are in calculator mode.
- $ disappears.
- The input to the calculator is taken line by line.
- $bc
- 10/2*2
- 10
- 2.5*2.5+2
- 8.25
- quit
- Typing quit ends our tryst with bc.

- $bc
- Scale=1
- 2.25+1
- 3.3
- Quit
- Bc also supports sqrt, cosine, sine, tangent etc
- $bc
- Sqrt(196)
- 14
- Quit

```
$factor
15
        3
        5
28
        2
        2
        7
Q
$
```

- $bc
- For(i=1;i<=5;i=i+1)i
- 1
- 2
- 3
- 4
- 5
- quit

## expr command

- Evaluate expression we are using expr command
- $expr 100+50
- 150
- $expr 3*2
- Working with shell a * always expands to all files in the current directory. The \ just takes away this special meaning of *.
- expr cant handle floating point arithmetic

## factor command

- Factor used to display the factors.
- When factor is invoked without an argument it waits for a number to be typed in.
- If we type a positive number less than 2^46 it will factories the number and prints its prime factors

## units

- Converts quantities expressed in various standard scale to their equivalents in other scales
- Works interactively
- $units
- You have: inch
- You want: cm
- *2.540000e+00
- /3.937008e-01

## Shells in Linux

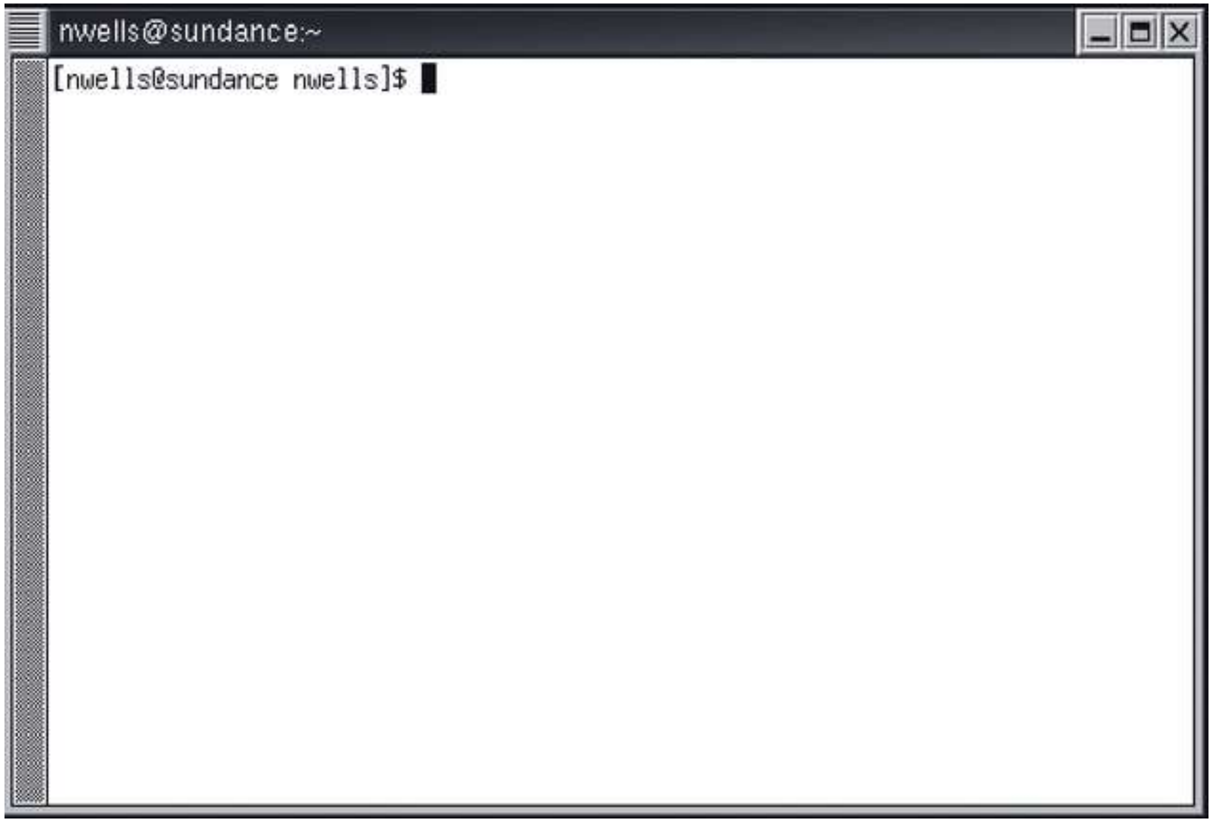
## Understanding the Shell

- Command interpreter: program that accepts input from the keyboard and launches commands or otherwise controls the computer system
- Linux command interpreter is called the shell
- The shell is only loaded when a user logs in at a text mode login prompt
- Different types of shells are available for Linux

## The Shell Prompt

- Shell prompt: set of words or characters indicating that the shell is ready for commands
- The default shell prompt includes four components:
  - The user account name
  - The hostname
  - The last part of the full directory path
  - A prompt character

## A Standard Shell Prompt

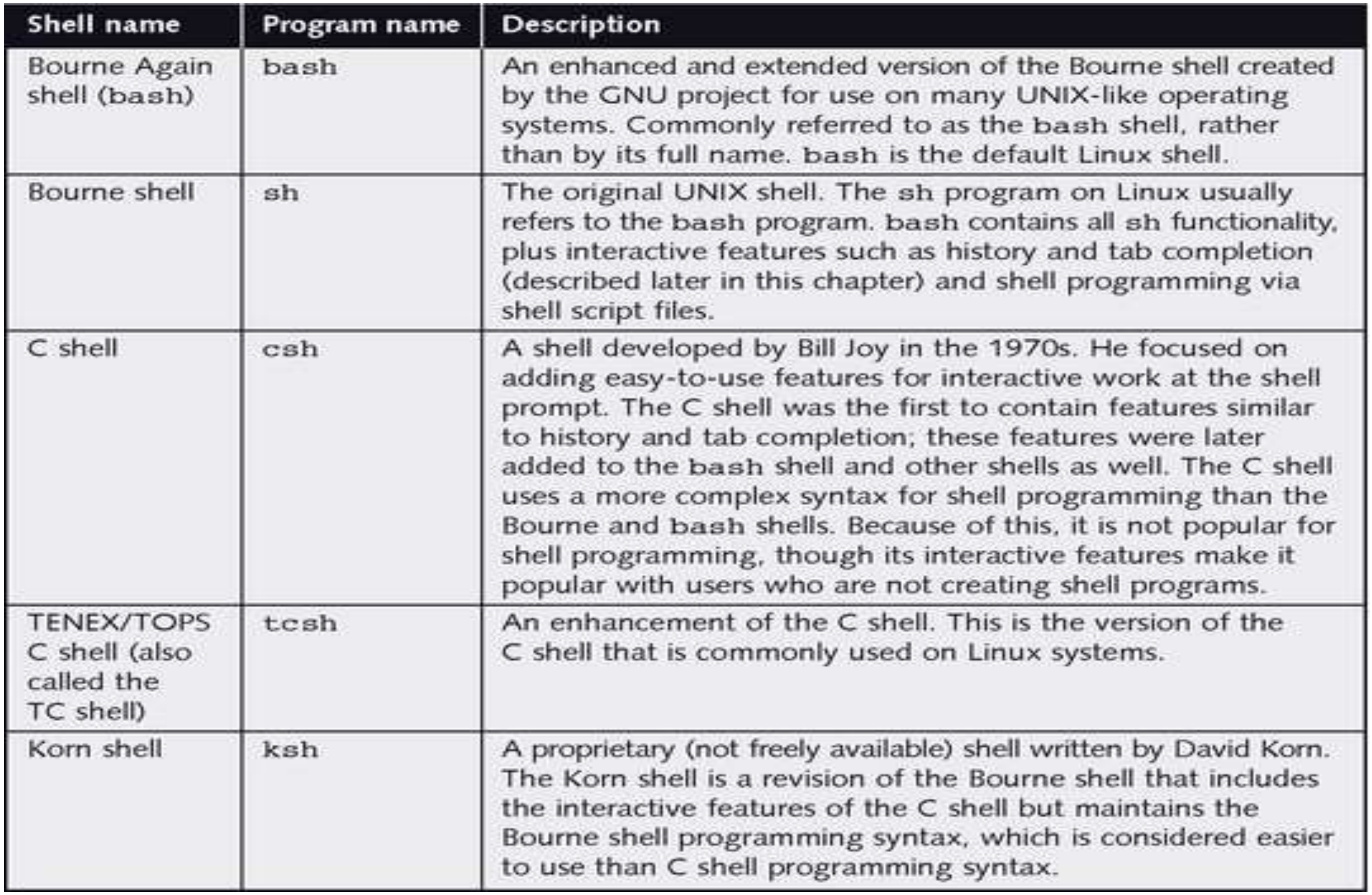


```
[nwells@sundance nwells]$
```

# The Functions of a Shell

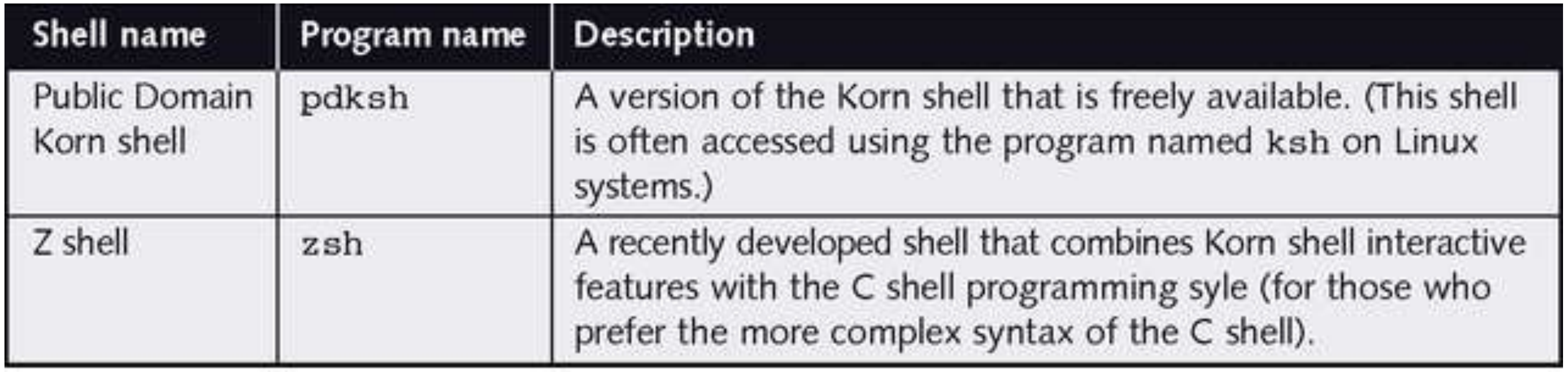
- A shell’s primary purpose is to launch programs
- A Linux shell gives users the ability to write scripts that the shell can execute
- The shell has many built-in features to work with files and commands on a Linux system

# Different Types of Shells



| Shell name | Program name | Description |
|---|---|---|
| Bourne Again shell (bash) | bash | An enhanced and extended version of the Bourne shell created by the GNU project for use on many UNIX-like operating systems. Commonly referred to as the bash shell, rather than by its full name. bash is the default Linux shell. |
| Bourne shell | sh | The original UNIX shell. The sh program on Linux usually refers to the bash program. bash contains all sh functionality, plus interactive features such as history and tab completion (described later in this chapter) and shell programming via shell script files. |
| C shell | csh | A shell developed by Bill Joy in the 1970s. He focused on adding easy-to-use features for interactive work at the shell prompt. The C shell was the first to contain features similar to history and tab completion; these features were later added to the bash shell and other shells as well. The C shell uses a more complex syntax for shell programming than the Bourne and bash shells. Because of this, it is not popular for shell programming, though its interactive features make it popular with users who are not creating shell programs. |
| TENEX/TOPS C shell (also called the TC shell) | tcsh | An enhancement of the C shell. This is the version of the C shell that is commonly used on Linux systems. |
| Korn shell | ksh | A proprietary (not freely available) shell written by David Korn. The Korn shell is a revision of the Bourne shell that includes the interactive features of the C shell but maintains the Bourne shell programming syntax, which is considered easier to use than C shell programming syntax. |

## Different Types of Shells



| Shell name | Program name | Description |
|---|---|---|
| Public Domain Korn shell | `pdksh` | A version of the Korn shell that is freely available. (This shell is often accessed using the program named `ksh` on Linux systems.) |
| Z shell | `zsh` | A recently developed shell that combines Korn shell interactive features with the C shell programming syle (for those who prefer the more complex syntax of the C shell). |

## Shell Programming

- A shell program is a series of linux commands and a shell script is like a batch file which can take input from the user or file and output them on screen.
- A shell is command language interpreter that executes commands read from the standard input device or from a file and it uses the system kernel to execute programs.
- We can create our own commands using shell script which will save time and can  be used in automating some daily routine tasks.
- $ cat /etc/shells – to find all available shells I the system

## Shell script

- A shell script can be defined as a series of command written in plain text file and is very interactive.
- Instead of giving commands one by one, we can save this sequence of commands in a text file and instruct the shell to execute this text file. This is known as shell script.

## Advantages of shell scripts

- A shell script can take input from the user, file and output them on the screen.
- It is useful to create your own commands.
- It is time saving
- It can automate routine tasks.
- System automation can also be automated using shell scripts.

## Shell programming in bash

- Most useful and recommended shell is the **bash** shell because these scripts will be portable between machines, distributions and operating systems.
- There are 2 primary ways to use shell
  - **Interactively –** here the user types a single command and the result is printed out on the screen.
  - **Writing shell scripts** – here the user types anything from a few lines to an entire program into a text editor, then execute the resulting file as a shell script.

## Common commands

- **$** - to refer normal user session
- # - to refer a root session
- **$ Whoami** – displays the name of current user.
- **$ pwd –** prints  the current working directory. Eg : /home/bca08
- **$ cd~** - to go to home directory (if we are in a particular sub directory and we can go to home directory directly by giving this or $cd /home/bca08. ~ is used to refer home directory.)

- ls – to list out the contents of current directory
- Man command – to get more details about a command. Eg : man ls
- find  - used to find files by name, type.    Eg : find .-name '*.jpg'
- tree - list contents of directories in a tree-like format. Eg  tree -d
  - ➢ -a -  All files are printed.
  - ➢ -d -  List directories only.
- file – identifies the file by examining its contents. eg : file fig_small.png

fig_small.png:png image data,128X151,8-bit/color RGB, non-interlaced

- cat – prints the contents of  a file eg: cat
- more – to view screen by screen
- Grep –to search eg :grep the file1.txt
- Tr –translate

  eg : tr “ “ “\n” – for translating all space s by line feeds .

Pipes – eg : echo “cherry apple peach” | tr “ “ “\n” | sort -r

## Redirection

- > - to redirect output to a file
- >> - to append output to a file
- < - to make the contents of a file serve as input of a command
- $ date > datefile.txt – output of date command is sent to datefile.txt. If datefile.txt does not exists then it will be created automatically and of it exists then it will get overwrite.
- $ date >> datefile.txt – append
- wc < datefile.txt – o/p 5 12 58

## Shell scripts

```
#!/bin/bash
# My first script
echo “hello world”
```

- This program displays hello world.
- # - commenting in linux
- #!/bin/bash – this line implies that the file to be executed by bin/bash
- echo – to print
- Type this program in vi editor and save (Esc ->:wq - > enter) .
- Then give 755 permission for this file by using chmod 755 shell1.sh
- For executing - ./shell1.sh
- OR sh filename.sh(here no need to give permission)

- echo “hello     world” – displays all the spaces between hello and world
- echo hello     world – will display only one space between hello and world
- echo “hello \”*\” world” - \(backward slash) is the escape character. Here this / is used to display the * in double quotes. I.e. to get the o/p in the form

hello “*” world

## Shell variables

- There are no data types in shell script.
- Variables in shell can contain numbers, character and strings.
- No need to declare variables
- Eg:

```
var=“testing”
echo $var
```

## Conditional statements

- Conditional statements are mainly used to decide which part of the program to execute depending on specified conditions.

## If Statement

- If statement evaluates a logical expression to make a decision.
- The word **fi** is used to indicate end of **if** statement.

## If - syntax

```
if [condition]
then
    statements
else
    statements
fi
```

**OR**

```
if [condition] ; then
    statements
else
    statements
fi
```

## Nested if

- The if condition can be nested, which means an if condition can have another if condition within it. The else part is executed if none of the expressions specified in the if statement are true.

```
if  [condition ]; then
     statements
elif [condition ]; then
       statements
else
       statements
fi
```

- Eg:

```
read a
if  [ $a = “yes” ]; then
    echo “value is yes”
elif [$a = “no” ]; then
      echo “value is no”
else
     echo “Invalid value”
fi
```

## Case statement

- It is an alternative to nested if statements. Ie. When case statement can be used when there is a large number of conditions to be checked.
- The case statement enables you to compare a pattern with several other patterns and execute a block of code if a match is found.

## Case - synatx

```
case $variable in
   value1)
        statements ;;
   value2)
        statements ;;
   *)
        statements ;;
esac
```

- The case statement compares the value of the variable ($ variable in this case) to one or more values (value1, value2, …). Once a match is found, the associated statements are executed and the case statement is terminated.
- Statements under each condition should end with double semicolon(;;).
- The optional last comparison *) is a default case and will match anything.
- Each clause must be terminated with ";;". Each **case** statement is ended with the **esac** statement.
- Wildcard characters also can be used with case statements

## Examples – case statement

```
echo “enter a number between 1 to 7”
read var
case $var in
1)
        echo "sunday";;
2)
        echo "monday";;
3)
        echo "tuesday";;
*)
        echo “Invalid Day";;
esac
```

- Eg 2.

```
echo "Enter value of a"
read a
case $a in
[1-9])
        echo "Integer";;
[a-z])
        echo “Alphabets";;
*)
        echo "invalid";;
esac
```

- **Eg 3.**

```
read a
case $a in
a|b)
        echo "a or b";;
c|d)
        echo "c or d";;
*)
        echo "invalid";;
esac
```

## Looping or iteration statements

Here a block of code is executed repeatedly until a loop exit condition is satisfied

- For
- While
- Until
- Select
- shift

## For loop

A 'for loop' is a bash programming language statement which allows code to be repeatedly executed. A for loop is classified as an iteration statement i.e. it is the repetition of a process within a bash script.

- **Syntax**

**for** VARIABLE **in** list ; **do**

  Statements

**done**

## Example - for

```
var="1 2 3 4 5 "
for i in $var ; do
        echo $i
done

o/p
1
2
3
4
5
```

- Eg:2

```
for myfile in $(ls); do
        if [ -d $myfile ]; then
                echo "$myfile (dir)"
        else
                echo "$myfile (not a directory)"
        fi
Done
```

- o/p

```
for2.sh (not a directory)
for3.sh (not a directory)
for4.sh (not a directory)
new (dir)
shel1l.sh (not a directory)
shell2.sh (not a directory)
```

## While statement

- It is used to execute a series of commands while a specified condition is true.
- Syntax :

```
while [ condition ] ; do
  Statements
done
```

## Example - while

**Program to print first 10 numbers(0-9)**

```
cnt=0
while [ $cnt -lt 10 ]; do
        echo "The counter is $cnt"
        let cnt=cnt+1
done
```

## Until statement

- It is used to execute a series of commands until a specified condition is true. The loop terminates as soon as the specified condition evaluates to true.
- Syntax :

```
    until [ condition ] ; do
      Statements
    done
```

## Example - until

**Program to print first 10 numbers(0-9)**

```
cnt=0
until [ $cnt -lt 10 ]; do
        echo "The counter is $cnt"
        let cnt=cnt+1
done
```

## Select statement

- It is used to generate a menu list, that expects online input from the user.
- Syntax :

```
select item in itemlist; do
    statements
done
```

Here the system iterates through each entry in the itemlist and the current value of itemlist is assigned to item for each iteration

## Example - select

```
select item in continue finish; do
        if [ $item = "finish" ]; then
                break
        else
                echo "hi“
                break
        fi
done
```

- o/p

```
1) continue
2) finish
#?

1) continue
2) finish
#? 2
```

Then it will break the execution and command prompt will come

#? 1 – it will print hi and then break the execution and command prompt will come

## explanation

- When the select statement is executed the system displays a menu with numeric choices to the user . In this case, 1 for continue, and 2 for finish. F the user chooses 1, the variable item contains a value of continue; if user chooses 2, the variable item contains a value of finish.

## Shift Statement

- It is used to process positional parameters one at a time from left to right.
- Syntax : **shift [number]**

  here the number is optional. If the number value is empty, it shifts one position to the left.
- The positional parameters are identified as $1,$2,$3 and so on.

- The shift command moves the current values stored in the positional parameters to the left one position. For example, if the values of the current positional parameters are:
  - $1 = -r
  - $2 = file1
  - $3 = file2

and you executed the shift command the resulting positional parameters would be as follows:

  - $1 = file1
  - $2 = file2
- You can also move the positional parameters over more than one place by specifying a number with the shift command.

## Shift - a program to print all the command line parameters

```
until [ -z $1 ] ; do
    echo $1
    shift
  done
```

run this program using ./shift_pgm.sh file1 file2

**o/p**

```
file1
file2
```

# Using test

-
  ```
  if test $1 -gt 0
  then
  echo "$1 number is positive"
  fi
  ```

## Integer Operators

- int1 -eq int2 - Returns True if int1 is equal to int2.
- int1 -ge int2 - Returns True if int1 is greater than or equal to int2.
- int1 -gt int2 - Returns True if int1 is greater than int2.
- int1 -le int2 - Returns True if int1 is less than or equal to int2.
- int1 -lt int2 - Returns True if int1 is less than int2.
- int1 -ne int2 - Returns True if int1 is not equal to int2

- str1 = str2  - Returns True if str1 is identical to str2.
- str1 != str2  - Returns True if str1 is not identical to str2.
- str  - Returns True if str is not null.
-  -n str - Returns True if the length of str is greater than zero.
- -z str  - Returns True if the length of str is equal to zero.

## Parameter passing & Arguments

- Global variables or environment variables are defined in shell programming using export keyword.
- Once exported, further changes in that variable value will be automatically affected.
- *Syntax:*
  export variable1, variable2,.....variableN
- *Examples:*

```
$ str=Bus
$ echo $str
Bus
$ export str
$ /bin/bash
$ echo $ str
Bus
$ exit
$ echo $ str
Bus
```

## path variables

- PATH defines a list of directories to search through when looking for a command to execute.
- PATH contains a list of directories separated by colons:

Eg : **echo $PATH**

```
/bin:/usr/bin:/usr/local/bin
```

## Scope of variables

### Local Variables

- Local variables are created using the keyword local. local variables will have scope only in the function where it is declared.

Eg : local variable_name

## Example – local variables

```
var="hello"
function test_fn()
{
    local var="world"
    echo $var
}
echo $var
test_fn
echo $var
```

**o/p**

```
hello
World
hello
```

- In the above example if we don’t use the keyword local for the variable ‘var’ inside the function, then the output will be

**o/p**

```
hello
world
world
```

## Examples

```
echo "what is your name"
read user_name
echo "hello $user_name"
echo "i will create a file called $user_name"
touch "${user_name}_file“
```

**o/p**

First it will ask the user to input your name. After inputting the name it will create an empty file named ‘yourname_file’

## Built-in shell variables.

When variables are used they are referred to with the $ symbol in front of them. There are several useful variables available in the shell program. Here are a few:

- $$ = The PID number of the process executing the shell.
- $! = PID of last run background process
- $? = Exit status variable (contains the exit value of the last run command).
- $0 = The name of the command you used to call a program.
- $1 = The first argument on the command line.
- $2 = The second argument on the command line.
- $n = The nth argument on the command line.
- $* ($@) = All the arguments on the command line.
- $# The number of command line arguments.

## Example program for passing parameters

```
echo "i was called with $# parameters"
echo "my name is $0
echo "my first parameter is $($1)"
echo "my second parameter is $2"
echo "all parameters are $@“
```

o/p

Run by sh var2.sh ls file1

```
I was called with 2 parameters
My name is var2.sh
My first parameter is
case1.sh
For1.sh
New
My second parameter is file1
All parameters are ls file1
```

## Using Default values

- By using curly braces ({}) and the special ‘:-’ usage, we can specify default values to a variable if the variable is unset.
- The syntax ‘:=‘, sets the variable to the default value if it is undefined.
- -en – tells not a line break

## Example – default values

```
echo -en “your name "
read myname
echo "name is : ${myname:-ann}“
```

**o/p**

**Your name - ** press enter without typing anything

**Name is : ann**

**OR**

**Your name smitha**

**Name is : smitha**

## Using backtick

```
echo -en “your name "
read myname
echo "name is : ${myname:-`whoami`} “
```

**o/p**

**Your name - ** press enter without typing anything

**Name is : bca08**

**OR**

**Your name smitha**

**Name is : smitha**

## Example - Using :=

```
echo -en “your name "
read myname
echo "name is : ${myname:=`whoami`} “
echo “name is : $myname
```

**o/p**

```
Your name    - press enter without typing anything
Name is : bca08
Name is : bca08
```

**If we use** `echo "name is : ${myname:-`whoami`} “`

**o/p**

```
Your name    - press enter without typing anything
Name is : bca08
Name is :
```

## Using Default Values – no need

```
echo -en “your name [ `whoami` ]"
read myname
if [ -z "$myname" ]; then
        myname=`whoami`
fi
echo your name is : $myname
```

**o/p**

```
Your name [ bca08 ]
```

without typing the name press enter then the output will be

```
Your name is : bca08
```

**OR**

```
Your name [ bca08 ]  anna
Your name is : anna
```

## External Programs

- Backtick (`) is often associated with external programs.
- Bcktick is used to indicate that the enclosed text is to be executed as a command.
- Eg:
- html_files = `find / -name “*.html`

## Functions

- whenever programs gets complicated, we divide it into small entities which is know as function.
- Function performs particular activity in shell and can return a single value .
- A return statement will terminate the function
- Parameters are passed into the function via numbered arguments just like scripts.
- Function definition statements (code inside curly braces { }) is executed only when the function is called.

## Syntax - functions

- *Syntax:*

```
function-name ( )
{
    command1
    command2
    .....
    ...
     commandN
    return
}
```

## Eg :1

```
#function defined
simple_fn()
{
        #print hello world
        echo "Hello World";
}
#calling function
simple_fn
```

## Eg :2(passing parameters)

```
#function defined
simple_fn()
{
        #print hello world
        echo “first argument is : “ . $1;
        echo “second argument is : “ . $2;
}
#calling function
simple_fn smitha jose
```

**o/p**

```
first argument is : smitha
second argument is : jose
```

## Scope of variables in functions

```
myfn()
{
echo "function called with : $@"
 x=2
}
echo "script called with : $@"
x=1
echo "x is : $x"
myfn 1 2 3
echo "x is : $x“
```

**o/p** : run by sh fnscope.sh a b

```
Script called with a b
X is : 1
Function called with 1 2 3
X is :2
```

## Return functions

```
myfn()
{
echo "function called with : $@"
 x=2
}
echo "script called with : $@"
x=1
echo "x is : $x"
val=$(myfn 1 2 3)
echo "val is : $val"
echo "x is : $x“
```

**O/p : here last x value will be 1 not 2. if we give an echo of x inside function then it will print 2**

## Explanation - function

- The $@  parameters are changed within the function to reflect how the function was called. The variable x is a global variable and myfn changed the value of that variable and that change is still effective when control returns to main script.