

2.4 Booting Sequence

when the system starts, it perform following tasks:

- **ROM diagnostics are run:** here hardware and memory tests are performed.
- **boot loader is loaded:** the ROM BIOS reads the boot loader from boot block and loads into RAM.
- **kernel is loaded:** the kernel program is called by user. Recall that the user needs to enter the name of the program (e.g. UNIX).
by default, UNIX is loaded into the memory by bootstrap program.
now, the control is hand over to the kernel.
- **kernel initialization take place:** it involves performing memory tests, initializing the devices through device driver, swapper scheduler process, the init process and many more.
- The init program resides in /etc directory which is invoked by the kernel. this program takes over the system control.
- **The init program:** this process has the PID (Process Identification) 1 which is the second process of the system. The init program reads the instruction of /etc/inittab file to carry out processes like identifying the run level i.e. the mode in which system should run single user/multi-user, maintaining files to track activities etc.
- **the getty process:** the init program also invokes the getty (it is responsible for the print the login prompt on the respective terminals and then goes off to sleep) program which establishes a communication with the terminals of the unix system. the getty programs uses the /etc/gettydefs for instructions to provide the login: prompt at each terminals connected to system and goes into suspended(sleep) mode and activated when user attempts to login.

- **The login program:** once the user types login name and password, getty transfer control to a login programs, to verify the log-in name and password entered by user. If log-in is successful the \$ prompts appears on the screen. it is essential to know that it is the /etc/profile that enables the system administration to provide information like time and date, system name, number of user etc. this file can be modified by the system administrator to accommodate relevant messages for the user.

init ----> getty ---->login----->shell

- summary of startup process.

Function	Program	File involved
It identifies run level (single/multi-user), maintains files, invokes getty program.	Init (it reads inittab files)	/etc/inittab
It provides login: prompt, accepts login name and password, invokes login program.	Getty	/etc/gettydefs
It compares login name, checks for password validity, runs /etc/profile program and profile program.	Login	/etc/passwd /etc/profile .profile

INIT PROCESS:

In **Unix-based computer operating systems**, **init** (short for *initialization*) is the first **process** started during **booting** of the computer system.

Init is a **daemon** process that continues running until the system is shut down.

It is the direct or indirect **ancestor** of all other processes and automatically adopts all **orphaned processes**.

Init is started by the **kernel** using a **hard-coded filename**, and if the kernel is unable to start it, a **kernel panic** will result.

Init is typically assigned **process identifier 1**.

Structure of /etc/passwd

- all user information except the encrypted password is stored in /etc/passwd.
- /etc/passwd file stores essential information, which is required during login i.e. user account information.
- /etc/passwd is a text file, which contains a list of the system's accounts, giving for each account some useful information like user ID, group ID, home directory, shell, etc.
- It should have general read permission as many utilities like ls use it to map user IDs to user names, but write access only for the superuser/root account.
- the structure of /etc/passwd file:

**username : password : UID : GID : Comment :
home directory : login shell.**

```
oracle:x:1021:1020:Oracle user:/data/network/oracle:/bin/bash
```

1 2 3 4 5 6 7

- Significance of all seven fields is as follow:

1. **Username:** It is used when user logs in. It should be between 1 and 32 characters in length.
2. **Password:** An x character indicates that encrypted password is stored in /etc/shadow file. Please note that you need to use the passwd command to computes the

hash of a password typed at the CLI or to store/update the hash of the password in /etc/shadow file.

3. **User ID (UID):** Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups.
4. **Group ID (GID):** The primary group ID (stored in /etc/group file)
5. **User ID Info:** The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. This field use by finger command.
6. **Home directory:** The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /
7. **Login Command/shell:** The absolute path of a command or shell (/bin/bash). Typically, this is a shell. Please note that it does not have to be a shell.

Structure of /etc/shadow

- The **/etc/shadow** file stores actual password in encrypted format (more like the hash of the password) for user's account with additional properties related to user password.
- Basically, it stores secure user account information. All fields are separated by a colon (:) symbol. It contains one entry per line for each user listed in /etc/passwd file Generally, shadow file entry looks as follows (click to enlarge image):
- General Structure of this file :

username : pwd :Last pwd Change: minimum : maximum : warn : Inactive : expire : reserve.

/etc/shadow file fields

```
vivek:$1$Infflc$PgtEyHdicpGOffXX4ow#5:13064:0:99999:7:::
```

The diagram shows the following mapping:

1	2	3	4	5	6
---	---	---	---	---	---

(Fig.01: /etc/shadow file fields)

1. **Username** :It is your login name.
2. **Password** : It is your encrypted password. The password should be minimum 8-12 characters long including special characters, digits, lower case alphabetic and more. Usually password format is set to \$id\$salt\$hashed, The \$id is the algorithm used On GNU/Linux as follows:
 - a. **\$1\$** is MD5
 - b. **\$2a\$** is Blowfish
 - c. **\$2y\$** is Blowfish
 - d. **\$5\$** is SHA-256
 - e. **\$6\$** is SHA-512
3. **Last password change (last changed)** : Days since Jan 1, 1970 that password was last changed
4. **Minimum** : The minimum number of days required between password changes i.e. the number of days left before the user is allowed to change his/her password
5. **Maximum** : The maximum number of days the password is valid (after that user is forced to change his/her password)
6. **Warn** : The number of days before password is to expire that user is warned that his/her password must be changed
7. **Inactive** : The number of days after password expires that account is disabled
8. **Expire** : days since Jan 1, 1970 that account is disabled i.e. an absolute date specifying when the login may no longer be used.
9. **Reserve**: it is a reserve field.

startup and shutdown files

2.5 File Access Permissions

Here, we will discuss in detail about file permission and access modes in Unix.

File ownership is an important component of Unix that provides a secure method for storing files.

Every file in Unix has the following attributes –

- **Owner permissions** – The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** – The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** – The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

ls command display list of files.

```
Administrator@ADMIN ~  
$ ls  
f1.txt f4.txt neha
```

While using ls -l command, it displays various information related to file permission as follows

```
$ls -l /home/amrood  
-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10  
myfile  
drwxr-xr--- 1 amrood users 1024 Nov 2 00:10  
mydir
```

```
Administrator@ADMIN ~
$ ls -l
total 4
-rw-rw-r--+ 1 Administrator None 0 Jun 13 11:55 f1.txt
-rw-rw-r--+ 1 Administrator None 0 Jun 13 11:55 f4.txt
drwxrwxr-x+ 1 Administrator None 0 Jun 22 11:29 neha

Administrator@ADMIN ~
$
```

[neha@linux neha]\$ ls -l /home/neha

```
total 56
-rw-rw-r-- 1 neha neha 19 Jul 30
2018 1
-rw-rw-r-- 1 neha neha 2 Jul 30 2018
2
-rw-rw-r-- 1 neha neha 2 Jul 26 2018
5
-rw-rw-r-- 1 neha neha 168 Sep 20
2018 a.awk
drwxrwxr-x 2 neha neha 4096 Jul 1
02:29 bca
-rw-rw-r-- 1 neha neha 741 Sep 20
2018 emp.lst
-rwxrwxr-x 1 neha neha 229 Sep 6
2018 math.sh
drwxrwxr-x 5 neha neha 4096 Sep 5
2018 n
-rw-rw-r-- 1 neha neha 51 Jul 26
2018 num
-rwxrwxr-x 1 neha neha 225 Sep 5
2018 q1.sh
-rwxrwxr-x 1 neha neha 170 Sep 6
2018 q11.sh
-rwxrwxr-x 1 neha neha 210 Sep 6
2018 rev.sh
drwxrwxr-x 3 neha neha 4096 Jun 18
05:18 unix1
```

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) –

The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwx**–xr-- represents that the owner has read (r), write (w) and execute (x) permission.

The second group of three characters (5-7) consists of the permissions for the group to which the file belongs.

For example, **-rwx**–**xr**-- represents that the group has read (r) and execute (x) permission, but no write permission.

The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwx**–xr-- represents that there is read (r) only permission.

File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system.

The basic building blocks of Unix permissions are the read, write, and execute permissions, which have been described below –

Read (r): Grants the capability to read, i.e., view the contents of the file.

Write (w): Grants the capability to modify, or remove the content of the file.

Execute (x): User with execute permissions can run a file as a program.

Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned –

Read: Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

Write: Access means that the user can add or delete files from the directory.

Execute: Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have execute access to the bin directory in order to execute the ls or the cd command.

Changing Permissions

To change the file or the directory permissions, you use the chmod (change mode) command.

There are two ways to use chmod — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

S.No.	Chmod operator & Description
1	+ Adds the designated permission(s) to a file or directory.
2	- Removes the designated permission(s) from a file or

	directory.
3	= Sets the designated permission(s).

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--X
2	Write permission	-W-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-WX
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-X
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwX

Unix session

- Unix session is a time period between logging in and logged out.
- when user is successfully logged in to the system then the session is started and terminated as soon as user logged out from the system.

- **Starting Unix session- Logging in**

- Before beginning, make sure your Caps Lock key is off.
- On most keyboards it is above your left Shift key.
- To log into your Unix account:

1. At the Login: prompt, enter your username.

e.g. : **login: bca01**

2. At the Password: prompt, enter your password.

e.g.: password:**** [enter]

For security reasons, your **password does not appear on the screen** when you type it. If you enter an incorrect password, you'll be asked to enter your username and password again.

(Be aware that the Backspace or Del keys might not work properly while you are entering your password.)

3. After then Unix shell prompt will appear.

like \$

4. You can now enter commands at the Unix prompt.

- **Starting Unix session- Logging out**

- Logging out of UNIX may be achieved simply by typing logout, or <ctrl-D> or exit.

E.g.: \$<ctrl-d>

login:

- All three terminate the login shell and , in the former case, the shell performs commands from the .bash_logoutfile in your home directory.
- Exit is a C function that kills the calling process and circumvents all cleanup.

- When logging out of the console, you should log out of each window before exiting from your window manager.

- **command Line Structure:**

A command provide an interaction between the user and shell.

A command is a program that tells the unix system to do something.

Syntax: **command** [**options**] [**arguments**]

where an argument indicates on what the command is to perform its action, usually a file or series of files.

An option modifies the command, changing the way it performs.

Commands are case sensitive. e.g.:
command and Command are not the same.

Options are generally preceded by a hyphen (-), and for most commands, more than one option can be strung together, in the form:

command -[option][option][option]

e.g.: **ls -alR**

will perform a long list on all files in the current directory and recursively perform the list through all sub-directories.

For most commands you can separate the options, preceding each with a hyphen, e.g.:

command -option1 -option2 -option3

as in:

ls -a -l -R

Some commands have options that require parameters. Options requiring parameters are usually specified separately, e.g.:

lpr -Pprinter3 -# 2 file
will send 2 copies of file to printer3.

These are the standard conventions for commands. However, not all Unix commands will follow the standard. Some don't require the hyphen before options and some won't let you group options together, i.e. they may require that each option be preceded by a hyphen and separated by whitespace from other options and arguments. Options and syntax for a command are listed in the man page for the command.

1. List Files using ls with no option

ls with no option list files and directories in bare format where we won't be able to view details like file types, size, modified date and time, permission and links etc.

```
# ls
0001.pcap      Desktop      Downloads
index.html    install.log.syslog  Pictures
Templates
anaconda-ks.cfg  Documents
fbcmd_update.php  install.log  Music
Public        Videos
```

2 List Files With option -l

Here, ls -l (-l is character not one) shows file or directory, size, modified date and time, file or folder name and owner of file and its permission.

```
# ls -l
```

```
total 176
-rw-r--r--. 1 root root 683 Aug 19 09:59
0001.pcap
-rw----- 1 root root 1586 Jul 31 02:17
anaconda-ks.cfg
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Desktop
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Documents
drwxr-xr-x. 4 root root 4096 Aug 16 02:55
Downloads
-rw-r--r--. 1 root root 21262 Aug 12 12:42
fbcmd_update.php
-rw-r--r--. 1 root root 46701 Jul 31 09:58
index.html
-rw-r--r--. 1 root root 48867 Jul 31 02:17
install.log
-rw-r--r--. 1 root root 11439 Jul 31 02:13
install.log.syslog
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Music
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Pictures
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Public
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Templates
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Videos
```

3. View Hidden Files

List all files including hidden file starting with ‘.’.

```
# ls -a
```

```

.                .bashrc  Documents
.gconfd          install.log
.nautilus        .pulse-cookie
..              .cache   Downloads
.gnome2          install.log.syslog
.netstat.swp     .recently-used.xbel
0001.pcap        .config  .elinks
.gnome2_private .kde     .opera
.spice-vdagent
anaconda-ks.cfg .cshrc   .esd_auth
.gtk-bookmarks  .libreoffice
Pictures        .tcshrc
.bash_history    .dbus    .fbcmd
.gvfs           .local   .pki
Templates
.bash_logout    Desktop  fbcmd_update.php
.ICEauthority   .mozilla Public
Videos
.bash_profile    .digrc   .gconf
index.html       Music    .pulse
.wireshark

```

4. List Files with Human Readable Format with option -lh

With combination of -lh option, shows sizes in human readable format.

```

# ls -lh
total 176K
-rw-r--r--. 1 root root 683 Aug 19 09:59
0001.pcap
-rw-----. 1 root root 1.6K Jul 31 02:17
anaconda-ks.cfg
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Desktop
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Documents

```

```
drwxr-xr-x. 4 root root 4.0K Aug 16 02:55
Downloads
-rw-r--r--. 1 root root 21K Aug 12 12:42
fbcmd_update.php
-rw-r--r--. 1 root root 46K Jul 31 09:58
index.html
-rw-r--r--. 1 root root 48K Jul 31 02:17
install.log
-rw-r--r--. 1 root root 12K Jul 31 02:13
install.log.syslog
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Music
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Pictures
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Public
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Templates
drwxr-xr-x. 2 root root 4.0K Jul 31 02:48
Videos
```

List Files in Reverse Order

The following command with `ls -r` option display files and directories in reverse order.

```
# ls -r
Videos      Public      Music
install.log fcmd_update.php Documents
anaconda-ks.cfg
Templates  Pictures    install.log.syslog
index.html Downloads   Desktop
0001.pcap
```

Recursively list Sub-Directories

`ls -R` option will list very long listing directory trees. See an example of output of the command.

```
# ls -R
```



```

total 1384
-rw----- . 1 root root 33408 Aug
8 17:25 anaconda.log
-rw----- . 1 root root 30508 Aug
8 17:25 anaconda.program.log
./httpd:
total 132
-rw-r--r-- 1 root root 0 Aug 19 03:14
access_log
-rw-r--r-- . 1 root root 61916 Aug 10 17:55
access_log-20120812
./lighttpd:
total 68
-rw-r--r-- 1 lighttpd lighttpd 7858 Aug
21 15:26 access.log
-rw-r--r-- . 1 lighttpd lighttpd 37531 Aug
17 18:21 access.log-20120819
./nginx:
total 12
-rw-r--r-- . 1 root root 0 Aug 12 03:17
access.log
-rw-r--r-- . 1 root root 390 Aug 12 03:17
access.log-20120812.gz

```

Reverse Output Order

With combination of `-ltr` will shows latest modification file or directory date as last.

```

# ls -ltr
total 176
-rw-r--r-- . 1 root root 11439 Jul 31 02:13
install.log.syslog
-rw-r--r-- . 1 root root 48867 Jul 31 02:17
install.log
-rw----- . 1 root root 1586 Jul 31 02:17
anaconda-ks.cfg

```

```

drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Desktop
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Videos
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Templates
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Public
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Pictures
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Music
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Documents
-rw-r--r--. 1 root root 46701 Jul 31 09:58
index.html
-rw-r--r--. 1 root root 21262 Aug 12 12:42
fbcmd_update.php
drwxr-xr-x. 4 root root 4096 Aug 16 02:55
Downloads
-rw-r--r--. 1 root root 683 Aug 19 09:59
0001.pcap

```

Sort Files by File Size

With combination of `-lS` displays file size in order, will display big in size first.

```

# ls -lS
total 176
-rw-r--r--. 1 root root 48867 Jul 31 02:17
install.log
-rw-r--r--. 1 root root 46701 Jul 31 09:58
index.html
-rw-r--r--. 1 root root 21262 Aug 12 12:42
fbcmd_update.php
-rw-r--r--. 1 root root 11439 Jul 31 02:13
install.log.syslog

```

```

drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Desktop
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Documents
drwxr-xr-x. 4 root root 4096 Aug 16 02:55
Downloads
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Music
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Pictures
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Public
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Templates
drwxr-xr-x. 2 root root 4096 Jul 31 02:48
Videos
-rw----- . 1 root root 1586 Jul 31 02:17
anaconda-ks.cfg
-rw-r--r-- . 1 root root 683 Aug 19 09:59
0001.pcap

```

Display Inode number of File or Directory

We can see some number printed before file / directory name. With `-i` options list file / directory with inode number.

```

# ls -i
20112 0001.pcap          23610 Documents
23793 index.html        23611 Music
23597 Templates
23564 anaconda-ks.cfg  23595 Downloads
22 install.log        23612 Pictures
23613 Videos
23594 Desktop          23585
fbcmd_update.php     35 install.log.syslog
23601 Public

```

```
Administrator@ADMIN ~
```

```
$ ls -l
```

```
3096224743960068 f1.txt 2533274790538755 f4.txt 5629499534368670 neha
```

List Directory Information

With `ls -l` command list files under directory `/tmp`.
Wherein with `-ld` parameters displays information of `/tmp` directory.

```
# ls -l /tmp
```

```
total 408
```

```
drwx----- . 2 narad narad 4096 Aug 2
```

```
02:00 CRX_75DAF8CB7768
```

```
-r----- . 1 root root 384683 Aug 4
```

```
12:28 htop-1.0.1.tar.gz
```

```
drwx----- . 2 root root 4096 Aug 4
```

```
11:20 keyring-6Mfjnk
```

```
drwx----- . 2 root root 4096 Aug 16
```

```
01:33 keyring-pioZJr
```

```
drwx----- . 2 gdm gdm 4096 Aug 21
```

```
11:26 orbit-gdm
```

```
drwx----- . 2 root root 4096 Aug 19
```

```
08:41 pulse-gl6o4ZdxQVrX
```

```
drwx----- . 2 narad narad 4096 Aug 4
```

```
08:16 pulse-UDH76ExwUVoU
```

```
drwx----- . 2 gdm gdm 4096 Aug 21
```

```
11:26 pulse-wJtcweUCtvhn
```

```
-rw----- . 1 root root 300 Aug 16
```

```
03:34 yum_save_tx-2012-08-16-03-
```

```
34LJTAa1.yumtx
```

```
# ls -ld /tmp/
```

```
drwxrwxrwt. 13 root root 4096 Aug 21 12:48
```

```
/tmp/
```

Display UID and GID of Files

To display UID and GID of files and directories. use option `-n` with `ls` command.

```
# ls -n
total 36
drwxr-xr-x. 2 500 500 4096 Aug  2 01:52
Downloads
drwxr-xr-x. 2 500 500 4096 Aug  2 01:52
Music
drwxr-xr-x. 2 500 500 4096 Aug  2 01:52
Pictures
-rw-rw-r--. 1 500 500  12 Aug 21 13:06
tmp.txt
drwxr-xr-x. 2 500 500 4096 Aug  2 01:52
Videos
```

Unix Commands

In Unix, some commands work on file, directory or both. Unix commands is categorized into the following 4 different categories:

1. Directory command
2. File commands
3. Directory and file commands
4. other useful commands

1. Directory command

- a. **pwd** : Print the name of the working directory. It will print the full system path of the current working directory to standard output. It doesn't accept any arguments.

Syntax : pwd

ans : /home/bca1

pwd prints the full pathname of the current working directory.

Options

- L,** If the environment variable **PWD** contains an absolute
- logical** name of the current directory with no "." or ".." components, then output those contents, even if they contain symbolic links. Otherwise, fall back to default (**-P**) behavior.
- P,** Print a fully resolved name for the current directory, in which
- all components of the name are actual directory names, and **physical** not symbolic links.
- help** Display a help message, and exit.
- Display version information, and exit.
- version**

b. **cd**: The command `cd directory` means change the current working directory to '*directory*'.

The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree. It is equivalent to **chdir** command.

When `cd` command is run without any arguments or `path-name` then always takes the user back to his home directory.

Syntax: `cd [path-name/directory name]`

```
e.g.: $cd /usr/local/bin
$
```

OPTIONS WITH CD COMMAND:

`cd`: it change to user's home directory

`cd/` : it changes the directory to the system's root.

`cd..` : it goes up one directory level up.

`cd../..` : it goes up 2 directory levels.

cd~ : it switches to home directory to the user which is similar to cd without any argument.

- c. **mkdir: it stands for make directories. it is used to create one or more new directories. it is same as ms-dos md or mkdi command.**

```
syntax:  mkdir [options] path
name/directories-name [path name/directories-
name]
```

e.g. : mkdir bca01

OR

```
mkdir /tmp/morestuff [enter]
```

we can create more than one directory as follow:

```
mkdir dir1 dir2 dir3
```

options with md command:

Option	Description
-m mode	<p>This option sets the permission mode of new directory or we can say Sets the access mode for the new directory.</p> <p>To create a directory named <i>myprg</i> having only read permission to owner, group and other user the command is</p> <p>\$mkdir -m444myprg OR \$mkdir -m a=r myprg</p> <p>To create a directory named <i>temp</i> having initial,o=r temp read permission, write to owner and read to other user the command is</p> <p>\$mkdir -m u=rw</p>

Option	Description
-p	<p>This option create a sub directory tree under the current directory.</p> <p>If the parent directories don't exist, this command creates them.</p> <p>e.g.: \$mkdir sem5;cd sem5</p> <p style="padding-left: 40px;">\$mkdir unix;cd unix</p> <p style="padding-left: 40px;">\$mkdir script;cd../..</p> <p>They create directory named sem5,unix in sem5 anf finally script in unix.</p> <p>Sem5->unix->scripts</p> <p>This is done by using -p option as follow</p> <p style="padding-left: 40px;">\$mkdir -p sem5/unix/script</p> <p>\$</p>
-v	<p>If a directory created successfully then this optin shows the name of the directory.</p> <p>e.g.: \$mkdir -v d2</p> <p>d2</p> <p>\$</p>

d. rmdir : The rmdir command is used to remove directories. It stands for remove directories.
syntax : rmdir [option] directory_names
e.g.: \$rmdir mydir [enter]

To remove more than one directories in current directory

e.g.: `$rmdir dir1 dir2 dir3`

To remove more than one empty directories e.g.:
`$rmdir dir1/dir2/dir3`

options with `rmdir` command:

Mode	DESCRIPTION
-p, --parents	This option remove directories and any intervening parent directories that become empty as a result, useful for removing sub directories. remove DIRECTORY and its ancestors; e.g., <code>\$rmdir -p a/b/c</code>
-v, --verbose	By default <code>rmdir</code> don't display any message after removing the empty directories. This option displays the names of removal empty directories. <code>\$rmdir -v d111</code> Rmdir: removing directory ,d111 \$

e. **ls: ls stands for list**

`ls` is a Linux shell command that lists directory contents of files and directories, by default.

syntax: `ls [options] [argument-list]`

where, `argument-list` should be the file name, directory name or path name.

if we apply `ls` command **without argument** it **display all the files** and directories in the working directories.

e.g.: `$ls file1 [enter]`

`file1`

\$

- we can use multiple file or directory name with ls command like,
\$ls f1 f2 f3
ans: f1 f2 f3
\$

options with ls command:

Option	Description
-a	Displays all files present in current directories. Also list all files including hidden file starting with '.'
-A	It shows all files and those beginning with dot(.) except current directory and parent directory (i.e. double dot (..))
-b	Displays nonprinting characters in octal.
-c	List entries by columns. It displays files listing in multi-column. the output should be stored on column by column. e.g.: ls -c ans: d1 d2 dir f1 f3 file2 t1 d11 d22 f2 file1 file3 \$
-C	Displays files in a columnar format (default)
-d	Displays only directories. List directory entries instead of contents, and do not dereference symbolic links.
-f	Interprets each <i>name</i> as a directory, not a file.
-F	Append indicator (one of */=>@) to entries. This option is useful to identifying the directories, executable or symbolic link files. This option appear character to file name

Option	Description
--------	-------------

indicating the file type.
 The file type '*' for executable directories, '@' for symbolic links, '|' for FIFO, '=' for sockets and nothing for regular files.
 e.g.: ls -F
 ans: d1/ d12/ f1* f11 f1.a*
 \$

-g Displays the long format listing, but exclude the owner name.

-i Displays the i-node for each file.
 e.g.: \$ls -i
 ans: 1732640d111 1732639f1
 \$
 The number precedes the file is i-node number.

Displays the long format listing or detailed information about files or directories.

```

TargetUbuntu01 - VMware Player  File  Virtual Machine  Help
$ ls -la
total 28
drwxr-xr-x 3 student student 4096 2011-04-18 10:36 .
drwxr-xr-x 6 root root 4096 2011-04-13 08:40 ..
-rw-r--r-- 1 student student 220 2010-04-18 18:51 .bash_logout
-rw-r--r-- 1 student student 3103 2010-04-18 18:51 .bashrc
drwxr-xr-x 2 student student 4096 2011-01-08 08:37 .cache
-rw----- 1 student student 109 2011-04-18 10:36 .mysql_history
-rw-r--r-- 1 student student 675 2010-04-18 18:51 .profile
$
  
```

```

% ls -al
total 94
drwxr-xr-x 2 john doc 512 Jul 10 22:25 .
drwxr-xr-x 4 bin bin 1024 Jul 8 11:48 ..
-rw-r--r-- 1 john doc 136 Jul 8 14:46 .exerc
-rw-r--r-- 1 john doc 833 Jul 8 14:51 .profile
-rw-rw-rw- 1 john doc 31273 Jul 10 22:25 ch1
-rw-rw-rw- 1 john doc 0 Jul 10 21:57 ch2
  
```

type	# of links	owner	group	size (in bytes)	modification date and time	name
------	------------	-------	-------	-----------------	----------------------------	------

Above output first line denotes permission of the file. Then no. of links, owner, group of

Option	Description
	owner, size of files , modification date & time, filename.
-L	When showing file information for a symbolic link, show information for the file the link references rather than for the link itself. \$ls -L
-m	Displays the names as a comma-separated list.
-n	<p>It lists numeric user-id and group-id instead of names OR Displays the long format listing, with GID and UID numbers.</p> <p>e.g.: \$ls -n</p> <pre> Total 2 drwxrwxrwx 2 615 502 10 24 oct 11 2012 dir --w---x-w- 2 615 502 49 jul23 16:40 f1 -rwxrw---- 1 615 502 28 jul23 16:40 f2 </pre> <p>Where 615 is uid,502 is gid</p>
-o	Displays the long format listing, but excludes group name.
-p	Displays directories with /
-q	Displays all nonprinting characters as ?
-r	Displays files in reverse order.
-R	<p>Displays subdirectories as well.</p> <pre> sh-4.4\$ ls -r text1.txt test1.txt test.txt file.txt bin bigfile.txt abcde sh-4.4\$ █ </pre>
-s	<p>sorts by file size , largest file display first.</p> <p>\$ls -S</p> <pre> D11 f1 f11.l f3 </pre>

Option	Description
-t	Displays newest files first. (based on timestamp) It sorts by last modification time. Last modified should be display first and older file should be in last.
-u	Displays files by the file access time. last access file should be in first.
-U	It does not sort. it display files in order in which they are sorted in the directory at a time of creation.
-x	Displays files as rows across the screen. It displays files listing in multi-column. the output should be stored on line by line. Sort alphabetically by entry extension. e.g.: <code>\$ ls -x [enter]</code> ans: <code>d1 d11 d2 dir dir1 f1 f2 f3</code> <code>file1 file2 file3</code> <code>\$</code>
-1	Displays each entry on a line OR it lists one file per line.

2. File commands.

cat:

The **cat** command reads one or more files and prints their contents to standard output.

Files are read and output in the order they appear in the command arguments. similar the type command in DOS.

It is the simplest way to display the contents of a file at the command line.

cat is one of the most commonly-used commands in Linux.

It can be used to:

Display text files
Copy text files into a new document
Append the contents of a text file to the end of
another text file, combining them

```
Syntax : cat [options] [files]
```

```
e.g.: $ cat sample.txt  
This is a sample text file
```

```
we can also write like $cat  
/user/user1/file2 [enter]  
ans: this is file 2  
$
```

cat without any arguments then it will take input from standard input device(key board) and display them on standard output device(VDU) until press <ctrl+d> .

```
$ cat [enter]  
ans: this is line1. [enter]  
this is line1  
this is line2 [enter]  
this is line2  
<ctrl+d>
```

Create a File with Cat Command (copy con in DOS)

We will create a file called test2 file with below command.

```
$ cat >test2
```

Awaits input from user, type desired text and press CTRL+D (hold down Ctrl Key and type 'd') to exit. The text will be written in test2 file. You can see content of file with following cat command.

```
$ cat test2  
hello everyone, how do you do?
```

Display Line Numbers in File

With `-n` option you could see the line numbers of a file `song.txt` in the output terminal.

```
$ cat -n song.txt
1  "Heal The World"
2  There's A Place In
3  Your Heart
4  And I Know That It Is Love
5  And This Place Could
6  Be Much
7  Brighter Than Tomorrow
8  And If You Really Try
9  You'll Find There's No Need
10 To Cry
11 In This Place You'll Feel
12 There's No Hurt Or Sorrow
```

6. Display \$ at the End of File

In the below, you can see with `-e` option that '\$' is shows at the end of line and also in space showing '\$' if there is any gap between paragraphs. This options is useful to squeeze multiple lines in a single line.

```
# cat -e test
hello everyone, how do you do?$
$
Hey, am fine.$
How's your training going on?$
$
```

7. Display Tab separated Lines in File

In the below output, we could see TAB space is filled up with '^I' character.

```
# cat -T test
hello ^Ieveryone, how do you do?
Hey, ^Iam fine.
```

```
^I^IHow's your training ^Igoing on?  
Let's do ^Isome practice in Linux.
```

8. Display Multiple Files at Once

In the below example we have three files test, test1 and test2 and able to view the contents of those file as shown above. We need to separate each file with ; (semi colon).

```
# cat test; cat test1; cat test2  
This is test file  
This is test1 file.  
This is test2 file.
```

9. Use Standard Output with Redirection Operator

We can redirect standard output of a file into a new file else existing file with '>' (greater than) symbol. Careful, existing contents of test1 will be overwritten by contents of test file.

```
# cat test > test1
```

10. Appending Standard Output with Redirection Operator

Appends in existing file with '>>' (double greater than) symbol. Here, contents of test file will be appended at the end of test1 file.

```
# cat test >> test1
```

11. Redirecting Standard Input with Redirection Operator

When you use the redirect with standard input '<' (less than symbol), it use file name test2 as a input for a command and output will be shown in a terminal.

```
# cat < test2  
This is test2 file.
```


12. Redirecting Multiple Files Contain in a Single File

This will create a file called test3 and all output will be redirected in a newly created file.

```
# cat test test1 test2 > test3
```

13. Sorting Contents of Multiple Files in a Single File

This will create a file test4 and output of cat command is piped to sort and result will be redirected in a newly created file.

```
# cat test test1 test2 test3 | sort > test4
```

This article shows the basic commands that may help you to explore cat command. You may refer man page of cat command if you want to know more options. In our next article we will cover more advanced cat commands. Please share it if you find this article useful through our comment box below.

Options with cat command

Some of the options available for the **cat** command are:

Option	Description
-b	Starting at 1, number non-blank output lines.
-e	Display control and non-printing characters followed by a \$ symbol at the end of each line. OR it prints \$ to mark end of line. Or display \$ instead of new-line character. \$ cat -ef1 This is a tab character \$ This is control character^[[Z \$
-n	Starting at 1, number all output lines.

Option	Description
-t	<p>Each tab will display as ^I and each form feed will display as ^L.</p> <pre>\$ cat -tf1 This is a tab character^I This is control character^[[Z \$</pre>
-u	Output is displayed unbuffered.
-v	<p>Display control and non-printing characters. Control characters print as ^B for control-B. Non-ASCII characters with the high bit set display as "M-" followed by their lower 7-bit value.</p> <p>Consider a file f1 contain tab character at the end of first line and control character(shift+tab) at the end of 2nd line. if we write a command without option it display contents as follow:</p> <pre>\$ cat f1 This is a tab character This is control character \$</pre> <p>If we use -v option then</p> <pre>\$ cat -vf1 This is a tab character This is control character^[[Z \$</pre>

NOTE: While the options provided here work on most UNIX systems, some UNIX flavors may have changed their meanings or uses. If you experience an incompatibility with these options, please consult

the **cat** manual page (see [man command](#)) on your system for a list of compatible options.

Examples

Command	Explanation
<code>cat file1.txt</code>	Display contents of file
<code>cat file1.txt file2.txt</code>	Concatenates 2 text files and will show them in the terminal
<code>cat file1.txt file2.txt > newcombinedfile.txt</code>	Concatenates 2 text files and writes them to a new file
<code>cat >newfile.txt</code>	Creates a file called newfile.txt - type the desired input and press CTRL+D to finish. The text will be in file newfile.txt.
<code>cat -n file1.txt file2.txt > newnumberedfile.txt</code>	Some implementations of cat, with option -n, can also number lines
<code>cat file1.txt > file2.txt</code>	Redirects standard output of a file into a new file
<code>cat file1.txt >></code>	Appends standard output

file2.txt	of a file into a new file
<pre>cat file1.txt file2.txt file3.txt sort > test4</pre>	Output of cat command is piped to sort and result will be redirected in a newly created file.
<pre>cat file1 file2 less</pre>	file1 and file2 is redirected as input to another program, less

3. Directory and File commands.

this command concern with command related to directory as well as files.

- a. **rm** : it deletes one or more files/directories. it is equivalent to MS-DOS del or erase command. rm command is silent. it immediately display prompt without any message.

Syntax : rm [option] file(S)/directory(s)

e.g.: \$ rm file1 [enter]

\$ [which remove the current directory named file1]

e.g.: \$ rm f1 f2 f3

\$ [through which we can delete more than one files]

Options with rm command

**-f,
--force**

Ignore nonexistent files, and never prompt before removing. To remove a file, you must have write permission to that file.

If you don't have write permission on the file, you will be prompted (y or n) to override. i.e. it displays prompt for removal of write protected file.

This option is used to remove a file forcibly without displaying write protected message.

-i Displays prompt before every removal.

This command removes files iteratively.

```
$ rm -i d.txt
rm: remove regular empty file
'd.txt'? y
```

```
$ ls
e.txt
```

To keep file press 'n' or hitting <enter> key and to remove press 'Y'

-l Prompt once before removing more than three files, or when removing recursively. This option is less intrusive than -i, but still gives protection against most mistakes.

-- Prompt according to **interactive[=WHEN]** to **WHEN**: never, once (-l), or always (-i). If **WHEN** is not specified, then prompt always.

-r,
-R,
--recursive Remove non empty directories and their contents recursively together with files and sub directories contents.

e.g.: `$ rm -r mydir`

\$
Which removes mydir directory with all files and sub directories in it.
This command \$ rm -r d1d2 d3 removes more than one files/directories.

Usage Notes

If the -i/--interactive=once option is given, and there are more than three files or the -r/-R/--recursive options are specified, rm will prompt before deleting anything. If the user does not respond yes/y/Y to the prompt, the entire command is aborted.

If a file is unwritable, stdin is a terminal, and the -f/--force option is not given, or the -ior --interactive=always option is given, rm prompts the user for whether to remove the file. If the response is not yes/y/Y, the file is skipped.

b. cp : cp stands for copy.

This command is used to copy files or group of files or directory.

It creates an exact image of a file on a disk with different file name.

cp command require at least two filenames in its arguments.

To make a duplicate copy of a file, use the command cp.

Syntax :

**\$ cp [options] source
filename/directoryname dest
filename/directory name**

e.g.: `$ cp file1 file2`
which copies the content of file1 into file2

cp command options

option	description		
<code>cp -a</code>	archive files		
<code>cp -f</code>	force copy by removing the destination file if needed		
<code>cp -i</code>	interactive - ask before overwrite i.e. prompt before over write. <pre>\$ cp -i test.c bak</pre> <pre>cp: overwrite 'bak/test.c'? y</pre>		
<code>cp -l</code>	link files instead of copy. Or it creates an alias name of the files. e.g.: <code>\$ cp -l f1 f1.ln</code>		
<u><code>cp -R</code></u>	recursive copy (including hidden files) it recursively copy an entire directory structure to another directory. e.g.: <code>\$ cp -r old new</code>		
<code>cp -p</code>	Normally copy operation changes access and modification date-time of destination file to current system date. This option preserve these attributes of files. Preserve the specified attributes, separated by a comma. Attributes are: <table border="0"><tr><td>mode</td><td>Preserve file mode bits (as set with chmod), and any ACLs.</td></tr></table>	mode	Preserve file mode bits (as set with chmod), and any ACLs .
mode	Preserve file mode bits (as set with chmod), and any ACLs .		

ownership	Preserve owner and group (as set with chown). Ability to preserve these attributes is restricted in the same way as using chown .
timestamps	Preserve time of last file access and modification (atime and mtime , as set with touch), if possible.
links	Preserve in the destination files any links between the source files. With -L or -H , this option can potentially copy symbolic links as hard links.
context	Preserve SELinux security context of source files, or fail with verbose diagnostics.
xattr	Preserve extended attributes of source files, or fail with verbose diagnostics.
all	Preserve all of the above. Same as specifying all the above attributes individually, with the exception that failing to copy context or xattr will not give an exit status of failure.

If not specified, the default value of *attr_list* is

"mode,ownership,timestamps".

```
e.g.: $ls -l f1;ls -lu f1
-rw-rw-r-- 1 bharat tmtbca 133 jul23 12:01
f1 #modification of date-time
-rw-rw-r-- 1 bharat tmtbca 133 jul23 14:45
f1 #access of date-time
```

```
$cp -p f1 f2;ls -lf2; ls -luf2
-rw-rw-r-- 1 bharat tmtbca 133 jul23 12:01
f1 #modification of date-time
```


Generally, you can get the times through a normal directory listing as well:

- `ls -l` outputs last time the file content was modified, the `mtime`
- `ls -lc` outputs last time of file status modification, the `ctime`
- `ls -lu` outputs last access time, the `atime`

c. **mv** : The `mv` command moves, or renames, files and directories on your filesystem.

- It has 2 functions:
 1. It renames a file/ directory
 2. It moves a group of files to a different directory.
- `mv` doesn't create a copy of the file; it renames it.
- No additional space is consumed on a disk during renaming. This command normally **works silently** means no prompt for confirmation.

Syntax: `mv [option] file/directory names file/directory names`

- Example,
To rename the file `chap1` to `man1`,
`$ mv chap1 man1`
- If the destination file doesn't exist, it will be created.
- `mv` simply replaces the filename in the existing directory entry with the new name.
- A group of files can be moved to a directory. The following command moves 3 files to `progs` directory.
`$ mv chap1 chap2 chap3 progs`

- mv can also be used to rename a directory.

```
$ mv mydir yourdir
```

- f, --force** Always overwrite existing files without prompting. This can be useful if you need to overwrite multiple files whose permissions are read-only; if you don't specify -f, you will be prompted for every file.
- i, --interactive** Prompt before overwriting an existing file, regardless of the file's permissions.

It is used for interactive copying.

The `-i` (interactive) option warns the user before moving the destination file.

- **Example,**

If unit1 exist then mv command prompts for a response.

```
$ mv -i chap1 unit1
```

```
mv: overwrite unit1 (yes/no)? y
```

4. Other useful commands.

- a. **who** : Displays who is **logged on** to the **system**.

The **who** command prints information about all users who are currently logged in. the system administrator can use who command to observe that terminals are being properly utilized or not.

Syntax : `who [option] [argumentlist]`

if we use who without argument, it display 3 column output like below:

```
user@myhost:~$ who
fred pts/0    2015-05-16 15:59 (:0.0)
fred pts/1    2015-05-16 16:01 (:0.0)
mary pts/2    2015-05-17 10:18 (:0.0)
```

1st column indicates user/login name, 2nd indicates unique terminal names given to the user and 3rd

represent what date and time a user logged into the unix system.

```
who am i
```

Displays the same information, but only for the terminal session where the command was issued, for example:

```
alan      pts/3          2013-12-25 08:52
(:0.0)
```

The `whoami` display only login name of user.

```
$whoami
neha
$
```

who command options

Option	Description									
	Print line of column headings/headers <code>\$who -H</code>									
-H	<table><thead><tr><th>NAME</th><th>LINE</th><th>TIME</th></tr></thead><tbody><tr><td>Bca</td><td>pts/o</td><td>Jul 25 13:56</td></tr><tr><td>Neha</td><td>pts/01</td><td>Jul 25 15:53</td></tr></tbody></table>	NAME	LINE	TIME	Bca	pts/o	Jul 25 13:56	Neha	pts/01	Jul 25 15:53
NAME	LINE	TIME								
Bca	pts/o	Jul 25 13:56								
Neha	pts/01	Jul 25 15:53								
-m	Only hostname and user associated with stdin									
-q	All login names and number of users logged on									
-t	Print last system clock change									

-T	Add user's message status as +, – or ?
-u	<p>List users logged in If system administrator wants to more about user who logged in. <code>\$who -Hu</code></p> <pre> NAME LINE TIME IDLE PID COMMENT LOGIN tty2 2014-01-05 10:03 8750 id=2 LOGIN tty1 2014-01-05 10:03 8748 id=1 LOGIN tty3 2014-01-05 10:03 8752 id=3 LOGIN /dev/ttyS1 2014-01-05 10:03 8747 id=v/tt LOGIN tty4 2014-01-05 10:03 8754 id=4 LOGIN tty5 2014-01-05 10:03 8756 id=5 LOGIN tty6 2014-01-05 10:03 8758 id=6 </pre>

b.script: it records interactive session in a specified file. in other words, it creates a log file for the user. whatever command applied by user at command line, output of the command and any error message generated during the user's session are stored in a file. This command is useful to store important task performed by user during the interactive session.

Syntax : script [option] [file name]

```
e.g.: script myfile.txt
```

Logs all results to file myfile.txt. This command opens a subshell and records all information through this session. The script ends when the forked shell exits (e.g., when the user types exit) or when Ctrl-D is typed.

A user can end recording by entering exit command. or press <ctrl+D> like \$ exit

- c. **passwd** : The passwd command is used to change the password of a user account. A normal user can run passwd to change their own password, and a system administrator (the superuser) can use passwd to change another user's password, or define how that account's password can be used or changed. If user type passwd command and press enter key, it display interactive prompt as follow:

```
$ passwd
```

output:

```
$passwd
```

```
Changingpasswordforubuntu.
```

```
(current)UNIXpassword:Enter
```

```
newUNIXpassword:
```

```
RetypenewUNIXpassword:
```

```
passwd: password updated successfully
```

Options

Option	Description
-s	Displays password information.

d. **man** : On Linux and other Unix-like operating systems, man is the interface used to view the system's reference manuals.

Description: man is the system's manual viewer; it can be used to display manual pages, scroll up and down, search for occurrences of specific text, and other useful functions.

Each argument given to man is normally the name of a program, utility or function.

The manual page associated with each of these arguments is then found and displayed.

A section number, if provided, will direct man to look only in that section of the manual. The default action is to search in all of the available sections, following a pre-defined order and to show only the first page found, even if page exists in several sections.

To Display the manual page for the item (program) ls :

```
$ man ls
```

output:

it will display man page of ls command

e. echo: to display a text or string/message on the string.

syntax : \$echo [option][text]

e.g.: \$ echo hello world

ans: hello world

\$

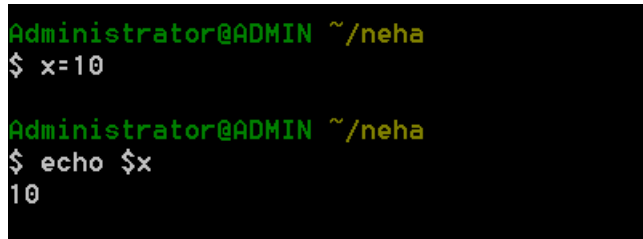
Example-2:

To print value of x, where x=10.

```
$ echo $x
```

output:

10



```
Administrator@ADMIN ~/neha
$ x=10
Administrator@ADMIN ~/neha
$ echo $x
10
```

Example-3:

Use **option '\b' - backspace** with backslash interpretor '-e' removes all the spaces in between.

```
$ echo -e 'Here \bthe \bspaces \bare \bbackspaced.'
```

output:

Herethespacesarebackspaced.

Example-4:

Use option **'\n' - New line with** backspace interpretor '-e' treats new line from where it is used.

```
$ echo -e 'Here \nthe \nspaces \nare \nnewlined.'
```

output:

Here
the
spaces

are
newlined.

Example-5:

Use option **'\t' – horizontal tab** with backspace
interpretor '-e' to have horizontal tab spaces.

```
$ echo -e 'Here \tthe \tspaces \thave \thorizontal  
\ttab \tspaces.'
```

output:

```
Here    the      spaces  have    horizontal  tab  
spaces.
```

Example-6:

Use option **'\v' – vertical tab with backspace
interpretor '-e'** to have vertical tab spaces.

```
$ echo -e 'Here \vthe \vspaces \vhave \vvertical  
\vtab \vspaces.'
```

output:

```
Here  
    the  
        spaces  
            have  
                vertical  
                    tab  
                        spaces.
```

Example-7:

To echo output to a file and not standard output.
in below example output is redirected to test file.

```
$ echo "Hello World!" > test
```

output:

```
$ cat test  
Hello World!
```



```
Administrator@ADMIN ~/neha
$ echo "hi" > f1.txt

Administrator@ADMIN ~/neha
$ cat f1.txt
hi

Administrator@ADMIN ~/neha
$ echo "hello" > f1.txt

Administrator@ADMIN ~/neha
$ cat f1.txt
hello
```

Sequence	Interpreted as
<code>\\</code>	A literal backslash character (" <code>\</code> ").
<code>\a</code>	An alert (The BELL character).
<code>\b</code>	Backspace.
<code>\c</code>	Produce no further output after this.
<code>\e</code>	The escape character; equivalent to pressing the escape key.
<code>\f</code>	A form feed .
<code>\n</code>	A newline .
<code>\r</code>	A carriage return .
<code>\t</code>	A horizontal tab.
<code>\v</code>	A vertical tab.
<code>\0NNN</code>	byte with octal value <i>NNN</i> (which can be 1 to 3 digits).
<code>\xHH</code>	byte with hexadecimal value <i>HH</i> (which can be either 1 or 2 digits)

Options

- n Do not output a trailing **newline**.
- e Enable interpretation of backslash **escape sequences** (see below for a list of these).
- E Disable interpretation of backslash escape sequences. This is the

default.

e.g.: \$echo -e "abc|ndef"

ans: abc

def

\$

f. date : date command is used to display the system date and time. date command is also used to set date and time of the system.

By default the date command displays the date in the time zone on which unix/linux operating system is configured. You must be the super-user (root) to change the date and time.

Syntax:

```
date [OPTION]... [+FORMAT-string]
```

e.g.:

```
$ date
```

```
sat sep 1 12:52:12 IST 2018
```

```
$ [Indian Standard Time]
```

Format options	Purpose of Option	Output
date +%a	Displays Weekday name in short (like Mon, Tue, Wed)	Thu
date +%A	Displays Weekday name in full short (like Monday, Tuesday)	Thursday
date +%b	Displays Month name in short (like Jan, Feb, Mar)	Feb
date +%B	Displays Month name in full short (like January, February)	February
date +%d	Displays Day of month (e.g., 01)	07

date +%D	Displays Current Date; shown in MM/DD/YY	02/07/13
date +%F	Displays Date; shown in YYYY-MM-DD	2013-02-07
date +%H	Displays hour in (00..23) format	23
date +%I	Displays hour (01..12) format	11
date +%j	Displays day of year (001..366)	038
date +%m	Displays month (01..12)	02
date +%M	Displays minute (00..59)	44
date +%S	Displays second (00..60)	17
date +%N	Displays nanoseconds (000000000..999999999)	573587606
date +%T	Displays time; shown as HH:MM:SS Note: Hours in 24 Format	23:44:17
date +%u	Displays day of week (1..7); 1 is Monday	4
date +%U	Displays week number of year, with Sunday as first day of week (00..53)	05
date +%Y	Displays full year i.e. YYYY	2013
date +%Z	alphabetic time zone abbreviation (e.g., EDT)	IS

EXAMPLE-:

To display Weekday name:

```
$ date +%a
```

```
$ date +%A
```

output:

Sunday

EXAMPLE-:

To display Month name:

\$ date +%b

\$ date +%B

output:

January

EXAMPLE-:

To display current day of month:

\$ date +%d

output:

08

EXAMPLE-:

To display Current Date in MM/DD/YY format:

\$ date +%D

output:

01/08/17

EXAMPLE-:

To display date in YYYY-MM-DD format:

\$ date +%F

output:

2017-01-08

EXAMPLE-:

To display time as HH:MM:SS, Note: Hours in 24 Format

\$ date +%T

output:

21:47:05

OPTIONS

TAG	DESCRIPTION
-d, --	display time described by STRING, not

date=STRING	'now'.
-f, --file=DATEFILE	like --date once for each line of DATEFILE
-r, --reference=FILE	display the last modification time of FILE
-R, --rfc-2822	output date and time in RFC 2822 format. Example: Mon, 07 Aug 2006 12:34:56 -0600
--rfc-3339=TIMESPEC	output date and time in RFC 3339 format. TIMESPEC='date', 'seconds', or 'ns' for date and time to the indicated precision. Date and time components are separated by a single space: 2006-08-07 12:34:56-06:00.
-s, --set=STRING	set time described by STRING.
-u, --utc, --universal	print or set Coordinated Universal Time
--help	display this help and exit
--version	output version information and exit

EXAMPLE-1:

To Print cureent system date and time:

```
$ date
```

output:

```
Sun Jan 8 21:38:15 IST 2017
```

EXAMPLE-2:

To print date of next Monday:

```
$ date --date="next mon"
```

output:

```
Mon Jan 9 00:00:00 IST 2017
```

EXAMPLE-3:

To display past date

```
$ date --date="1 day ago"  
$ date --date="yesterday"
```

output:

```
Sat Jan 7 21:39:53 IST 2017
```

EXAMPLE-4:

To display future date

```
$ date --date="1 day"  
$ date --date="tomorrow"  
$ date --date="10 day"
```

output:

```
Wed Jan 18 21:41:26 IST 2017
```

EXAMPLE-5:

To set date:

```
$ date -s "Sun Dec 18 21:00:00 PDT 2016"
```

EXAMPLE-6:

To display Universal Time:

```
$ date -u
```

output:

```
Sun Jan 8 16:13:26 UTC 2017
```

g. cal: it display a calendar.

Syntax: cal[-m]y1 3][month[year]]

Description

Cal displays a simple calendar. If arguments are not specified, the current month is displayed. The options are as follows:

-1

Display single month output. (This is the default.)

-3

Display prev/current/next month output.

-s

Display Sunday as the first day of the week. (This is the default.)

-m

Display Monday as the first day of the week.

-j

Display Julian dates (days one-based, numbered from January 1).

-y

Display a calendar for the current year.

A single parameter specifies the year (1 - 9999) to be displayed; note the year must be fully specified: `cal 89` will *not* display a calendar for 1989. Two parameters denote the month (1 - 12) and year. If no parameters are specified, the current month's calendar is displayed.

A year starts on Jan 1.

e.g.:

```
$cal
```

```
    September 2016
Mo Tu We Th Fr Sa Su
           1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

To output more than one month pass the `-n` option along with the number of months that you wish to show.

```
$cal -n 2
      September 2016                October 2016
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
      1  2  3  4                1  2
  5  6  7  8  9 10 11   3  4  5  6  7  8  9
12 13 14 15 16 17 18   10 11 12 13 14 15 16
19 20 21 22 23 24 25   17 18 19 20 21 22 23
26 27 28 29 30                24 25 26 27 28 29 30
                                31
```

To display feb 2015 calendar

```
$ cal 2 2015
      February 2015
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

OPTIONS

OPTIONS:

-1	Displays single month as output.
-3	Displays prev/current/next month output.
-s	Displays sunday as the first day of the week.
-m	Displays Monday as the first day of the week.
-j	Displays Julian dates (days one-based, numbered from January 1).
-y	Displays a calendar for the current year.

How to display the day of the year in numbers

To display the day of the year in numbers (or Julian dates) pass the `-j` option. This displays days numbered from January 1.

```
cal -j
      September 2016
Mon Tue Wed Thu Fri Sat Sun
                245 246 247 248
249 250 251 252 253 254 255
256 257 258 259 260 261 262
263 264 265 266 267 268 269
270 271 272 273 274
```

h. `tty(teletype)`: Print the file name of the terminal connected to standard input. Print the file name of the terminal connected to standard input.

```
$tty <enter>
/dev/pts/1
$
```

File permissions [?](#)

Three file permissions:-

read: permitted to read the contents of file.

write: permitted to write to the file.

execute: permitted to execute the file as a program/script.

Three directory permissions:

read: permitted to read the contents of directory (view files and sub-directories in that directory).

write: permitted to write in to the directory. (create files and sub-directories in that directory)

execute: permitted to enter into that directory.

Chmod command?

Use the chmod command to set file permissions.

Relative Permissions?

Detecting File Permissions—

You can use the ls command with the -l option to show the file permissions set. For example, for apple.txt, I can do this:

```
$ ls -l apple.txt
```

```
-rwxr--r-- 1 december december    81 Feb 12 12:45 apple.txt
```

The sequence -rwxr--r-- tells the permissions set for the file apple.txt. The first - tells that apple.txt is a file. The next three letters, rwx, show that the owner has read, write, and execute permissions. Then the next three symbols, r--, show that the group permissions are read only. The final three symbols, r--, show that the world permissions are read only.

u Sets permissions for the owner of the file, e.g.: "u+w" allows the owner to write to the file

g Sets permissions for the group (to which owner belongs), e.g. "g-x" suppresses the execution of the file by the group

o Sets permissions for other users (that are not in group), e.g.: "o=r" allows others only to read the file

a Sets permissions for all (owner, group and others), e.g.: "a-w" disables write access to the file for everyone

= Assigns the permissions, e.g. "a=rw", sets read and write permissions and disables execution for all

- Removes certain thing[s] from the permissions, keeping all other (not involved) permissions. E.g. "a-x" disables execution of the file for everyone, this example doesn't touch read and write permissions.

+ Adds certain thing[s] to the permissions, keeping all other (not involved) permissions. E.g. "a+x" allows execution of the file for everyone, this example doesn't touch read and write permissions.

chmod Examples?

[1] Give read, write and execute to everybody (user, group, and others)

read, write and execute = $4 + 2 + 1 = 7$.

```
$ chmod 777 file.txt
```

(or)

```
$ chmod ugo+rwx file.txt
```

[2] Give execute privilege to user. Leave other privileges untouched

execute = 1. If you want to just add execute privilege to users and leave all other privileges as it is, do the following.

```
$ chmod u+x file.txt
```

```
[3] chmod u=rwx,g=rx,o=rx script.sh
```

```
[4] chmod u=rwx,go=rx script.sh
```

```
[5] chmod u+rwx,g+rx,g-w,o+rx,o-w script.sh
```

```
[6] chmod u+rwx,go+rx,go-w script.sh
```

```
[7] chmod o+r data
```

This grants other read permission to the file data. The command

```
[8] chmod +x data
```

grants everyone (user, group and other) execute permission, and the command

```
[9]chmod g+rwx data
```

gives category group read, write and execute permission.

```
$ ls -l socktest.pl  
-rwxr-xr-x 1 nick users 1874 Jan 19 10:23 socktest.pl*
```

```
$ chmod a-x socktest.pl
```

```
$ ls -l socktest.pl  
-rw-r--r-- 1 nick users 1874 Jan 19 10:23 socktest.pl
```

```
$ chmod g+w socktest.pl
```

```
$ ls -l socktest.pl  
-rw-rw-r-- 1 nick users 1874 Jan 19 10:23 socktest.pl
```

```
$ chmod ug+x socktest.pl
```

```
$ ls -l socktest.pl  
-rwxrwxr-- 1 nick users 1874 Jan 19 10:23 socktest.pl*
```

```
$ chmod ug-wx socktest.pl
```

```
$ ls -l socktest.pl  
-r--r--r-- 1 nick users 1874 Jan 19 10:23 socktest.pl
```

Absolute Permissions:-

The chmod command uses a three-digit code as an argument.

The three digits of the chmod code set permissions for these groups in this order:

- Owner (you)
- Group (a group of other users that you set up)
- World (anyone else browsing around on the file system)

Each digit of this code sets permissions for one of these groups as follows.

- Read is 4
- Write is 2
- Execute is 1

The sums of these numbers give combinations of these permissions:-

(00) 0 = no permissions whatsoever; this person cannot read, write, or execute the file

(001) 1 = execute only

(010) 2 = write only 3 = write and execute (1+2)

(100) 4 = read only

(101) 5 = read and execute (4+1)

(110) 6 = read and write (4+2)

(111) 7 = read and write and execute (4+2+1)

Chmod commands on file apple.txt (use wildcards to include more files)

Command	Purpose
chmod 700 apple.txt	Only you can read, write to, or execute apple.txt
chmod 777 apple.txt	Everybody can read, write to, or execute apple.txt
chmod 744 apple.txt	Only you can read, write to, or execute apple.txt Everybody can read apple.txt;
chmod 444 apple.txt	You can only read apple.txt, as everyone else.
