(21) grep → • Scan a document to find a pattern in file
• present the result in a format we want.

Syntax: grep <Search String>

eg:- Cat filename | grep apple

here apple is the search string.

Options of grep command

(1) -b → precede each line with its block number.

(2) -c → print the count of matched lines.

(3) -i → ignores uppercase and lowercase

(4) -l → list filenames but not matched lines.

(5) -h → Display the matched lines but do not display the filenames.

(6) -n → print line number without matched lines.

(7) -w → Match whole word.

(8) -v → select non matching lines i.e
Shows all the lines that do not match the searched string

eg:- Cat filename | grep -i a

(22) Sort → Sort command sort the lines of a file or files in alphabetic order.

Syntax → Sort filename

Options

(a) -c → checks whether files are already sorted.

(b) -d → ignores punctuation

(c) -r → reverses the order.

(d) -b → ignores spaces and tabs.

(e) -n → sorts numerically

eg:- Sort -r filename

(23) cu → cu command is used for communication over a modem or direct line with another linux system.

(24) login → login command invokes a login session to a linux system, which then authenticates the login to a system. System prompts us to enter userid and password.

(25) date → this command displays todays date.

Syntax → date

... write entire line and begin ... the

Regular Expressions

Regular expressions are a set of characters used to check patterns in strings.

- They are also called 'regexp' and 'regex'.
- It is important to learn regular expressions for writing scripts.

→ Basic Regular expressions

(a) . → replaces any character.

eg:- grep ".ello" input

This will output the words which has any ~~single~~ single character followed by ello.

→ In case we want to search for a word which has only 4 characters we can use grep -w "...." , where single dot represents any single character.

(2) ^ → matches start of string

eg:- cat filename | grep ^a

This will output the lines that starts with a character 'a'.

(3.) $ → matches end of string.

eg:- cat filename | grep t$

This will output the lines that ends with character t.

(4.) * → matches up zero or more times the preceding character.

grep 'ap*le'

eg:- BA* output → B, BA, BAA, BAAAA - - -

eg:- B.* output → B, BA, BAA - - -

eg: -1* output → matches zero or more 1.

(5.) \ → Represent Special characters. This is used to treat the following characters as an Ordinary character.

eg:- \$ is used to match the dollar sign character, rather than the end of a line.

(6.) [] → groups regular expressions

grep 'a[1-4]b', grep [l], grep [h-l]

eg: (ab)* output → ab, abbb - - -

(7.) ? → Matches up exactly one character.

eg:- 0? matches single zero or nothing.

substitute one (

es in linux

Pipes in Linux

The symbol '|' denotes a pipe.

- Use pipes to run two command consecutively.
- Helps in creating powerful comands.

Eg:- Cat filename | wc
Cat filename | less

- pipe is a symbol used to provide output of one command as input to another command.
- A pipe is a way to connect the output of one program to the input of another program without any temprary file.

Syntax command1 | command2

→ Filter → If a linux command accepts its input from the Standard input and produces its output on Standard output, then it is known as a filter.

Eg:- cat filename | grep -v a
ls -l | grep Feb

→ Commands to view hardware information

(1) lscpu → The lscpu command reports information about the cpu and processing units. It doesn't have any further options or functionality.

(2) lshw → List hardware.

A general purpose utility, that reports detailed ~~information~~ information about multiple different hardware units such as cpu, memory, disk, usb controllers, network adaptors etc.

(3) hwinfo → hardware information.

hwinfo is another general purpose hardware utility, that reports the detailed information about ~~hardware~~ multiple different hardware components and more than what lshw can report.

4. lspci → list PCI (Peripheral Component Interconnect)

The lspci command lists out all the pci buses and details about the devices connected to them.
The graphics card, network adapter, USB ports etc. all fall under this category.

5. lsusb → list of usb buses and device details.

This command shows the USB controllers and details about devices connected to them. By default brief information is displayed.

lsusb -v → this -v option will displays information about each USB port.

6. df → disk space of file systems.

Reports various partitions, their mount points and the used and available space on each.

→ Vi editor

The Vi (Visual editor) is one of the most popular text editor under Unix type system. Under Linux, there is a free version of Vi called Vim (Vi Improved) is an editor that is fully in text mode, which means that all actions are carried out with the help of text commands.

While using Vi, at any one time we are in one of three modes of operation. These modes are called – command mode, insert mode, and last line mode.

(a) Command Mode

→ Vi editor opens in this mode.
→ move the cursor and cut, copy, paste the text.
→ Save the changes to the file.
→ commands are case sensitive.
→ command mode take the user commands.

(b) Insert mode

→ this mode is for inserting text in the file.

→ press 'i' on the keyboard for insert mode.

→ In Insert mode, any key would be taken as an input.

→ press Esc key to save changes and return to command mode.

(c) last line mode → this mode give extended commands to vi

→ while typing these commands, they appear on the last line.

→ eg:- when we type ":" in command mode, we jump into last line mode and can use commands like "wq" (write the file and quit vi)

→ Starting vi editor

vi <filename>

we can create a new file or edit an existing file.

eg:- Steps
1. type vi filename
2. we are in command mode
   ~
   ~ ← denote unused lines.
   ~
3. To add content in the file, press 'i' key.
4. press 'esc' key to exit the insert mode and goes back to the command mode.

→ Basic commands in vi (we should be in the command mode in order to executes these commands)

a. i → insert at cursor (goes into insert mode)

b. a → write after cursor.

c. A → write at the end of line.

d. ESC → Terminate insert mode.

e. u → undo last change.

(f) V → undo all changes to the entire line.

(g) o → open a new line.

(h) dd → Delete line.

(i) 3dd → delete 3 lines.

(j) D → Delete contents of line after cursor.

(k) c → delete contents of line after the cursor and insert new text.

(l) dw → delete word

(m) 4dw → delete 4 words

(n) cw → change word

(o) x → delete character at cursor

(p) r → replace character

(q) R → Overwrite characters from cursor onwards.

(r) s → Substitute one character under cursor continue to insert.

(e) S → Substitute Entire line and begin to insert at begining of the line.

(f) ~ → change case of individual character.

→ Moving within a File

(a) k → move cursor up

(b) j → move cursor down

(c) h → move cursor left.

(d) l → move cursor right.

(e) 0 (zero) → move cursor to the Start of the current line.

(f) $ → move cursor to the end of current line.

→ Saving or closing the File.

(a) Shift + zz → Save the file and quit.

(b) :w → Save the file but keep it open.

(c) :q! → Quit without saving the file.

(d) :wq → Save the file and quit.

(e) :q → Vi editor will warn, if the file is modified and not let us quit.

→ Shell → is the outermost part of the O.S. A Shell in a Linux O.S takes input from us in the form of commands, processes them and then gives the output. Shell is a command interpreter. Shell is the interface through which the user works on the programs, commands and scripts. A Shell is accessed by a terminal which runs it. When we run the terminal, the shell issues a command prompt ($), where in we can type the input, which is then executed when we hit the enter key.

→ Types of Shell

1. The Bourne Shell ($) → also known as Sh.
   → The Sub categories are :-
   - Korn Shell → also known as KSh
   - Bourne Again Shell → also known as bash.

(2.) The C Shell → prompt is %

The Subcategories are

→ C shell → also known as csh

→ Tops C shell → also known as tcsh.

→ Shell scripting

- writing a series of commands for the shell to execute is called shell scripting.
- It can combine lengthy and repetitive commands into a simple and single script which can be stored and executed anytime.
- This reduces the effort of the end user.

→ Steps in creating a shell script

(1.) Create a file using a text editor like Vi.

(2.) Name the script file with the extension .sh.

(3.) Start the script with #!/bin/bash or #!/bin/sh.

(4.) Write some code.

(5.) Save the script file as filename.sh

executing the script type bash filename.sh

Bash shell

Bash is an sh-comptabile shell that incorporates useful features from the korn shell (ksh) and C shell (csh). Bash offers functional improvements over sh for both programing and interactive use.

→ **Adding shell comments**

Syntax: # comment.

Eg:- #! /bin/sh
pwd
present working directory
ls this is the comment.

~~Shell Variables~~

→ **Purpose of shell Script**

The shell scripts can be created in various Situations like:

1. whenever we perform repetitive task we use shell scripting.
2. creating our own power tools/utilities.
3. creating simple applications.
4. customizing administrative tasks.

and so on.

→ Shell Variables

Variable is a named storage location that can take different values, but only one at a time.

→ In Linux (Shell), there are two types of variables :-

(1) System Variables → created and maintained by Linux itself. These types of variables are defined in CAPITAL LETTERS.

(2) User defined variables (UDV) → created and maintained by user. These types of variables are defined in Lower letters.

→ Various System Variables available in Linux

BASH, HOME, PWD, SHELL, USERNAME, LOGNAME etc

→ To see System Variables → $echo $USERNAME.

→ Defining the User Defined Variables (UDV)

Syntax :

Variable name = value

'Value' is assigned to given 'variable name' and Value must be on right side of =.

eg:- $n=10

Scanned by CamScanner

~~1st Program~~

Example of shell script

```
#!/bin/bash
STR="Hello World"
echo $STR
```

Line 2 creates a variable called STR and assigns the string "Hello World" to it. Then the value of this variable is retrieved by putting the '$' in the beginning.

→ To print or Access Value of UDV

Syntax: $ Variablename

eg:-

```
$ n=10
$ echo $n
```

→ Performing Arithmetic operations

expr is used to perform arithmetic operations.

Syntax:

expr op1 mathoperator op2

~~Example:~~

~~$ expr 1+3~~

~~expr is also~~ expr is also end with i.e back quote.

~~for expr 3+6~~

example

```
a=10
b=20
val=`expr $a+$b`
echo $val
echo "a+b=$val"
```

$echo $a

→ The Read Statement

It is used to get input (data from user) from keyboard and store (data) to variable.

Syntax: read variable1, variable2 .... variableN

Example

```
#!/bin/bash
echo -n "Enter some text> "
read text
echo "You entered text: $text"
```

Output

```
Enter some text> this is some text
You entered text: this is some text
```

→ The Decision Making in shell Scripts

① if condition

Syntax:

```
if condition
then
    command1 if condition is true or if exit status
    of condition is 0 (zero)
    ---
    ---
fi
```

Version control structures in shell scripts

1. If statement → Syntax

```
if <condition>
 then
 # series of code
fi
```

2. If else statement → Syntax

```
if <condition>
 then
 # series of code
else
 # series of code if condition is not satisfied
fi
```

3. Multiple if → syntax

```
If <condition1>
 then
 # series of code for condition 1
elif <condition 2>
 then
  # series of code for condition 2
else
 # series of ~~condition~~ code if condition 2
     is not satisfied
 fi
```

→ File based condition

Operator	Description
-a file	Returns true if file exists
-b file	Returns true if file exists and is a block special file.
-c file	Returns true if file exists and is a character special file.
-d file	Returns true if file exists and is a directory.
-e file	Returns true if file exists.
-r file	Returns true if file exists and is readable.
-w file	Returns true if file exists and is writable.
-x file	Returns true if file exists and is executable.

example-1

```
#!/bin/bash
cd
ls
if [ -e Sample.sh ]
  then
    echo "file exists".
```

```
    echo "file does not exists"
fi
```

→ Example-2

```
#!/bin/bash
echo "enter two numbers"
read a, b
echo "numbers entered are $a, $b"
if [ $a -eq $b ]
  then
  echo "numbs are equal"
  else
  echo "numbs are not equal"
fi
```

Shell Script

Q-1 Write a shell script that will add two numbers.

```
echo "enter two numbs"
read a, b
echoSum = 'expr $a + $b'
echo "the sum is = $Sum"
```

Q-2 write a shell script to print the as 5, 4, 3, 2, 1 using while loop.

```
i=5
while [$i -eq 0]
do
echo "$i"
  ((i--))
done
```

OR

```
i=5
while test $i !=0
  do
  echo "$i"
  i=`expr $i - i`
done
```

Q-3 Reverse a no.

```
reverse=0
echo "enter number to reverse"
read n
echo "$n"
while [ $n -neq 0]
  do
  reverse=`expr $reverse * 10`
  reverse=`expr $reverse + $n % 10`
done n  = `expr $n / 10      echo "$reverse".
```

→ Shell loops

(1) while loop

Syntax

```
while [condition]
do
   command 1
   command 2
   command 3
   -
   -
   =
done.
```

loop is executed as long as given condition is true.

example

```
c=1
while [$c -le 5]
  do
  echo " welcome $c time".
    ((c++))
done
```

loop

for {variable name} in {list}

do

Statements to be executed for everyword.

done

Example

```
for var in 0 1 2 3 4 5 6 7 8 9
    do
    echo $var
    done.
```

Output

```
0
1
2
3
4
5
6
7
8
9
```

→ & The 'case' Statement

Syntax:

```
case word in
pattern1)
    Statements to be executed if pattern1 matches
    ;;
```

pattern 2)
Statements to be executed if pattern 2 matches
;;
pattern 3)
Statements to be executed if pattern 3 matches
;;
esac

→ if no matches are found, the case statement exits without performing any action.
;; indicates that program flow jump to the entire case statement. This is similar to break in C language.

Example

```
FRUIT="Kiwi"
case "$FRUIT" in
"apple") echo "Apple is tasty".
;;
"banana") echo "I like banana";
;;
"kiwi") echo "Newzealand is famous for kiwi".
;;
esac.
```

For mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

NOTE: == is equal, != is not equal.

For string comparisons use

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined

Shell also test for file and directory types

Test	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

directories may become inaccessible).

2.2 THE POWERS OF 'ROOT' IN LINUX

Root is the default name for system administrator in a Linux system. Root is a super user who can do anything and everything within the operating system. As a result, root login should be used with special care. While working with a root login, we can end up doing a lot of harm to our system as well as the data, accidentally.

Power of the Root Account

Root can perform any of a multitude of functions on a Linux system. Here are just a few of the functions that only root can do:

1. To add new users to the system and administer the user data.
2. Adjust system resources and quotas.
3. Manage all configuration files.
4. Create directories and device files in any location on the machine, including those that root does not specifically own.
5. Set the system clock.
6. To install system-wide software.
7. Configure I/O devices like – a scanner or a TV tuner card, for example.
8. Configure network interfaces.
9. Configure system services like – a web or FTP server.
10. Shut down the system cleanly.

Process in Linux

Processes in Linux

- Anything that is running in Linux is a process.
- A process is defined as an "instance" of an executing program. Anytime a user executes a command at the command line, a process is created.
- eg:- (i) Shell that is running and taking our commands is a process,
- (ii) the commands that we type on terminal is a process etc.

- Processes are much like humans:-
  - → they born, they die
  - → they have parent and children

→ **Shell process**

- Shell is a process started by Linux kernel as soon as we login to our system.
- the shell creates or gives birth to all the other user command processes
- A shell can also give birth to another shell process.

- Giving birth to a process or creating a process is also called Spawning a process.

e.g:- when we open a terminal, we see a command prompt ($), this is the job of the shell process, when we type any command say 'date' and press enter, the shell process create a process called 'date' ie the shell process gives a birth to a date process, we can say that 'shell process is the parent of the date process and 'date' process is the child of the shell process, once the date process displays the date and time, date process will die.

→ Process attributes

- we are identified by attributes like our name, date of birth etc.
- Similarly processes have attributes:-
  (a) PID : Process ID

PPID: Parent Process ID

Start time etc.

These attributes are maintained by the Kernel in a process Table.

- PID : Each Process is uniquely identified by an Unique integer called PID. PID is allotted by the Kernel when the process is born.
- PPID : The PID of the parent of that process.

eg:- echo $$
3497 → This is PID of the current shell.

→ Process Types

(1.) Interactive Processes → They are initialized and controlled through a terminal session. They are not started automatically. These processes can run in the foreground and the background as well.

(2.) Automatic Processes → Automatic or batch processes are not connected to a terminal. These Processes are started by the system often during system start up or user login. eg:- bash

→ Process creation Phases

There are three distinct phases in the life of process and uses three important System Calls — fork, exec and wait.

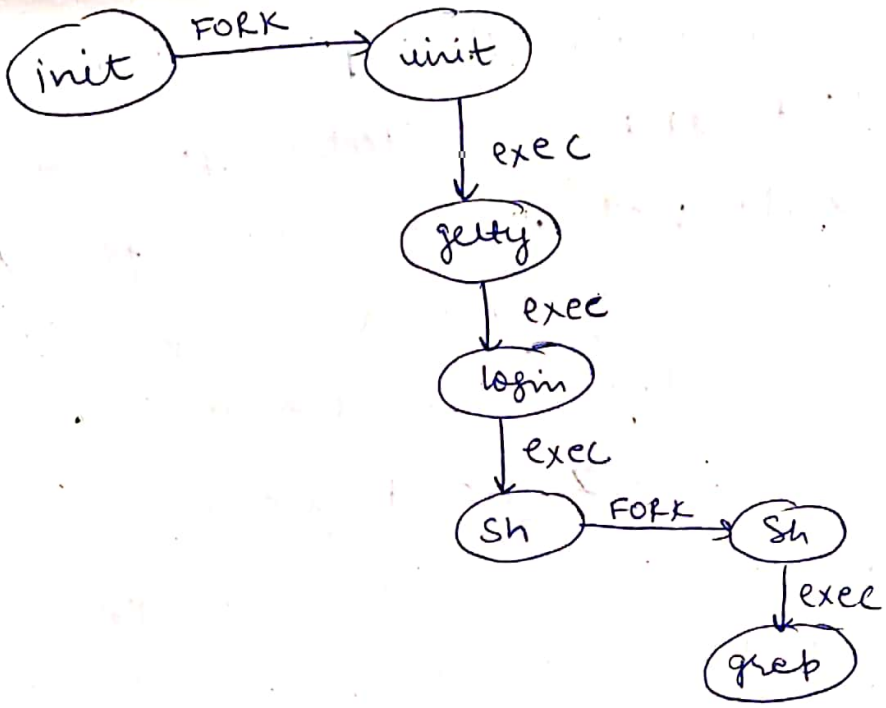
(1) fork → Forking creates a process by creating a copy of the existing process. The new process has a different PID, and the process that created it becomes its parent. Otherwise the parent and the child have the same process ~~image~~ id.

(2) exec → After forking process, the address space of the child process is overwritten with the new process data. This is done through an exec call to the system. This mechanism is called exec, and the child process is said to exec a new program.

(3) Wait → While the child is executing a new program, the parent normally waits for the child to die. It then picks up the exit status of the child before it does something else.

The Shell Creation Method

When the system moves to multiuser mode, init forks and execs a getty for every active communication port. Each one of these gettys prints the login prompt on the respective terminal and then goes off to sleep. When a user attempts to log in, getty wakes up and fork-execs the login program to verify the login name and the password entered. On successful login, login forks-execs the process representing the login shell.



→ Zombie Process

It is a process which has got an entry in the process table, but has exited from the system.

Parent process collects the exit status from the child and it knows that the child is exited, it will inform the operating system that this particular child with PID

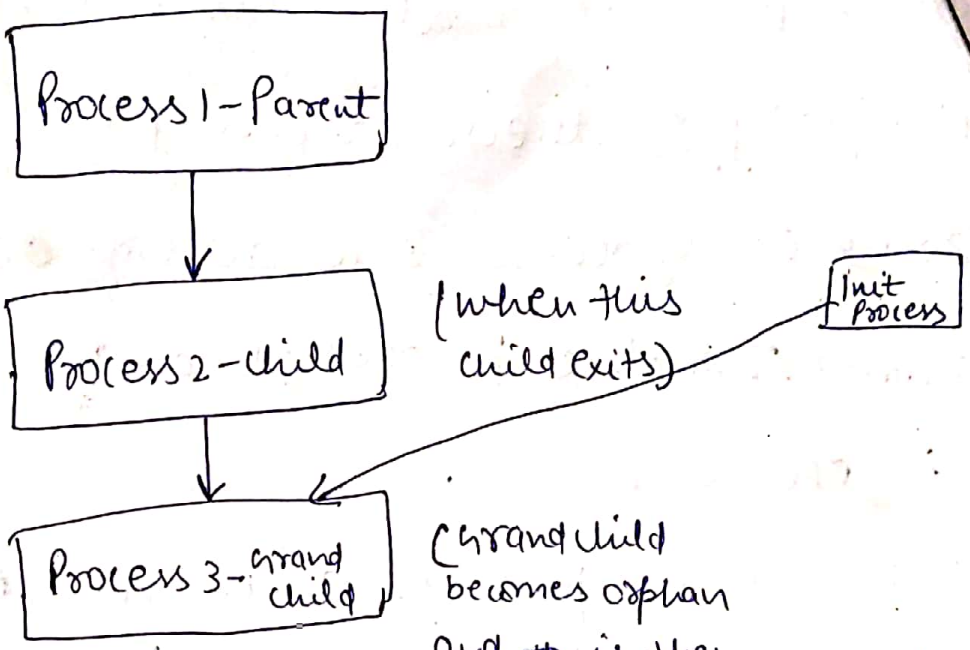
Say 3473 has exited, so remove the process table entry in the process table. But, if the parent doesn't collect the exit status from the child, it will not be able to inform the operating to remove the process table entry, such a kind of process becomes a Zombie Process.

The Problem with the Zombie Process is that it keeps on occupying the space in the process table. And after some time we can't create anymore processes & resulting in Memory leakage.

Q3. Zombie processes are the leftover bits of dead processes that have not been cleaned up properly by their parent process. In short, zombie processes are already dead, hence the name. Zombie processes are not really processes at all.

When a subprocess exits, its parent is supposed to use the 'wait' system call and collect the process's exit information. The sub process exits as a zombie process until this happens, which is usually immediately. However, if the parent process is not programmed properly or has a bug and never calls 'wait', the zombie process remains, eternally waiting for its information to be collected by its parent.

→ Avoid a zombie process by forking twice



Init process always collects the exit status of thus child process, avoiding the zombie process.

→ ps command

'ps' stands for process status. This command is used to provide information about the currently running processes, including their PID's (process identification numbers).

Syntax ps [options]

options of ps command

-a → List all processes in system except processes not attached to terminals.

(2) -e → List all processes in system.

(3) -f → Lists a full listing

(4) -j → Print process group ID and session ID.

→ Killing processes

'Kill' command is used to kill a process. In order to kill a process, we need to know the process ID (PID) of a process.

Syntax → Kill PID or Kill [Signal or Option] PID

Kill command is used to send signals to running processes in order to request the termination of the process.
Signals are mechanisms to communicate with processes. Linux supports 31 signals. Each signal is identified by a number from 1 to 31.

→ Some of them are -:

(1) SIGHUP - 1 → Stands for Signal hangup. The SIGHUP signal is commonly used to tell a process to shutdown and restart, this signal can be caught and ignored by a process.

Syntax $ Kill - 1 <pid> or Kill -SIGHUP<PID>

(2) SIGINT - 2 → This signal is commonly used when a user presses Ctrl+C on the keyboard. This is the interrupt signal. This will terminate the process, it can be caught or ignored.

(3) SIGKILL - 9 → This signal cannot be ignored by a process and the termination is handled outside of the process itself.

(4) SIGTERM - 15 → This signal is the default signal sent when invoking the kill command. This tells the process to shutdown and is generally accepted as the signal to use when shutting down cleanly. This is the termination signal.

Login Process of Linux

After the system boots, the user will see a login prompt similar to:

machinname login:

This prompt is being generated by a program, getty, which is regenerated by the init process every time a user ends a session on the console. The getty program will call login and login, if successful will call the user shell. The steps of the process are:

1. The init process spawns the getty process
2. The getty process invokes the login process when the user enters their name and passes the usr name to login.
3. The login process prompts the usr for a password, checks it, then if there is success, the usrs shell is started. On failure the program displays an error message, ends and then init will respawn getty.

(4.) The user will run their session and eventually logout. On logout, the shell program exits and we return to step 1.

→ Linux - Shutdown Process

(1.) The easiest way to shut down a Linux system is to enter the desired run-level in the command line. For eg:- the command init Linux 0 will shutdown the system. However, this command doesn't inform users that the system is about to be halted, and they will be unable to save any files they have open.

(2.) Another way to shutdown the system is to use the shutdown command. The shutdown command broadcasts a message to all users that the run level is about to change. The shutdown command tells init to change the current run level by using either the halt or boot command.

example → shutdown -r now → Shutdown and reboots now.

shutdown -h 5 → shutdown and halts the system in 5 minutes

shutdown -k 5 → Doesnot shutdown but sends broadcast message to users.