LINUX ADMINISTRATION, SHELL SCRIPTING, AWK PROGRAMMING

PREPARED BY ARUN NATARAJAN

==========

Level 1:

- Introduction
- Filesystem Hierarchy
- Command Promptgrep
- Command Syntax, Basic Commands, Help Commands
- File Types
- File and Folder Management
- Vi/Vim Editor
- File Descriptors, Tee
- Word Count
- History
- Filters (grep,cut,sort,tr)
- Process & Memory
- Backup & Restore (tar,zcat,gzip)shell
- Scheduler (at, crontab)
- Find
- Alias

Level 2:

- Shell scripting basics
- Variables
- Quotes, Sleep
- Control Statements
- Regular Expressions
- SED
- AWK
- Interactive Scripts
- Script Debugging

  Grep

Installation of Linux OS on Windows machine:

First we need to install vmware on windows and then linux on vmware.

i.e Windows -> VMware -> Linux

Installation of VMware tool 10.0 on Windows 7/8/10

Pre requisite:

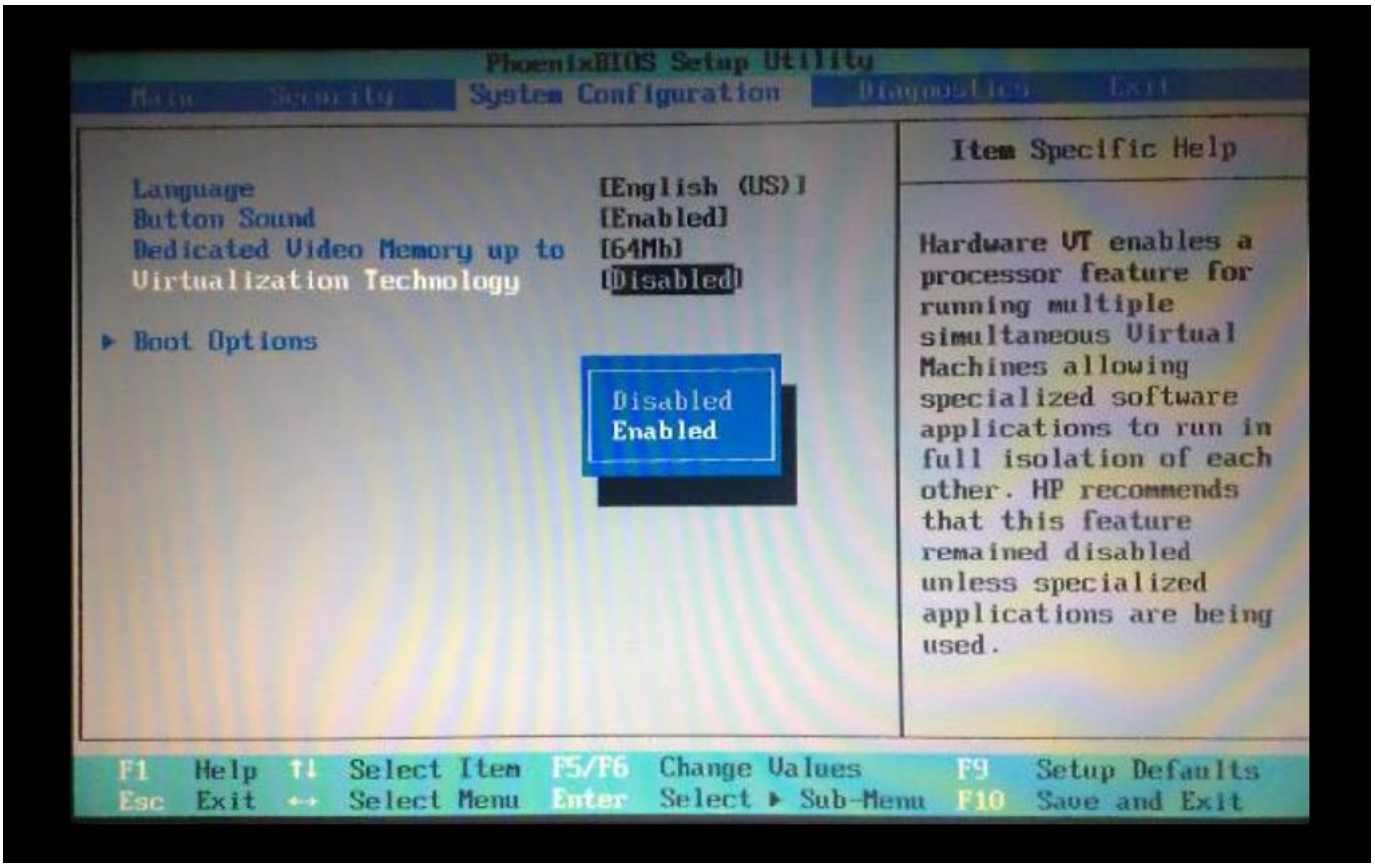
Your windows machine should statisfy below criteria:

Minimum of 4GB RAM and i3 processor……. more than that will be recommended.

Virtualization should be in "**Enabled**" status on your desktop/laptop BIOS settings, as per below image.

Note: As per system brand(HP, IBM, VAIO…) BIOS settings will differ, you need to check for the same as per your machine to enable "Virtualization" technology.

-usermod



Step 1

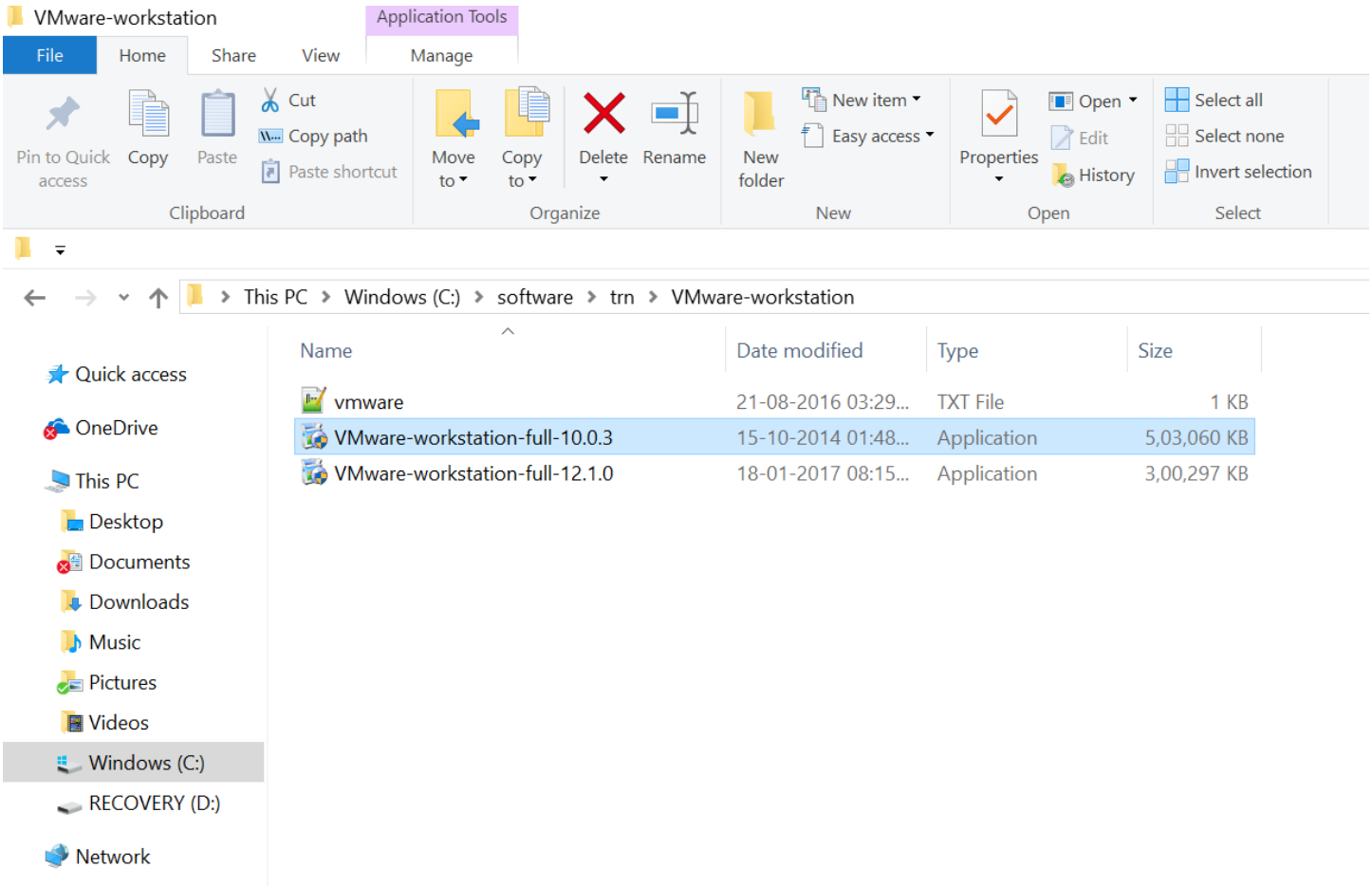
Now, we are going to install VMware 10.0 on Windows 10 machine.

You can download the setup file from my G-Drive or from google.

Software:
https://drive.google.com/open?id=0B1usxOmTVpWUfkVyVklmazhDemtqQ25VbUJWdG80U0NBU3FHREQyMVRCeVVubFJ5eVNuS1E

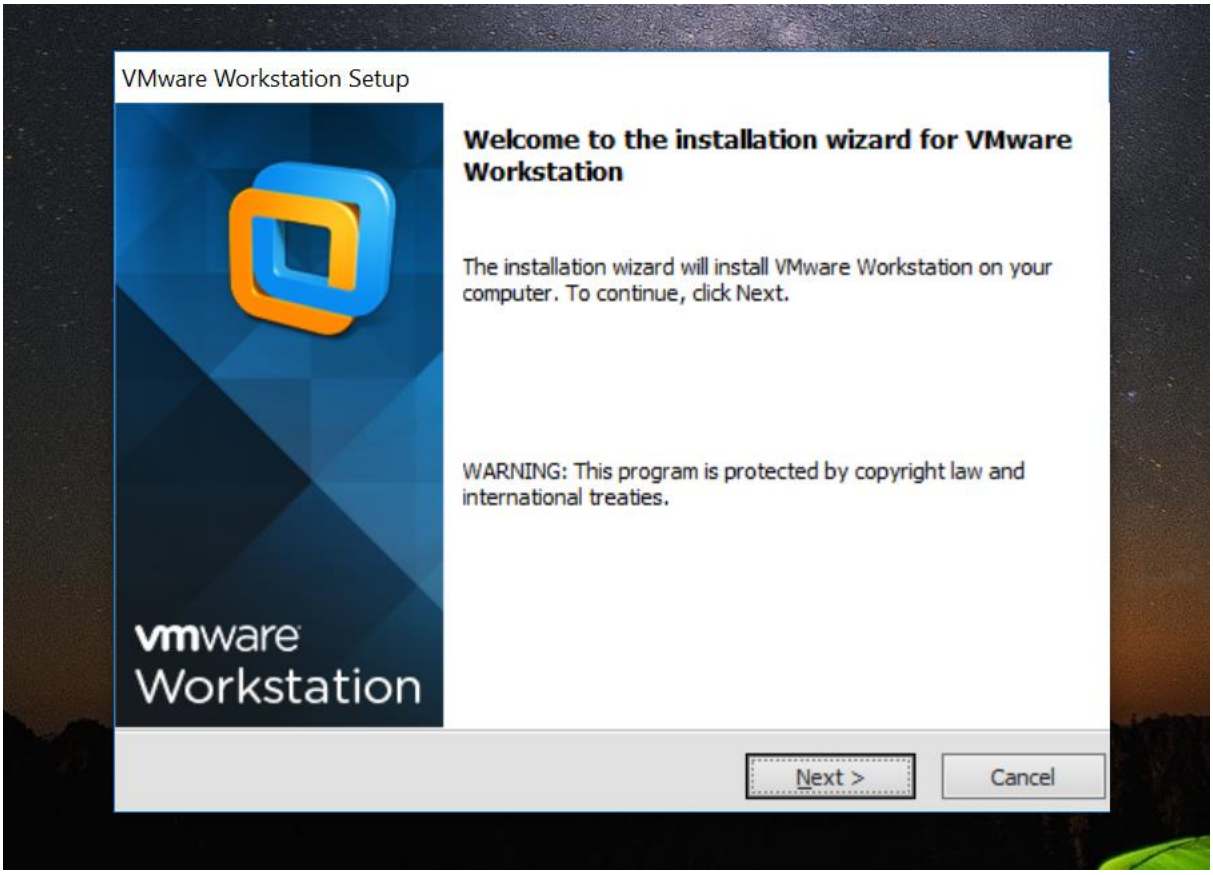
I ll be using below VMware setup file for installation.

Now, double click on VMware 10.0 setup file to initialize the installation.



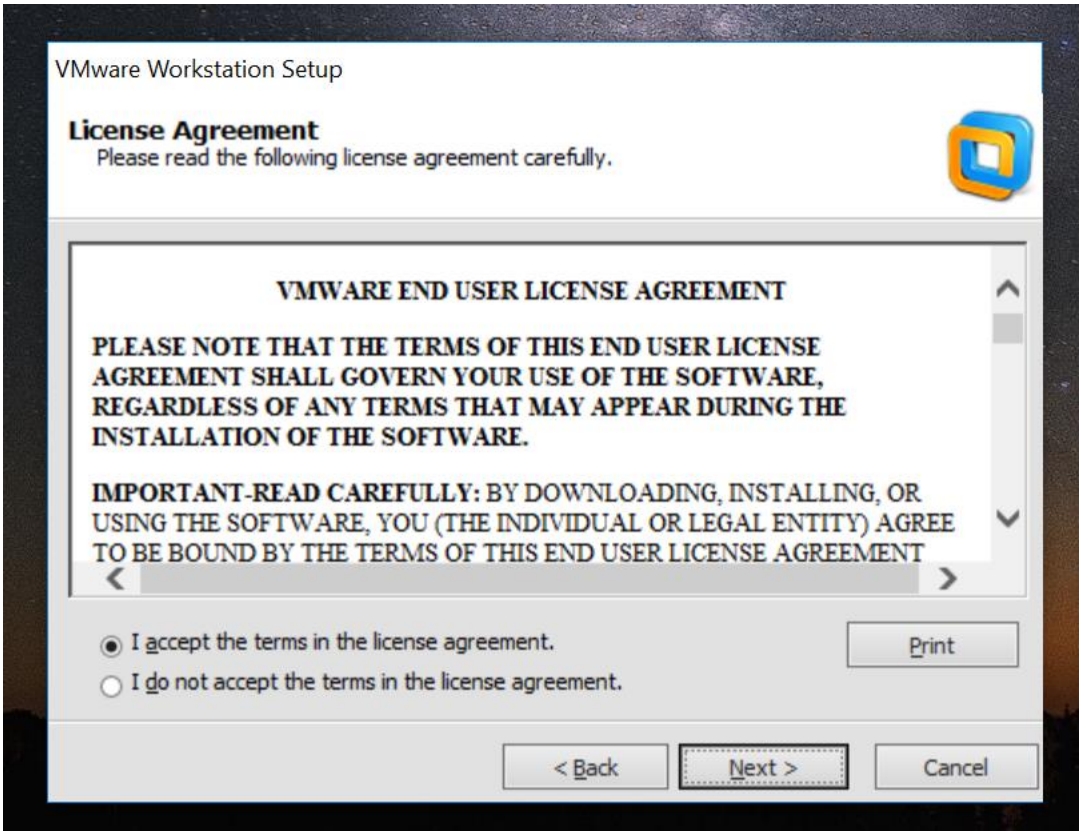
Step 2 :

Once you got VMware setup screen, click on NEXT.



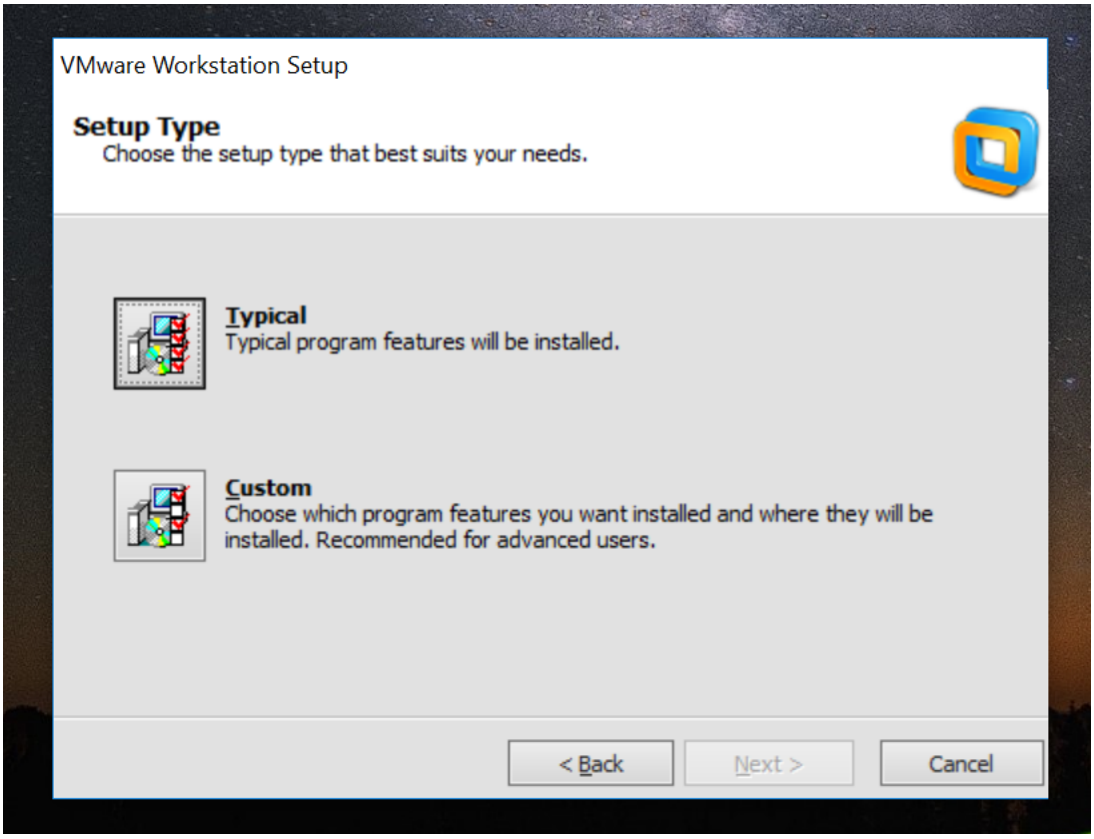
Step 3:

Agree with the license agreement and click on NEXT.



Step 4:

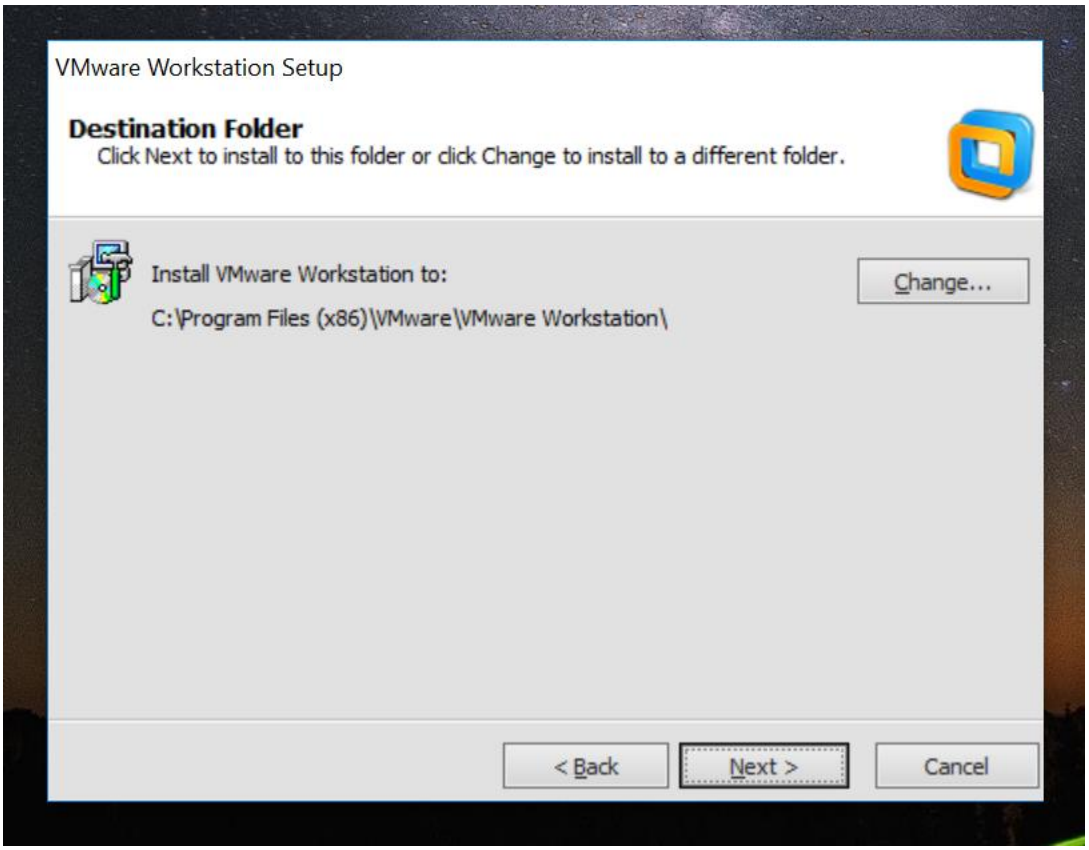
Select Typical installation and click on NEXT



Step 5 :

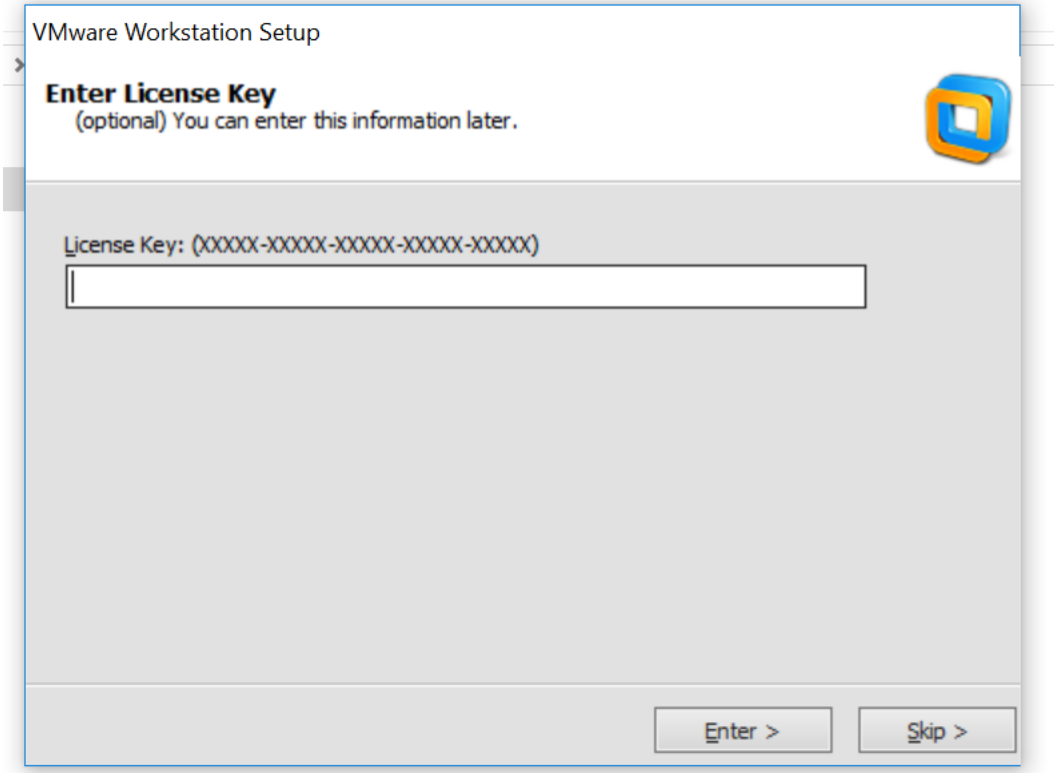
Choose your installation folder, by default it will be installed on C: drive.

You can change the installation where ever you have the space on your windows machine.



Step 6:

Enter the key information as per below from the provided key file from my G-Drive.



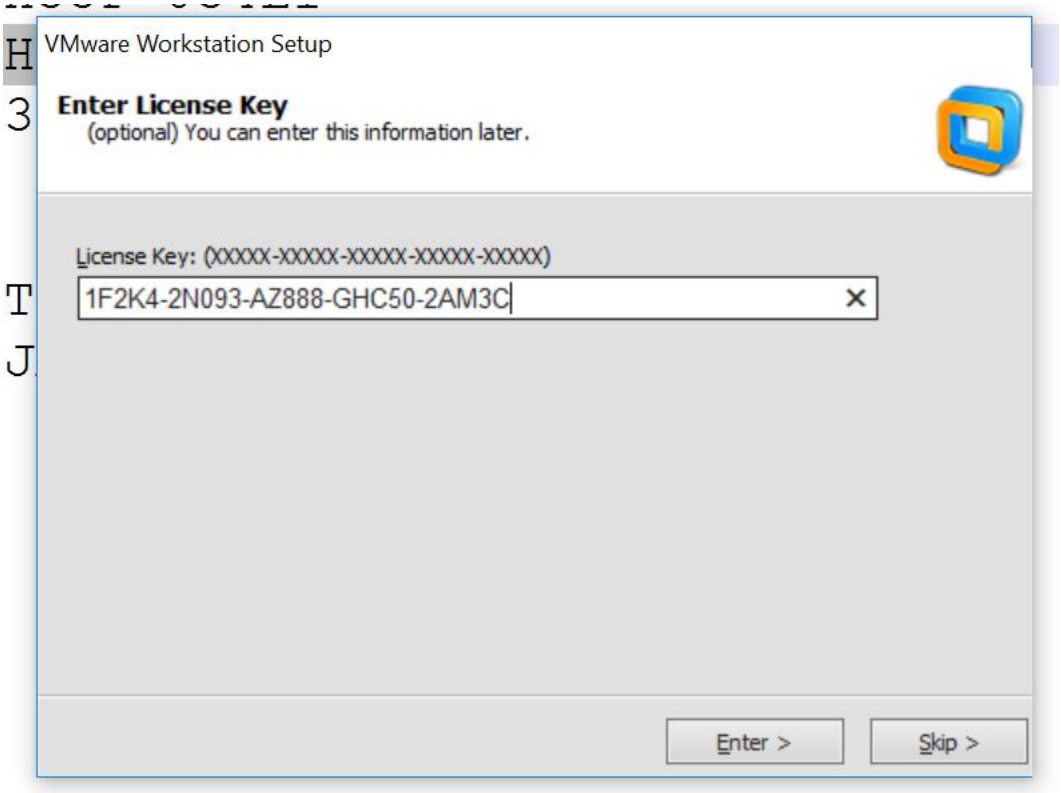
C:\software\trn\VMware-workstation\vmware.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

kdump.txt | to do.txt | sample shell scripts.sh | new 1 | trn details.txt | new 2 | vmware.txt

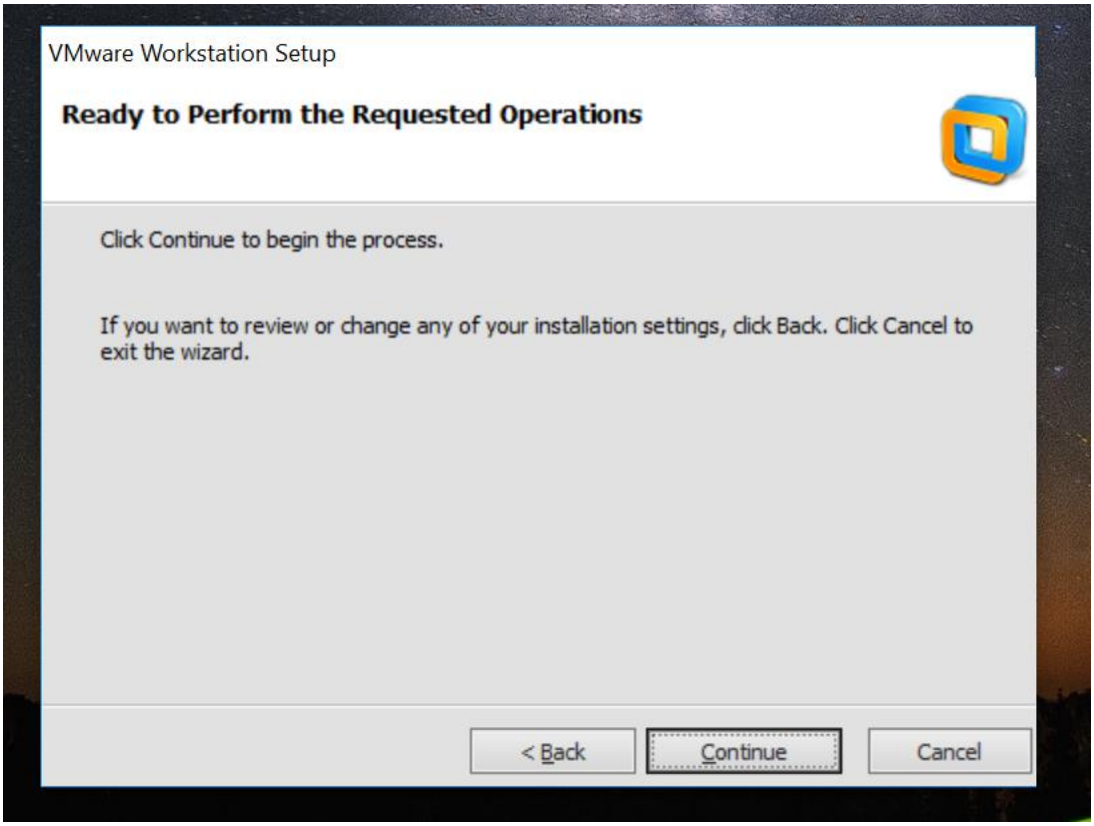
```
10.0
NA0RU-2YLE0-6ZMX9-HH35P-034L1
1F2K4-2N093-AZ888-GHC50-2AM3C
4F4V2-A038J-2ZF90-532QK-834LG

12.0
5A02H-AU243-TZJ49-GTC7K-3C61N
1J0KL-CZK41-H8JF3-0JAUK-CWC27
```



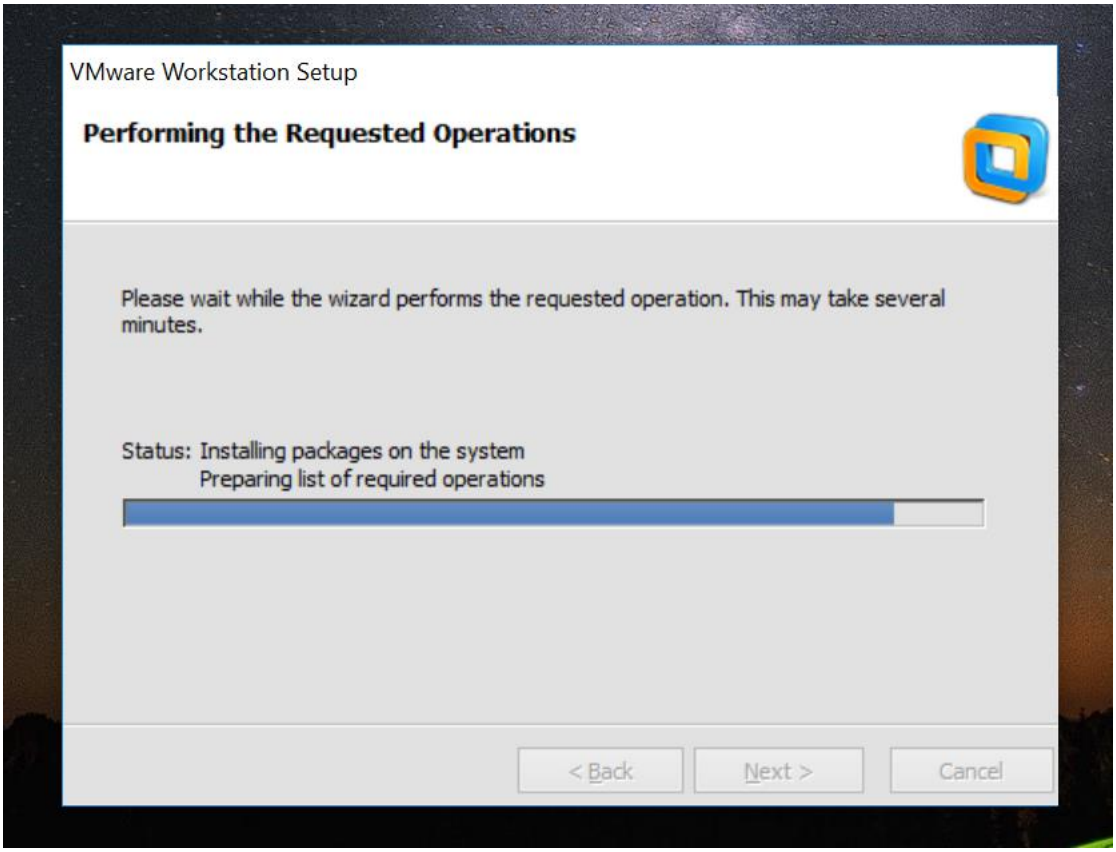
Step 7:

Keep proceeding with NEXT button for few more screens and at last click on Continue



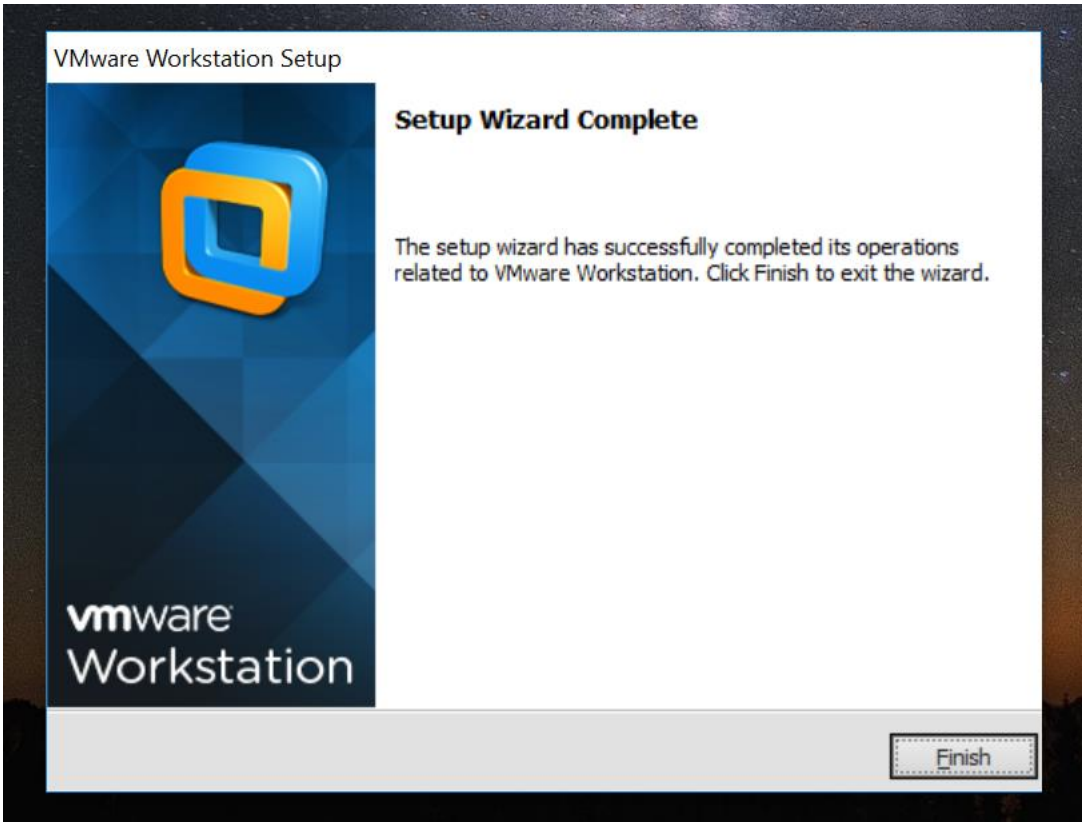
Step 8:

After clicking on Continue, your VMWare installation will start.



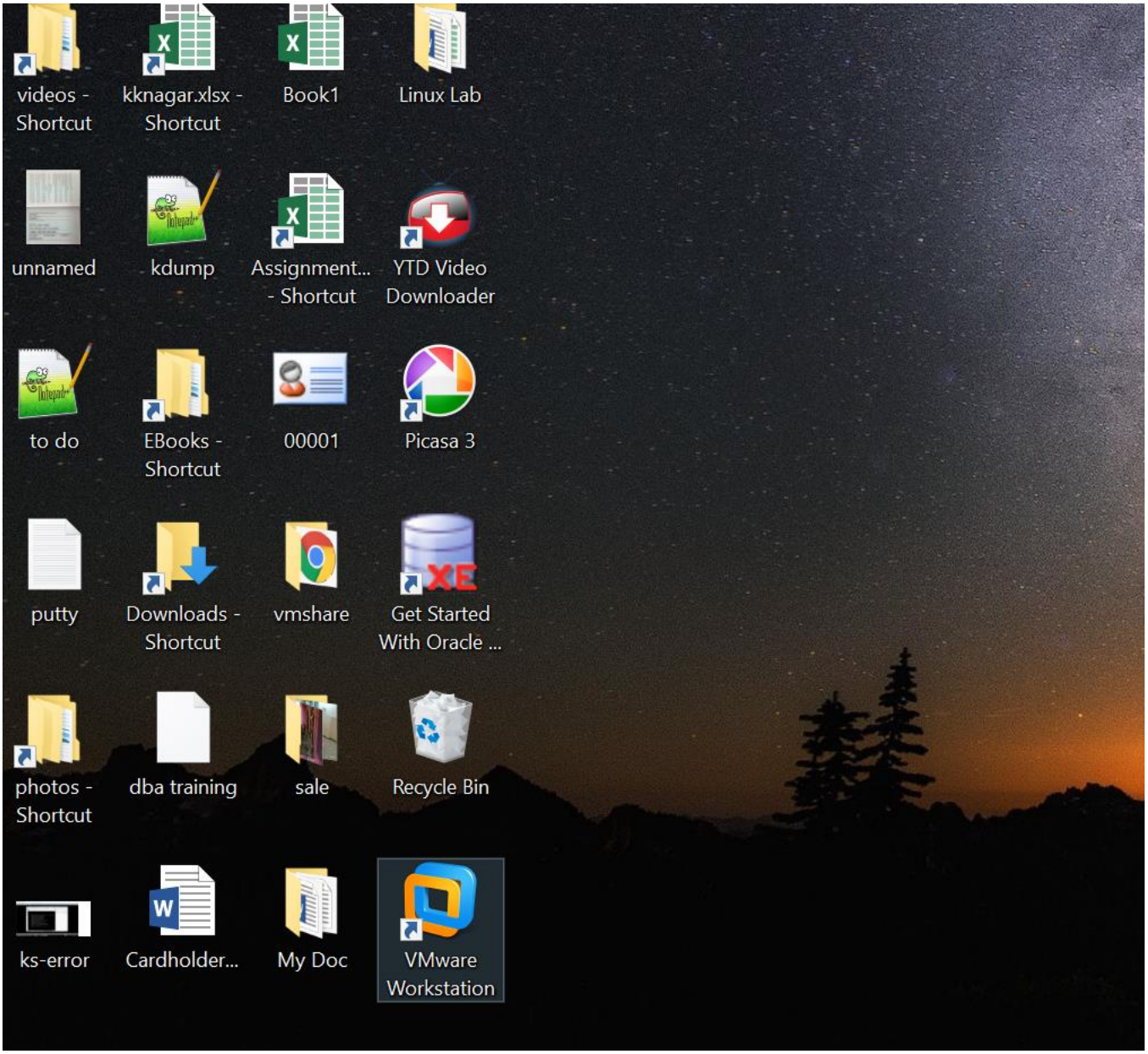
Step 9:

After VMware installation is done, click on *Finish.* Now you can able to see VMware icon on your desktop



Step 10:

Please verify the same below screenshot.



Now, we are done with installation VMware tool on windows machine.

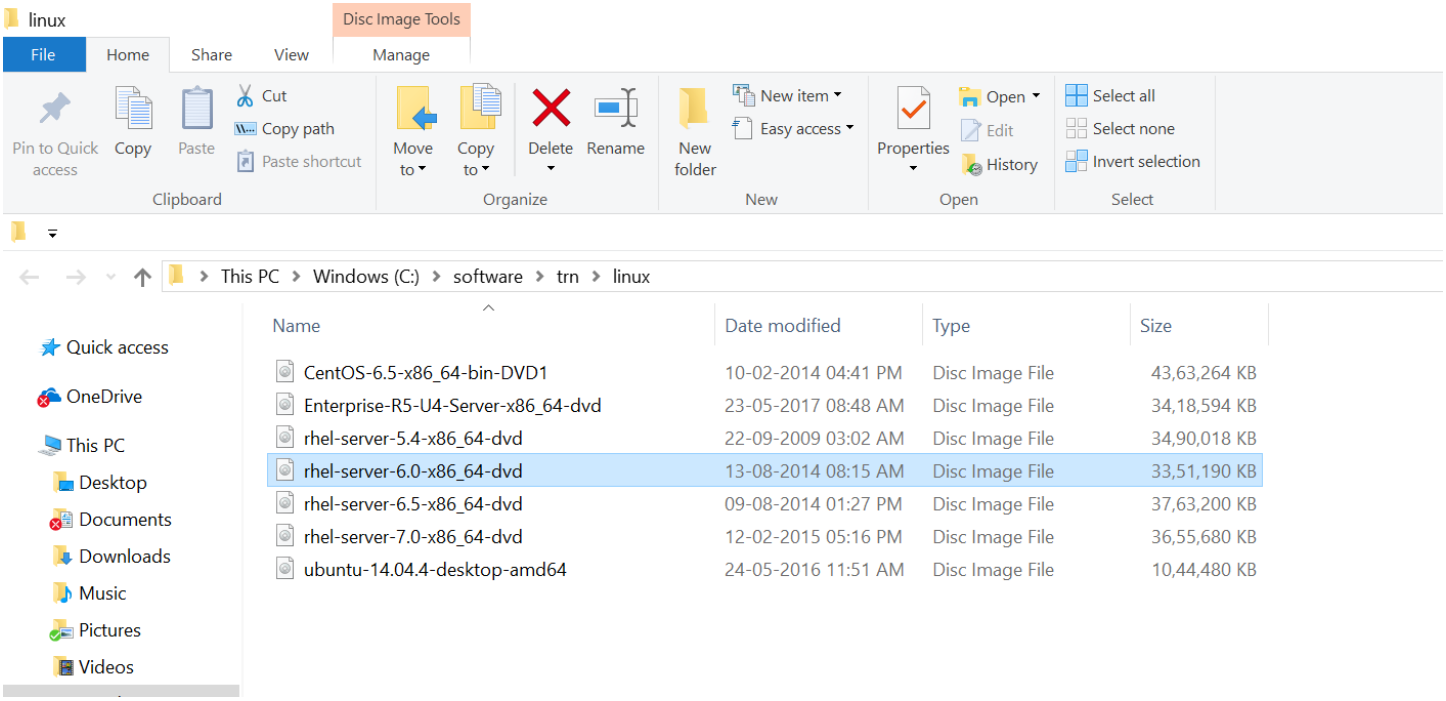
Installation of RHEL 6.0 on VMware tool.

Make sure you have RHEL 6.0 iso file ready with you before starting the installation.

You can download it from my G-Drive or Google.

Software:
https://drive.google.com/open?id=0B1usxOmTVpWUfkVyVklmazhDemtqQ25VbUJWdG80U0NBU3FHREQyMVRCeVVubFJ5eVNuS1E

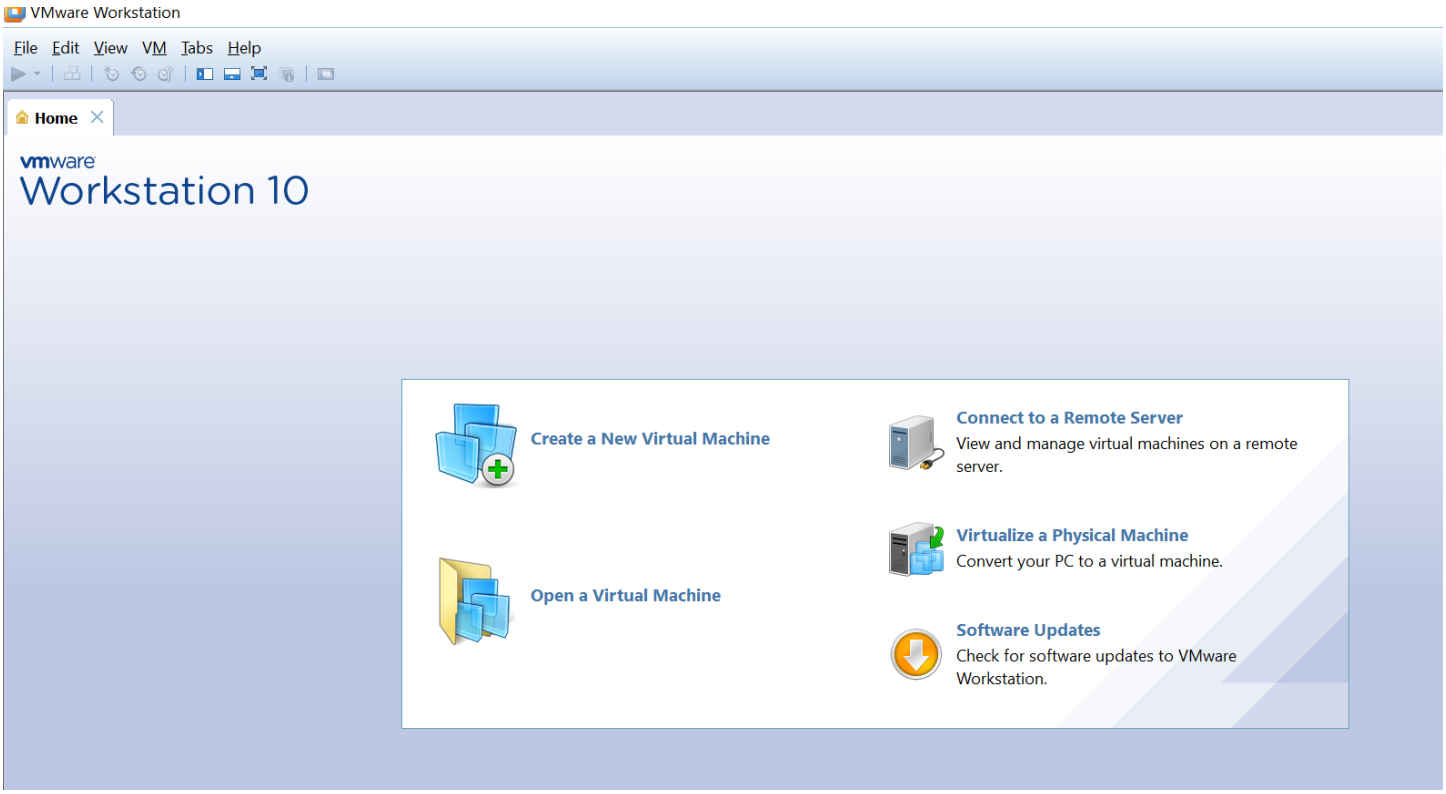
Below highlighted file will be used for installation now.



Step 1:

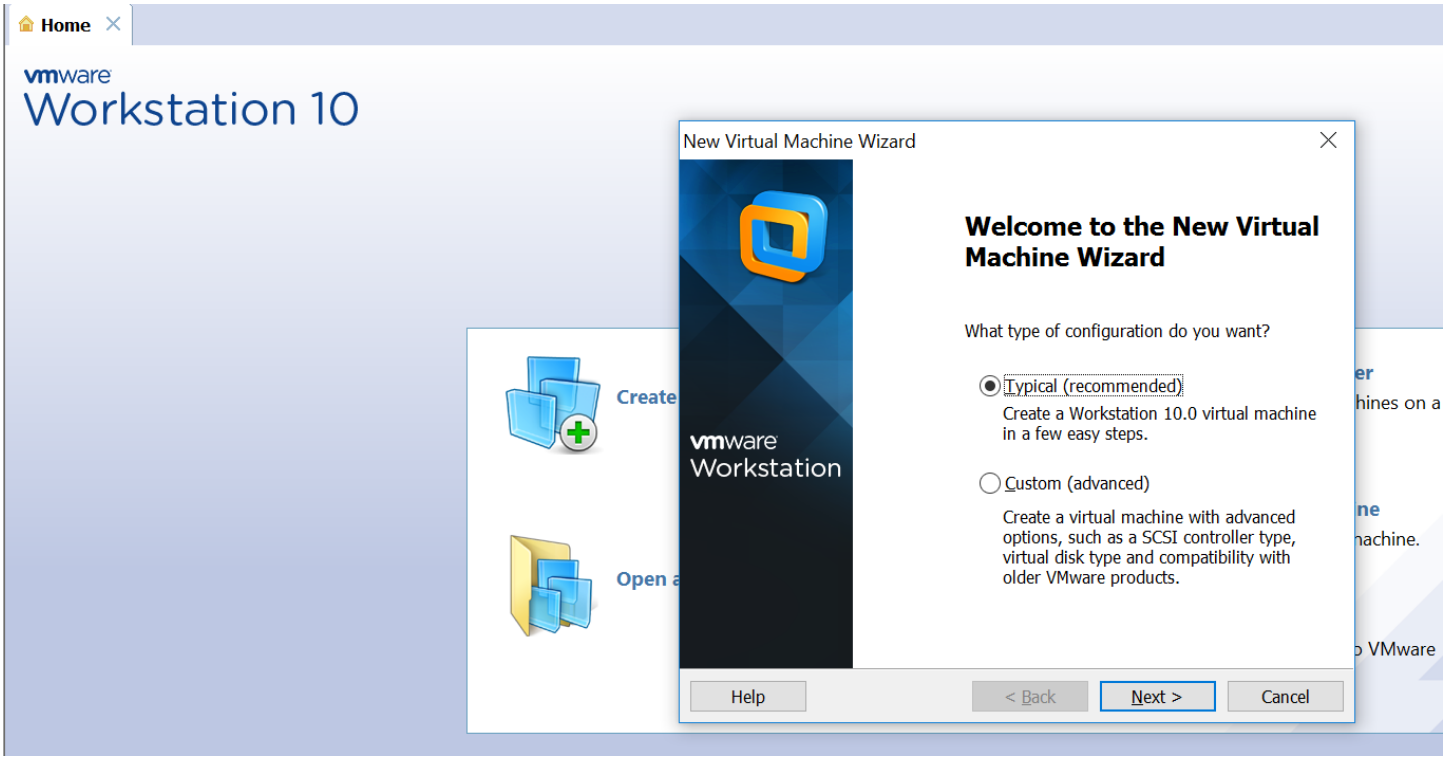
Click on VMware icon available on your Desktop, so that you will be getting the below screen.

Once VMware application is opened, click “Create a New Virtual Machine” .



Step 2:

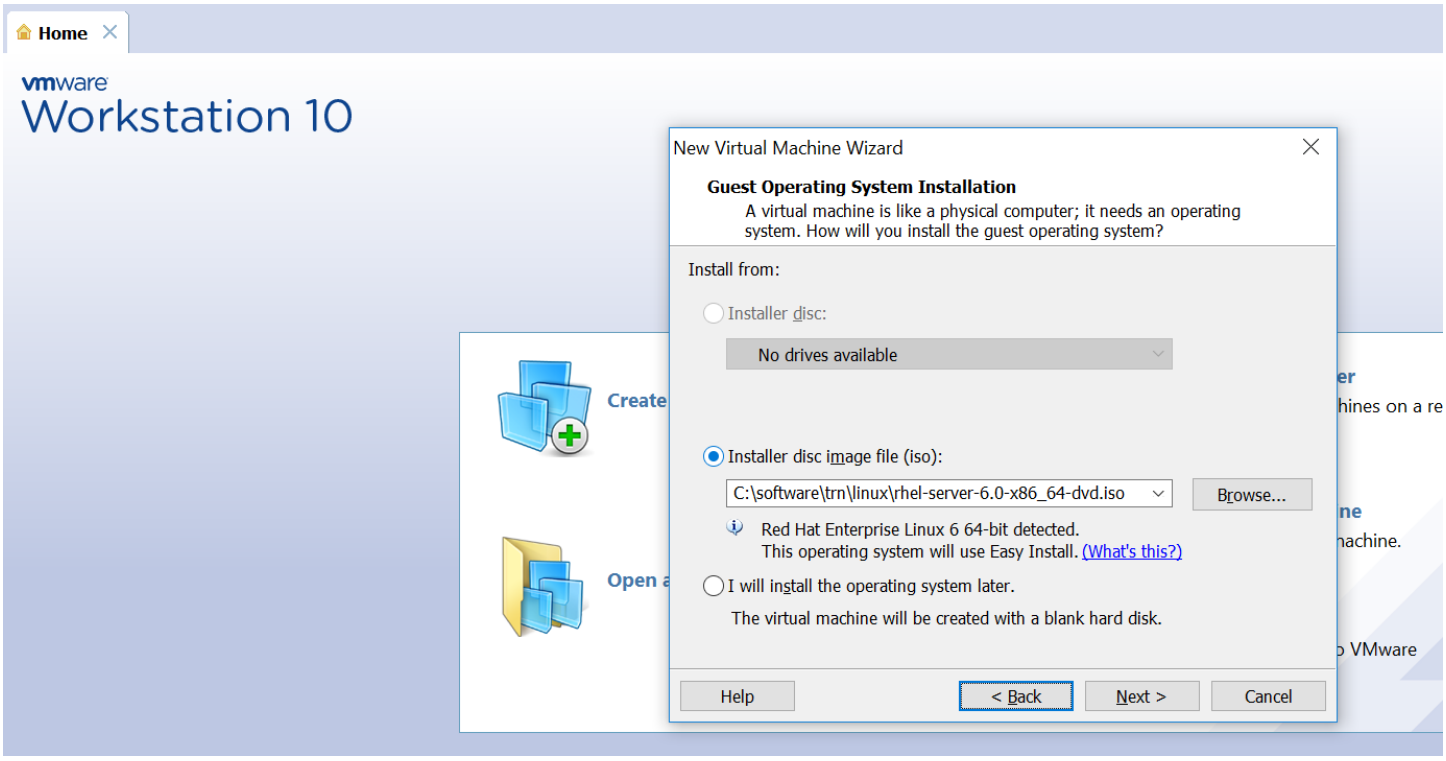
Select Typical installation on click on *NEXT*



Step 3:

Select ISO option and click browse to map your RHEL 6.0 iso file from your windows machine and click on *NEXT*

I have the RHEL 6.0 iso file at *c:\software\trn\linux\rhel-server-6.0-x86_64-dvd.iso*



Step 4:

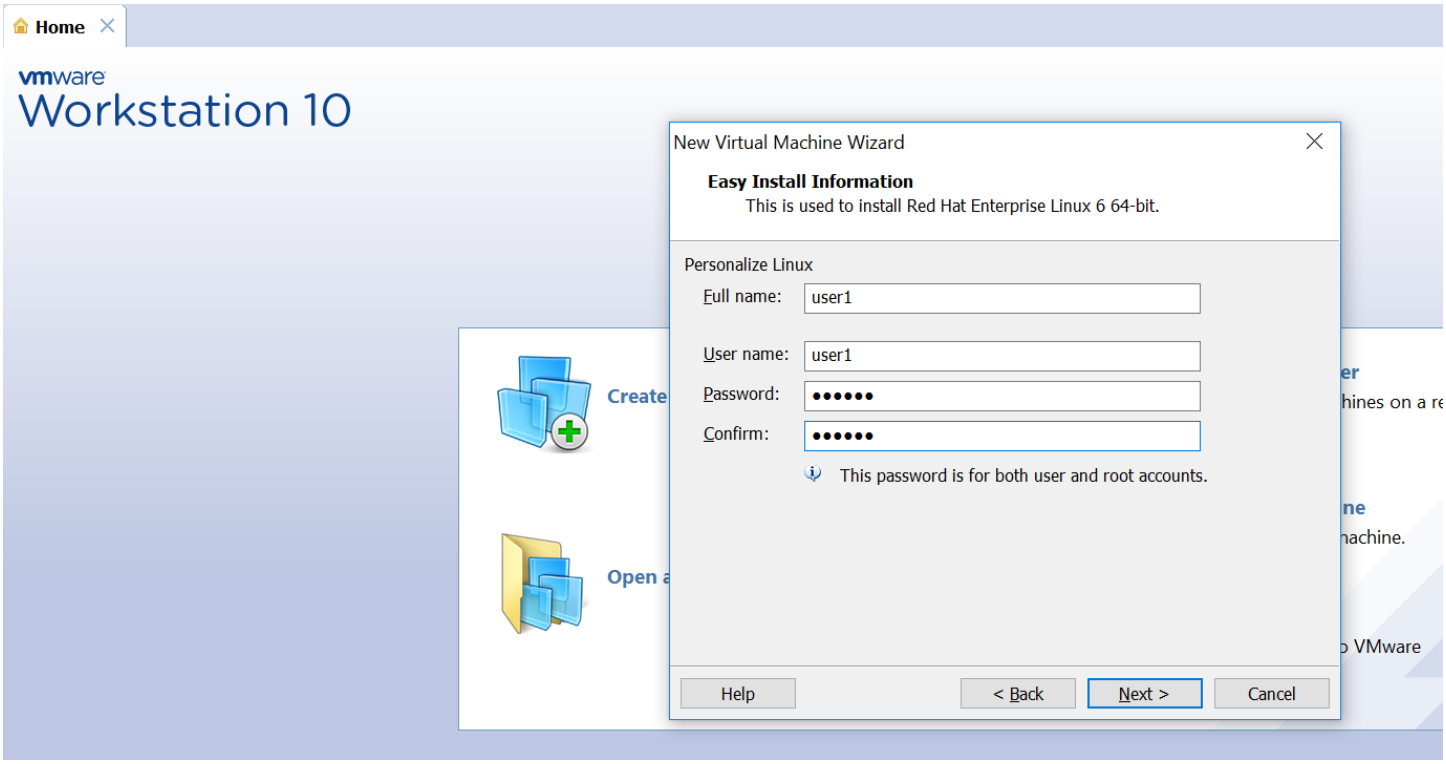
Now its time to create normal user and its password. Create the same with below credentials.

Full name: user1

User name: user1

Password: redhat

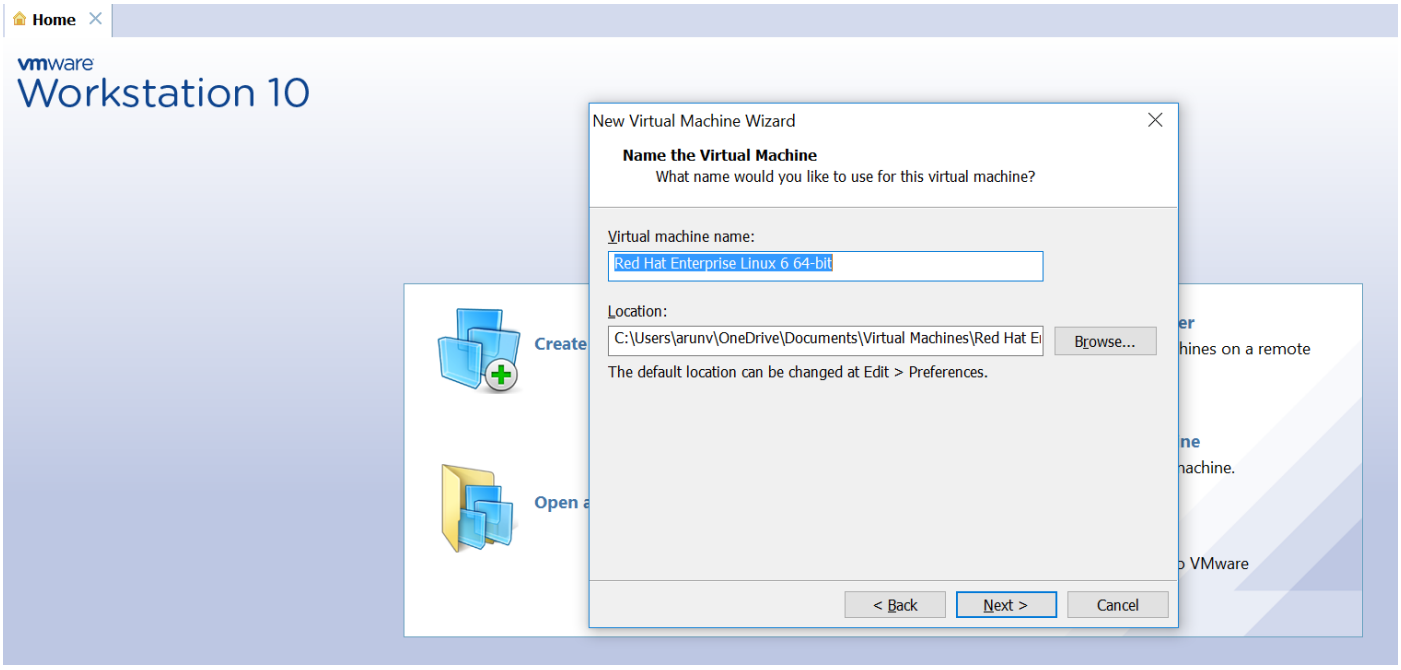
Confirm: redhat

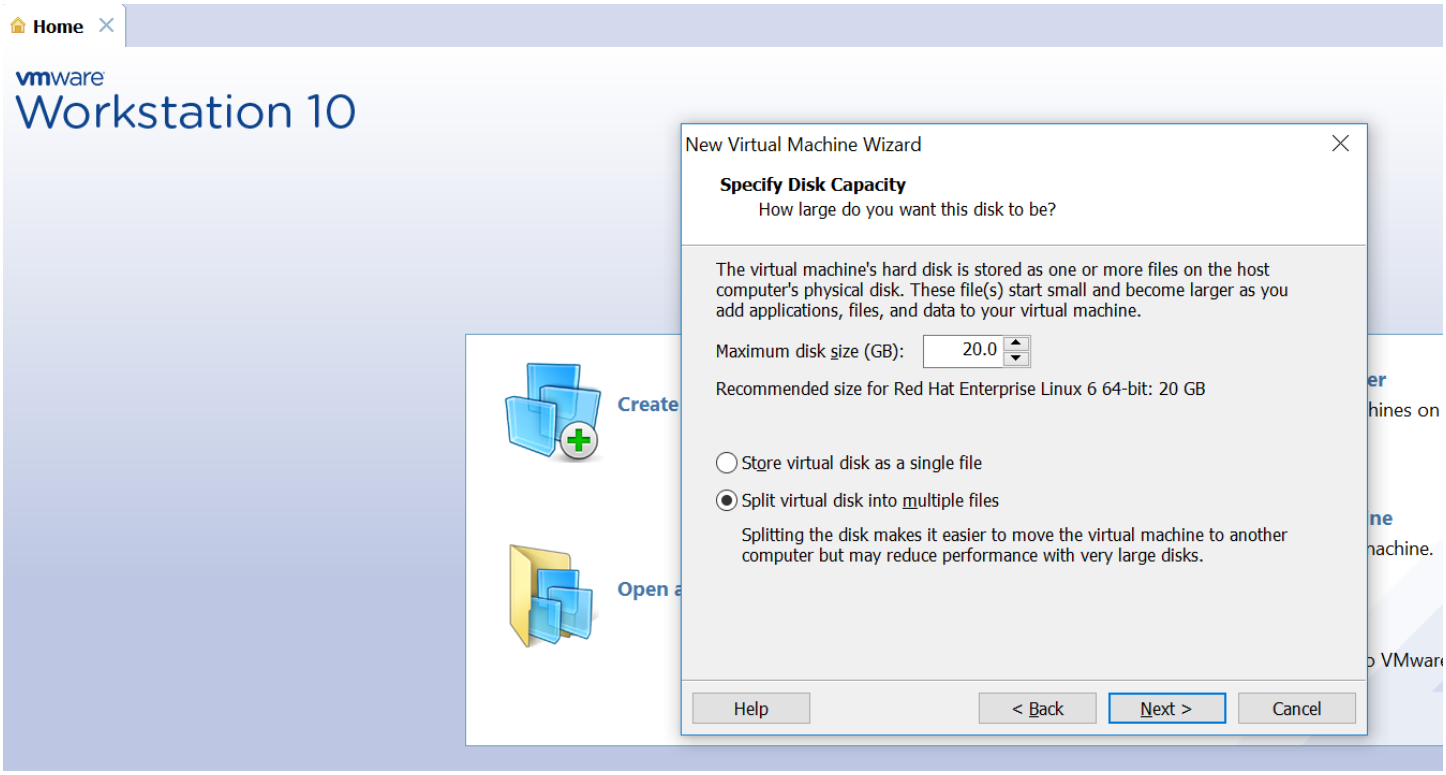


Step 5:

Choose your installation folder, by default it will be installed on C: drive.

You can change the installation destination where ever you have the space(20GB) on your windows machine.



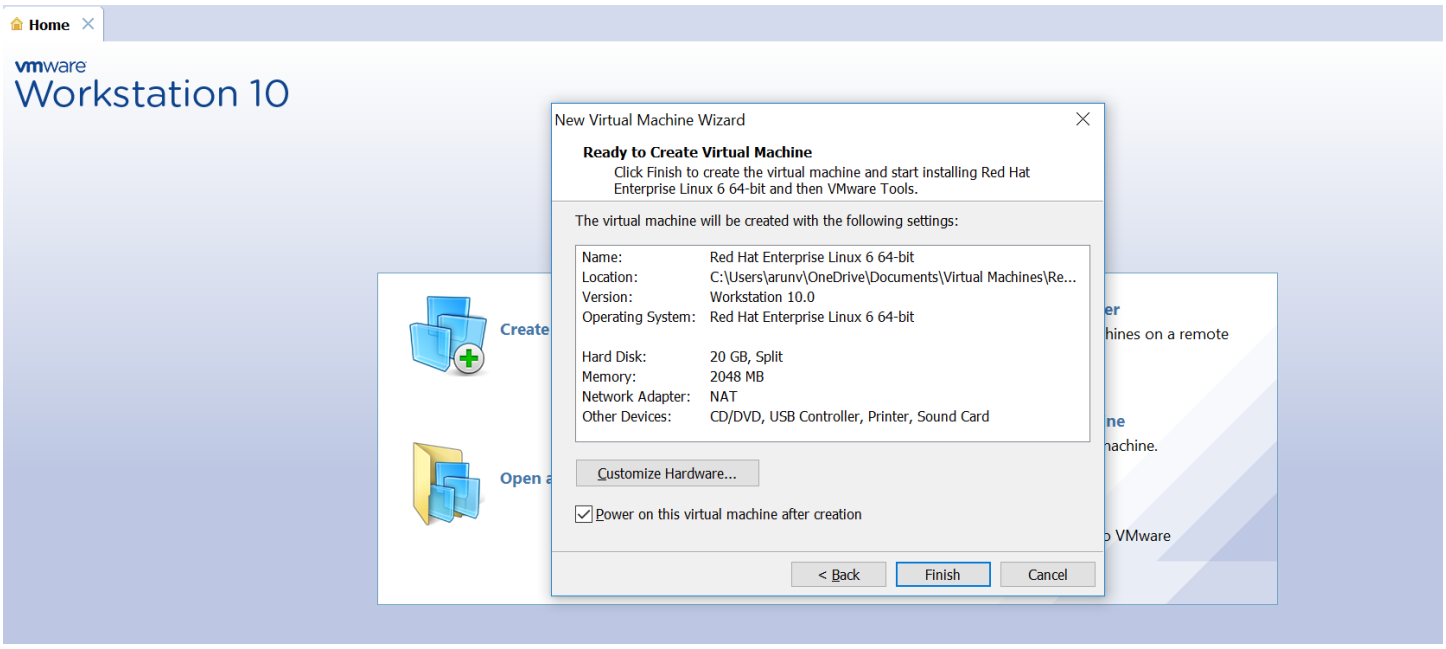


Step 6:

You can now summarize your installation parameter, before starting with installation process.

You can also increase your VMware image(RHEL 6.0) RAM size by clicking on *"Customize Hardware"*, if required.

Click on *Finish* now.



Step 7:

Now installation will start automatically as like below screen shots.

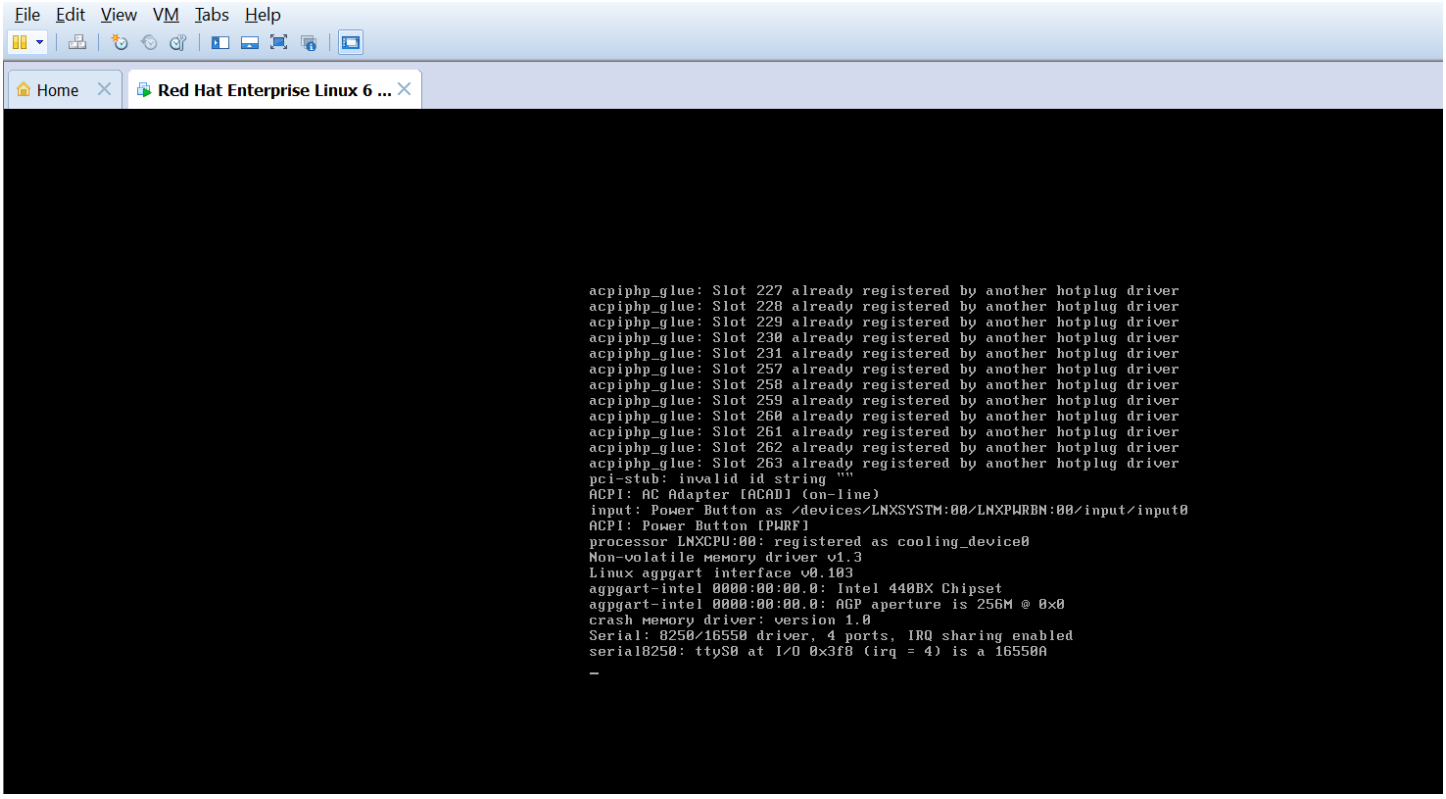
It ll will not ask for any information, because its an easy installation with pre-configured setup.

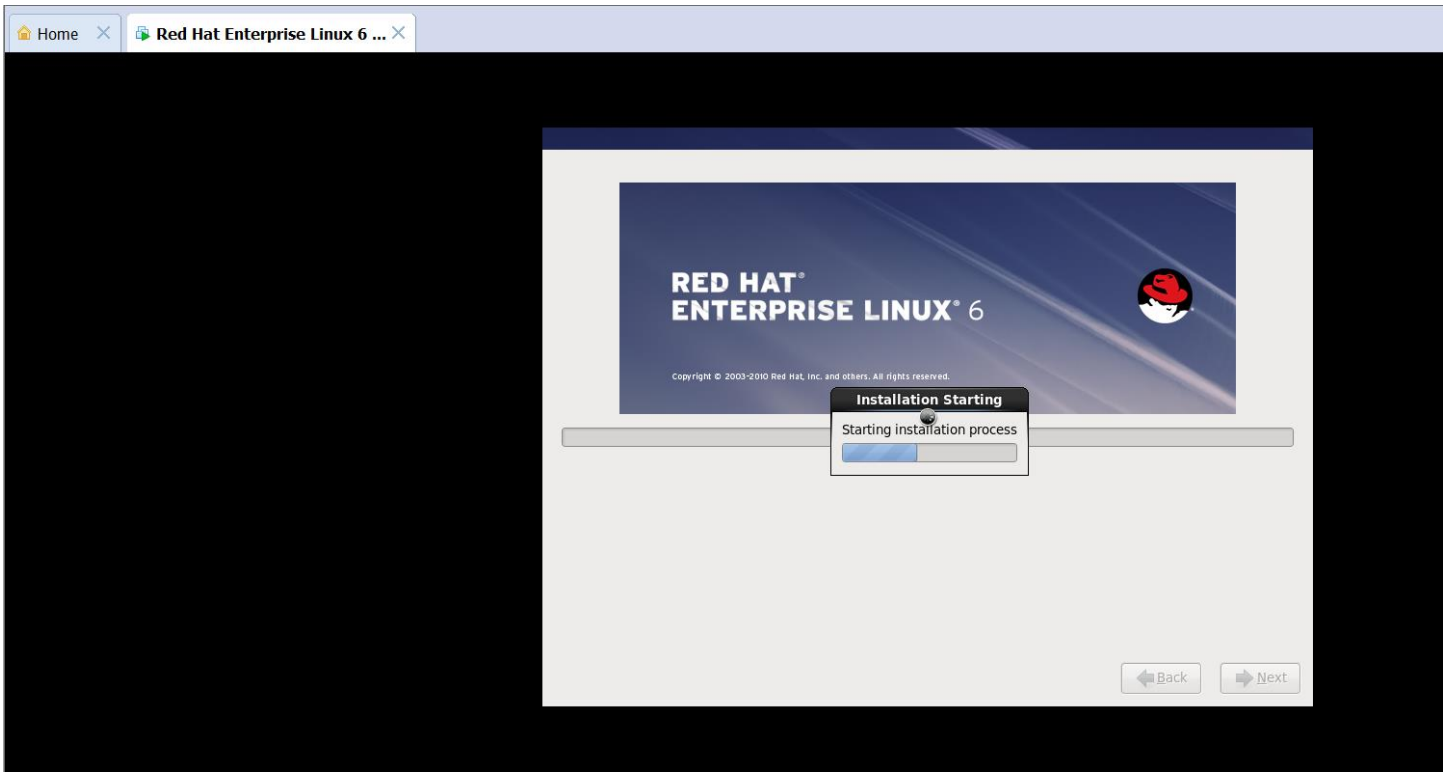
It ll restart the Virtual image (RHEL 6.0) automatically and will end up in asking *username* and *password.*

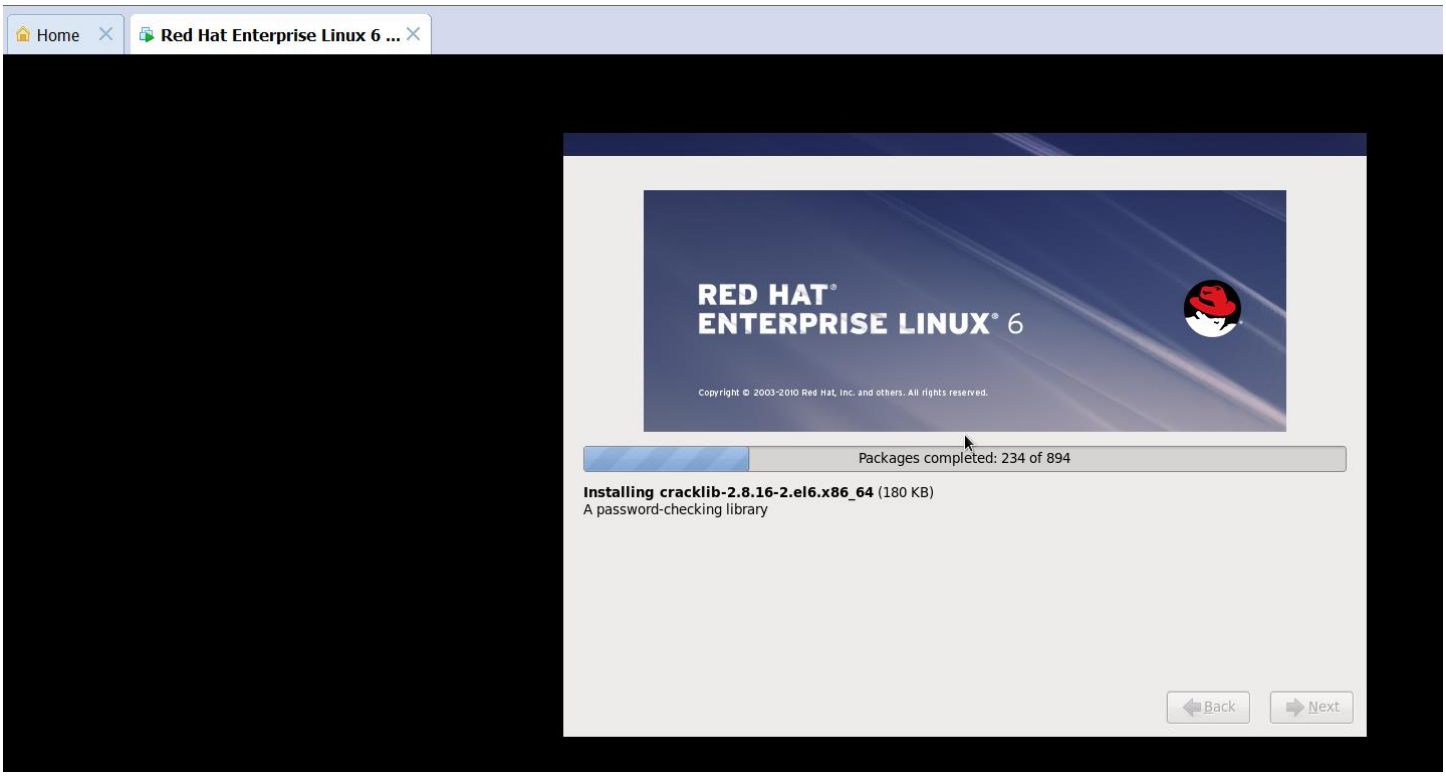
Provide below credentials by clicking on other:

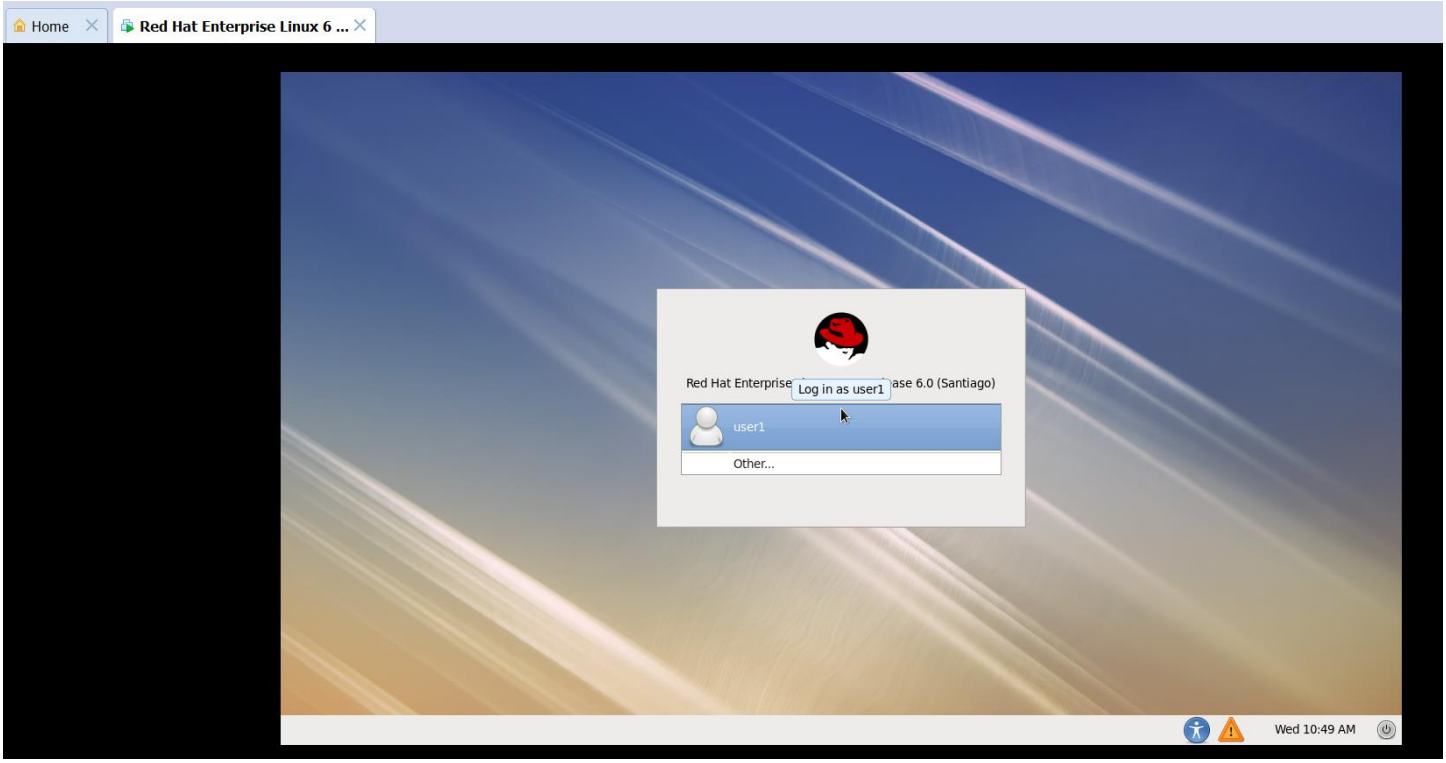
Username: root

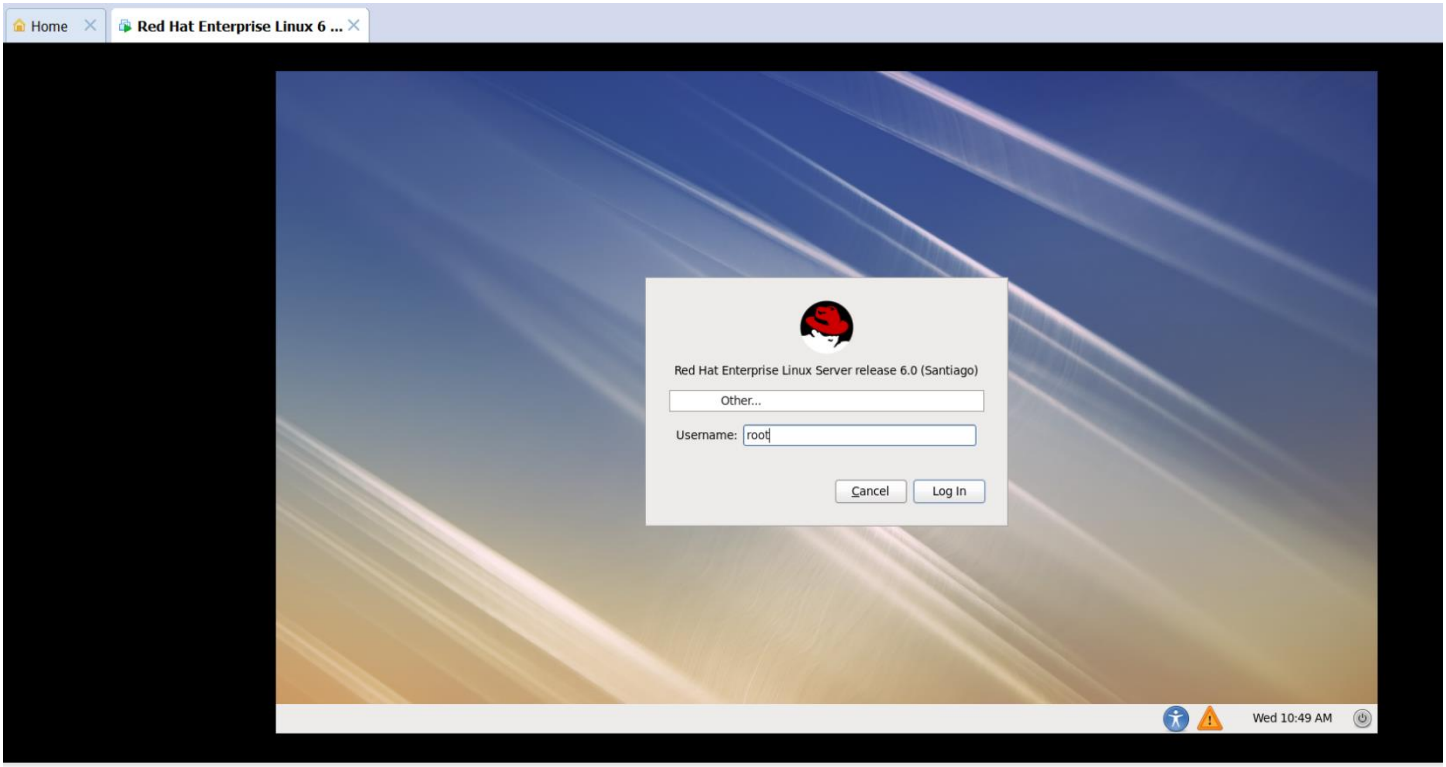
Password: redhat

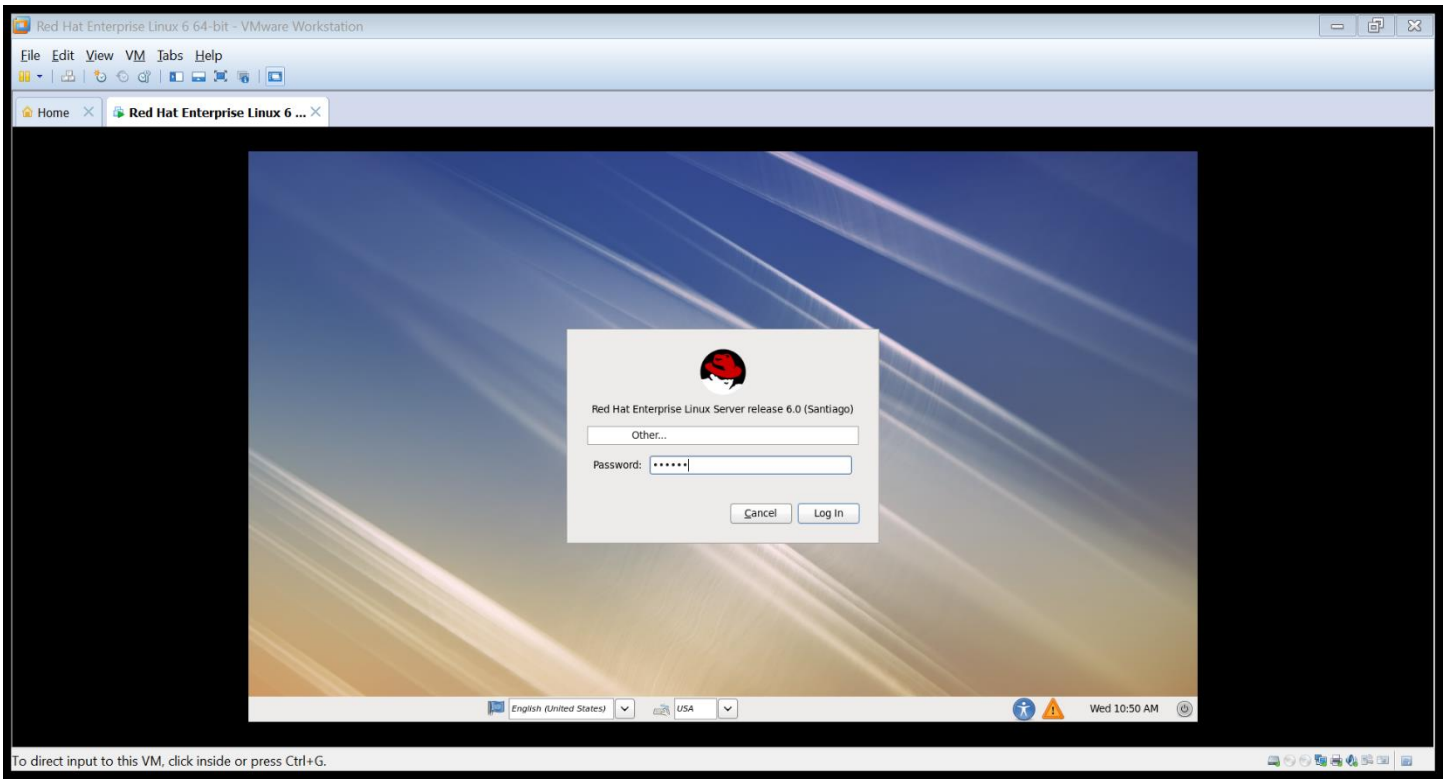



Home
Red Hat Enterprise Linux 6 ...
RED HAT®
ENTERPRISE LINUX® 6
Copyright © 2003-2010 Red Hat, Inc. and others. All rights reserved.
Installation Starting
Starting installation process
Back
Next


Home
Red Hat Enterprise Linux 6 ...
RED HAT®
ENTERPRISE LINUX® 6
Copyright © 2003-2010 Red Hat, Inc. and others. All rights reserved.
Packages completed: 234 of 894
Installing cracklib-2.8.16-2.el6.x86_64 (180 KB)
A password-checking library
Back
Next

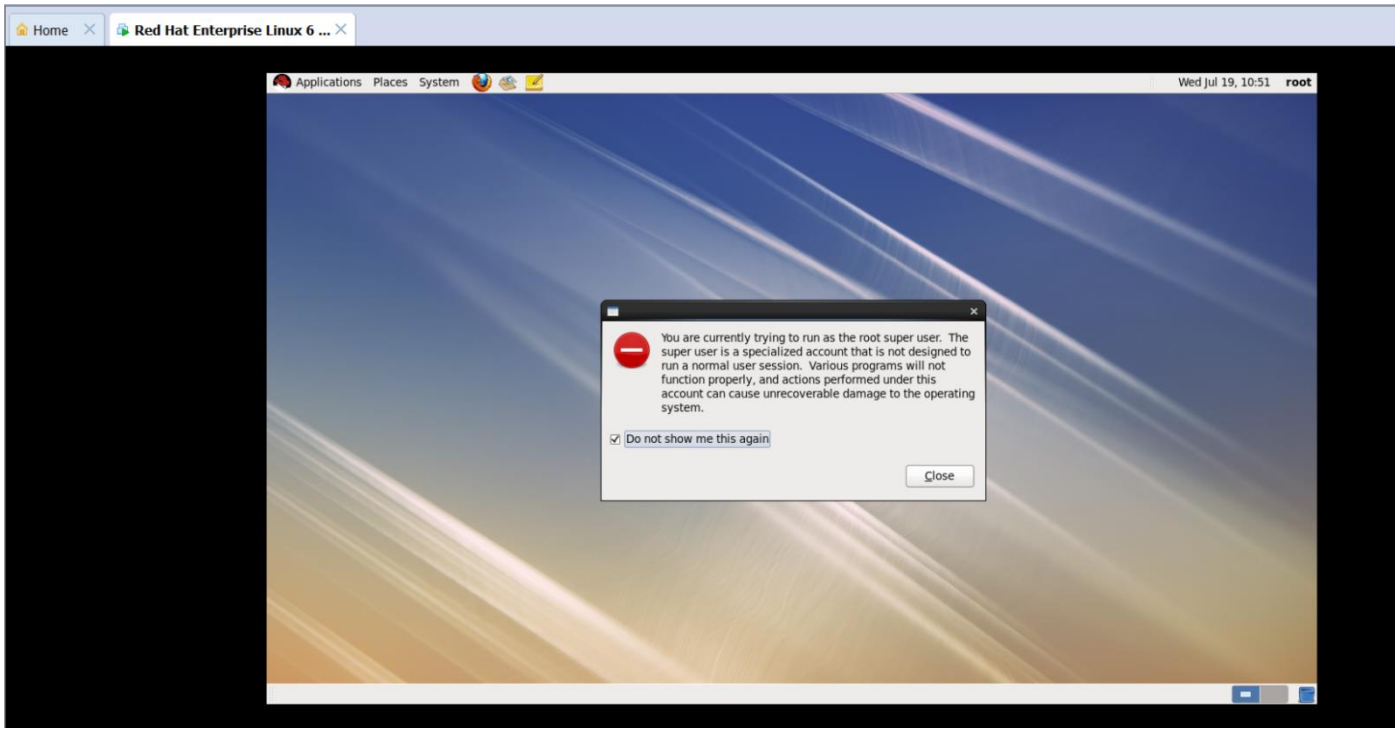

Home
Red Hat Enterprise Linux 6 ...
Red Hat Enterprise
ase 6.0 (Santiago)
Log in as user1
user1
Other...
Wed 10:49 AM


Home
Red Hat Enterprise Linux 6 ...
Red Hat Enterprise Linux Server release 6.0 (Santiago)
Other...
Username: root
Cancel
Log In
Wed 10:49 AM


Red Hat Enterprise Linux 6 64-bit - VMware Workstation
File Edit View VM Tabs Help
Home
Red Hat Enterprise Linux 6 ...
Red Hat Enterprise Linux Server release 6.0 (Santiago)
Other...
Password:
Cancel
Log In
English (United States)
USA
Wed 10:50 AM
To direct input to this VM, click inside or press Ctrl+G.

Step 8:

Click on the check box in order not to reoccur.

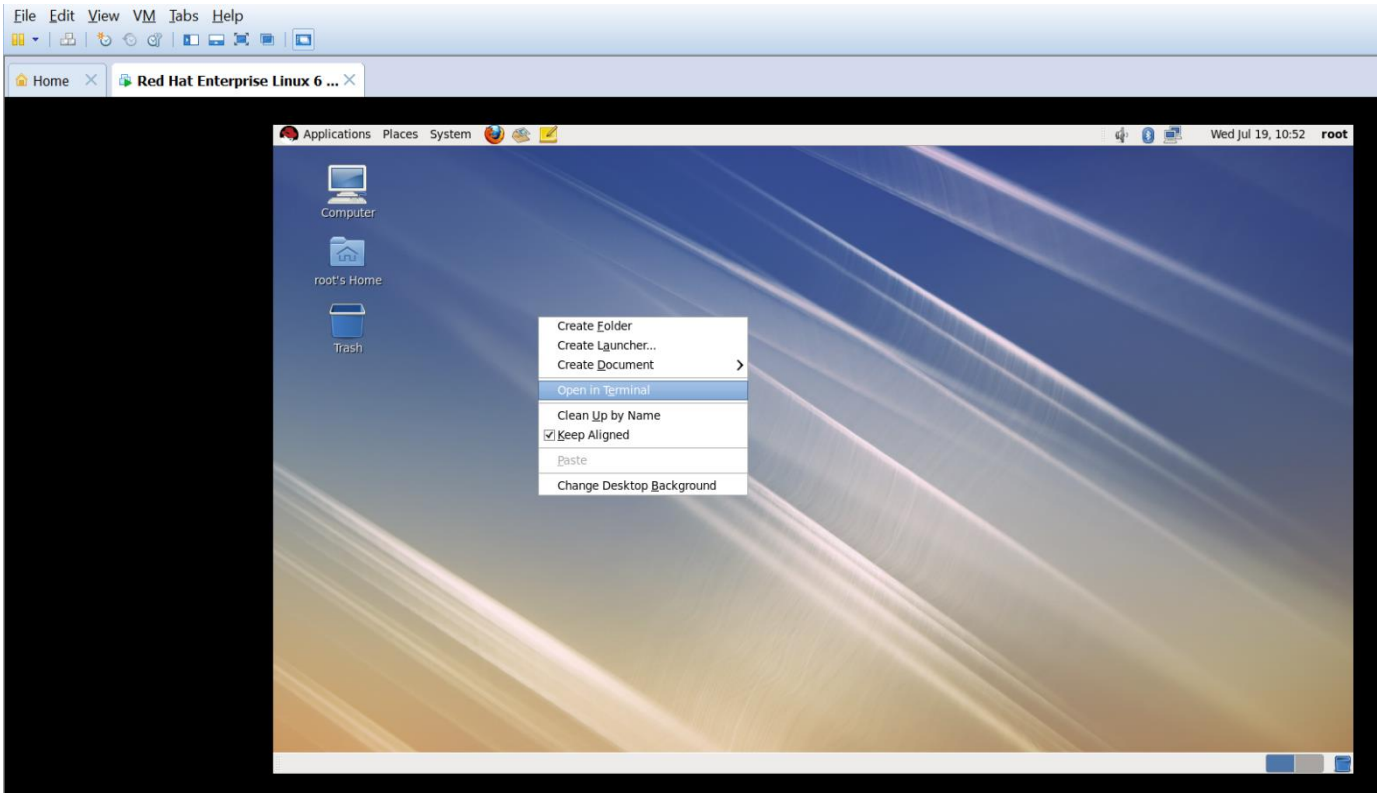


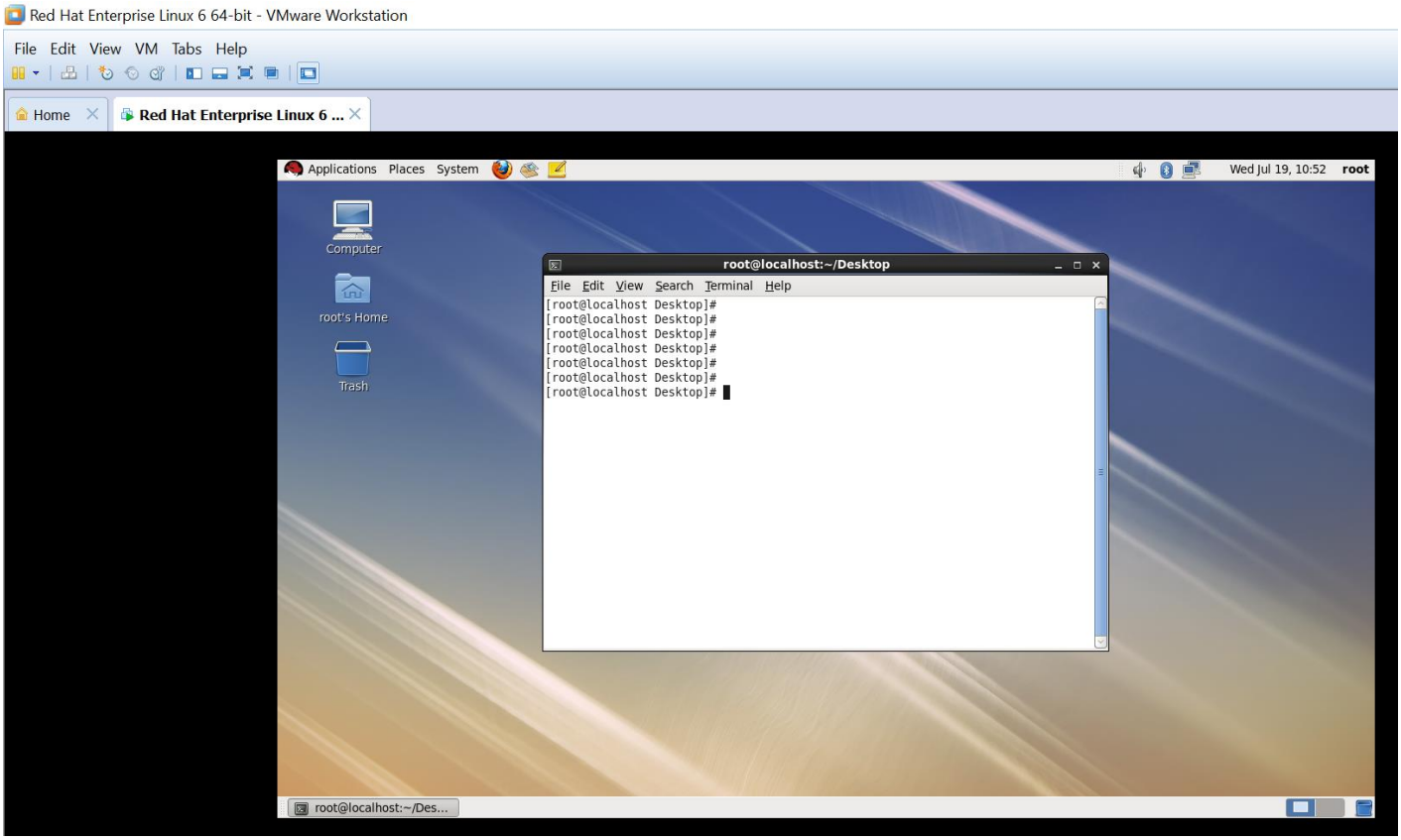
Step 9:

Once you are able to see desktop icons, right-click your mouse and click on *Open in Terminal.*

Now its time to start executing unix/linux commands

Enjoy ☺





Now, we are done with installation of RHEL 6.0 in VMware tool.

History

UNIX evolved at AT&T Bell Labs in the late sixties.
The writers of Unix are Ken Thomson, Rudd Canaday, Doug McIlroy, Joe Ossanna and Dennis Ritchie.
It was originally written as OS for PDP-7 and later for PDP-11.
UNIX OS exhibits the following features:

Features

It is a simple User Interface.
It is Multi-User and Multiprocessing System.
It is a Time Sharing Operating System.
It is written in "C" (HLL).
It has a consistent file format - the Byte Stream.
It is a hierarchical file system.
It supports Languages such as FORTRAN, BASIC, PASCAL, Ada, COBOL, LISP, PROLOG, C, C++, and so on.

Functions

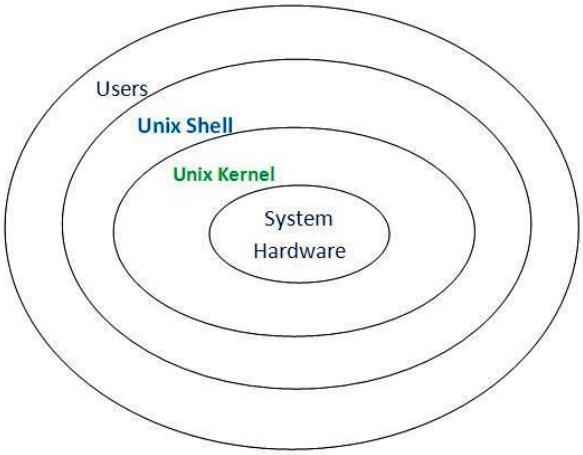
Process Management
Main-Memory Management
Secondary-Storage Management
I/O System Management
File Management
Protection System
Networking
Command-Interpreter System

INTRODUCTION

Unix based OS	
Linux Flavours	Other Unix OS
RHEL 5/6/7	IBM AIX
Cent OS	HP UX
Ubuntu	Oracle solaris
Mandriva	Oracle linux
Fedora	Mac

*

UNIX ARCHITECTURE:

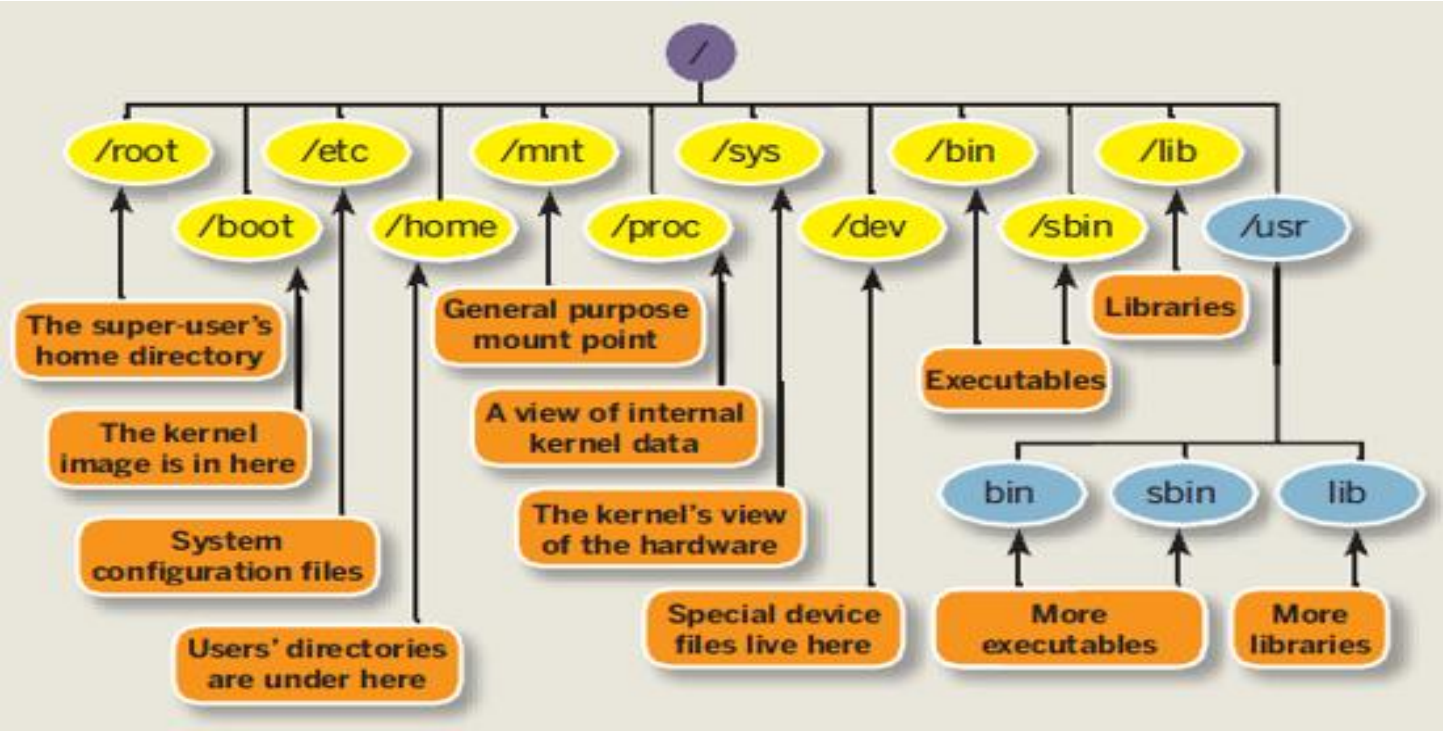


Unix: Uniplexed Information and Computing System.

Kernel: In unix OS its an code and acts as a bridge between the activities by user and hardware.

Shell: It's a program that interprets commands and acts as like an interface between users and kernel

FILESYSTEM HIERARCHY



/	base directory
/root	This is the home directory of root user and should never be confused with '/'
	—
/bin	All the executable binary programs (file) required during booting, repairing, files required to run into single-user-mode and other important, basic commands viz., cat, du, df, tar, rpm, wc, history, etc.
/sbin	Contains binary executable programs required by System Administrator, for Maintenance. Viz., iptables, fdisk, ifconfig, swap on, reboot, etc.
/boot	Holds important files during boot-up process, including Linux Kernel. kernel + os boot loader(grub2)+ kdump
	Till RHEL 4 boot file: LILO
	After RHEL 5 boot file: GRUB After RHEL 7 boot file: GRUB2
/proc	Its file-system which contains information about running process with a particular pid. No one can create files/folders in this directory
/srv	Service is abbreviated as 'srv'. This directory contains data files for specific services FTP,WWW, NTP, NFS, DNS
/sys	Modern Linux distributions include a /sys directory as a virtual filesystem, which stores and allows modification of the devices connected to the system. This contains info as same as /proc

/tmp	System's Temporary Directory, Accessible by users and root. Stores temporary files for user and system. Files exist till age of 30days. If its more than 30days it ll be automatically deleted.
/usr	Contains executable binaries, documentation, source code, libraries for second level program. eg for services (ftp, nfs). Files saved here will be shared across multiple machines.
/var	Stands for variable. The contents of this file is expected to grow. This directory contains log, lock, spool, mail and temp files. eg: databases, cache directory, log files, printer spooled documents
/etc	Place for service's configuration files like DNS, DHCP, Webserver, FTP, NFS
/dev	Contain device files for HDD, CDROM, USB etc

MOUNTING DIRECTORIES:

/media	Temporary mount directory is created for removable devices viz., media/cdrom.
/mnt	Temporary mount directory for mounting file system.
/opt	Optional is abbreviated as opt. Contains third party application software. Viz., Java, etc. Files related to package will be placed under /opt

COLOR CODING FOLLOWED IN RHEL:

White - normal file
Blue - directory/folder
Green - executable file
Sky blue -link file
Red highlighted - setuid permission
Red - .tar file
Green highlighted – sticky bit

PROMPT STRING:

[root@hp ~]#

root	current user
hp	hostname of a machine
~	pwd -present working directory/current directory
#	to indicate current user is root user
$	to indicate current user is normal user

COMMAND SYNTAX:

In unix, basic command syntax follows this way.

command -options argument1 argument2...argumentN

Command -- is a set of program executed in a single transaction
Options -- to change the output format of command
Argument -- input to the commands

# ls	only cmd
# ls –l	cmd with option
# ls f2	cmd with argument/input
# ls –l f2	cmd with option and argument

 # ls –ltrhiSRaFd

a --hidden files
l -- long list
t -- sort the files based on time modified
r –- reverse
i -- inode number
h -- human readable format for size
S -- sort files based on size
R -- Show contents of all sub folders/files with properties--
F – to differentiate files as normal file, executable file, directory
d - to check the details of directory

BASIC COMMNANDS

date
cal

cal -1
cal -3
cal 7 2015
whoami
hostname

HELP COMMANDS

man date - (manual page) man ls, man cp, man mkdir
cp --help

FILE AND FOLDER MANAGEMENT

cat : to concatenate more than one files

# cat > filename	To add content to a file and overwrites previous data
# cat >> filename	To append the contents by not disturbing the previous data
# cat filename	To view whole contents of file
# cat –n filename	To view whole contents of file with line numbers
# cat filename1 filename2	To merge output of two files

FILE :
ls –l a1.txt
-rwx rwx rwx 1 root root 0 Aug 31 00:18 a1.txt
set

-	---	---	---
-	rwx	rwx	rwx
file type	file owner (u)	group (g)	others (o)
	a=u+g+o		

Read	R	4
Write	W	2
Execute	X	1

r-- r-- r--.1 root root 0 Aug 31 00:18 a1.txt

- -- type of file (-,d,l,p,b,c,s)
rw- r—r- -- permission (owner group others)
1 -- no of link to a file/folder
root -- owner of a file
root -- file belongs to a group

0 -- size of a file
Aug 31 00:18 -- last modified date/time
a1.txt -- file/folder name

GNU `ls' uses a `.' character to indicate a file with an SELinux security context, but no other alternate access method.

A file with any other combination of alternate access methods is marked with a `+' character.

Memory measurement

8bits - 1byte
1024b - 1kb (kilo byte)
1024kb - 1mb (mega byte)
1024mb - 1gb (giga byte)
1024gb - 1tb (tera byte)
1024tb - 1pb (peta byte)
1024pb - 1eb (exa byte)
1024eb - 1zb (zetta byte)
1024zb - 1yb (yotta byte)

COPY, RENAME, MOVE, REMOVE FILE/DIRECTORY

Files:

cp - used to take backup of file/folder
- used to copy a file/folder from one path to another path

mv - used to rename a file/folder
- used to cut a file/folder from one path and paste in another path

rm - used to remove the file/folder

# cmd <options> <source-path> <destination-path>	Syntax
# cp -v 1.log /dev # cp -vp 1.log /dev	Copying file 1.log to /dev directory -v, verbose, to show status of a command -p, preserve, to copy file/directory with original date/time to another directory.
# mv -v 1.log 2.log	Renaming file from 1.log to 2.log
# mv -v 2.log /dev	Moving file 2.log from current directory to /dev directory
# rm -v 3.log	Removing file 3.log

/d1
/d2
/d3
/d4

Directories:

# mkdir college	to create directory in the current directory
# mkdir /school	to create directory under /
# mkdir -p /d11/d12/d13 /d21/d22/d23	to create directory with parent and child relationship
# mkdir –p /school/{a,b,c}	to create multiple directories
# mkdir /school/{1..10}	to create multiple directories using ranges

# cd	this ll take u to current user's home directory
# cd ..	this ll take u 1 level back from the current directory
# cd ../..	this ll take u 2 levels back from the current directory
# .	this indicates pwd (present working directory)
# cd ~	this ll take u to current user's home directory

# cmd <options><source-path><destination-path>	Syntax
# cp -r log /dev # cp -rvp log /dev	Copying directory log to /dev directory -v, verbose, to show status of a command -p, preserve, to copy file/directory with original date/time to another directory.
# mv -v log exe	Renaming log directory to exe
# mv -v 2.log /dev	Moving directory log from current directory to /dev directory
# rm -rv log	Removing directory log

Absolute Path:
To copy a file/dir starting from the head of the path.
It should start with /

cp -v /ss /tmp/govind/
cp -rv /root/arun/ /tmp

Relative Path:
To copy/move a file/dir by having pwd(present working directory) as reference path.

/d1/d2/d3

D3]# cp -v ../ss /tmp/govind/
D3]# cp -rv ../../csc /tmp

cp *.log *.pdf ~

Considering the directory structure as /d1/d2/dam/d3/d4

Copying directory dam to d4	Source Path	Destination Path
d2]# cp -rv /d1/d2/dam /d1/d2/dam/d3/d4	Absolute	Absolute
d2]# cp -rv dam /d1/d2/dam/d3/d4	Relative	Absolute
d2]# cp -rv /d1/d2/dam d3/d4	Absolute	Relative
d2]# cp -rv dam d3/d4	Relative	Relative

FILE TYPES:

Regular Files (-)
-rw-r--r--. 1 root root 39K May 6 13:05 install.log
Directory files(d)
drwxr-xr-x. 2 root root 4.0K May 6 07:38 Music
Character file(c)
Its used to communicate with hardware device with one character at a time ….arun
crw------- 1 ranga users 4, 0 Feb 7 13:47 /dev/tty0
Block file(b)
Its used to communicate with hardware device with large blocks. Arun raban van
brw-rw---- 1 root disk 8, 0 Feb 7 13:47 /dev/sda
Pipe file(p)
files that act just like temporary anonymous pipes on the command line.
prw-r--r-- 1 ranga wheel 0 Nov 22 17:39 mypipe

a -> pf -> b

Link files:

Hard link:

Used to create to links for a file.

Soft link or symbolic file: (shortcut in MSwin)
Used to create to shortcuts for files/folders.

lrwxrwxrwx. 1 root root 14 May 25 00:54 system-release -> centos-release

Link Files:

Hard link:
Same inode number and same permission
Can create only within a same partition
If original file is deleted, then link file will exist.
Applicable for only files
Files manipulation affects both the files.

ln <original_file> <link_file>
ln file1 file1_hl -- syntax to create hardlink

ls -li file1 file1_hl -- to view files with inode numbers
1835659 -rw-r--r--. 2 root g7 25 Jul 12 21:06 file1
1835659 -rw-r--r--. 2 root g7 25 Jul 12 21:06 file1_hl

Soft link or symbolic file: (shortcut in MSwin)
Can create with different partition in different filesystems.
Different inode number and different permission.
If original file is deleted, then link file will not exist.
Applicable for both files/folders.
Files manipulation affects both the files.

ln -s /root/file2 /dev/file2_sl (shortcuts in windows) -- syntax to create softlink

ls -li /root/file2 /dev/file2_sl
219182 lrwxrwxrwx. 1 root g7 5 Jul 12 21:10 /dev/file2_sl -> file2
835747 -rw-r--r--. 1 root g7 22 Jul 12 21:09 file2

VIRTUAL EDITOR
File editing tool

vi/vim

vi modes:
ESC mode : press esc key
INSERT mode : press i key
COMMAND mode : press shift :

VISUAL MODE:

insert mode -------→ esc---→ command mode
command mode---→ esc---→ insert mode

VISUAL Mode

:q quit
:q! force quit
:w Write(save) alone
:wq Write(save) and quit
:wq! force Write(save) and quit

vi <filename>
--commands with : should be executed in command mode
--commands without : should be executed in esc mode

copy	
Yy	to copy the single line
nyy, 2yy, 5yy	to copy n number of lines
yw	copy single word
nyw, 3yw, 6yw	to copy n number of words
cut	
cc	to cut single line
ncc, 2cc, 5cc	to cut n number of lines
cw	to cut single word
ncw, 3cw, 6cw	to cut n number of words
paste	
P	to paste below the cursor
P	to paste above the cursor

np, 2p, 8p	to paste n number of times
delete	
x or delete key	to delete single character
dw	to delete single word
ndw, 2dw, 6dw	to delete n number of words
ndd, 2dd, 6dd	to delete n number of lines from the cursor
cursor movement	
Shift + a	To move end of line and change to INSERT mode
Shift + d	After cursor position, data will be deleted.
gg	it takes to first line of file
G	it takes to last line of file
:101	to move cursor to the particular line number
Viewing more than 1 files	
:split <filename>	to read other file (we can include n no of files, cp ct also possible
ctrl + ww	to shift the cursor between the files
:read <filename>	to copy contents from other file to the current file
:!<cmd>	to execute cmd inside vim
:wq <filename>	to save as the another filename
search and replace	
:/<word>	to search the words (n-forward search, N-backward search)
:s/old/new	Finds each occurrence of 'old' (in the current line only) and replace it with 'new'
:%s/old/new/	Finds each occurrence of 'old' (in all lines) and replace it with 'new'
:%s/old/new/g	replace all old with new throughout file, g global
:%s/old/new/gc	replace all old with new throughout file with confirmations, c confirmation
:%s/\<foo\>/boo/	changes only whole words exactly matching 'foo' to 'bar'
%s/foo/boo/gi	to replace with case insensitive
:s/\(hello\\|hai\)/hi/g	Replace changes with more than one word in source
:s/\v(hello|hai)/hi/g	
:2s/old/new/	replace the changes at 2nd line
:2,5s/old/new/	Replace the changes from 2nd to 5th line
:.,$s/foo/bar/g	change each 'foo' to 'bar' for all lines from current line (.) to the last line ($)
:.,+2s/foo/bar/g	change each 'foo' to 'bar' from current line (.) to next two lines (+2).
:g/^baz/s/foo/bar/g	change each 'foo' to 'bar' in each line starting with 'baz'
set options can be manually used when required or use permanently by making a entry in ~/.exrc file	
:set nu	to set line numbers
:set nonu	to remove the line numbers
:set list	to show end of a line
:set ic	to remove case sensitive during the search
:set ruler	to show row and column positions

[root@server ~]# cat /root/.exrc
set nu
set list
set ruler

touch :to change the file/directory timestamps

# touch filename	To create a empty file
# touch filename1 filename2 filename3	To create multiple empty files
# stat filename	To get statistics of a file
# touch –a filename	To change access time for a file
# touch –m filename	To change modified time for a file
# touch –m *.log	To change modified time for the .log extension file
# touch –c filename	To change a, m, c time with current server time
# touch -t 199901010101.01 filename	To change date/time as per user desire
# touch -d "2012-10-19 12:12:12.000000000 +0530" tgs.txt	To change date/time as per user desire
# touch filename1 –r filename2	To change the timestamp of filename1 w.r.t filename2

FILTERS

head: to display top of file

head <filename> -- to view first 10 lines of a file
head -n <filename> -- to view n no of lines of a file
head -3 <filename> -- to view first 3 lines of a file
head –n -3 <filename> -- to view all lines expect last 3 lines of a file

tail: to display bottom of file

tail <filename> -- to view last 10 lines of a file
tail -n <filename> -- to view last n no of lines of a file
tail -4 <filename> -- to view last 4 lines of a file
tail –n +2 <filename> -- to view all lines expect first 2 lines of a file(used when filtering files with header)

less: to display file contents in moving the cursor up and down.

U can go both upwards and downwards.
Using spacebar, file will be scrolled page by page
Using enter, file will be scrolled line by line.

less <filename>

Pipeline: considering output of first command as a input for the second command

cmd1 -->o/p | ---> i/p cmd2

nl numbers of line in a file

nl -i2 emp.txt
nl -v 0 emp.txt
nl -v 10 emp.txt
nl -v 0 -i5 emp.txt
nl -v 10 -i5 emp.txt

grep

-- global regular expression print
-- capture the matching pattern from the files/command

Syntax:

grep <options> <pattern> <filenames>

# grep is example-grep.txt	string match
# grep color is example-grep.txt	to print matching string in color
	to make this as permanent , add a entry in /etc/environment as env variable,
	export BASH_OPTIONS=color
# grep "is" file1.txt example-grep.txt	using grep for more than 1 file
# grep -w is example-grep.txt	w exact word match
# grep -wi "Data entry hi hello" example-grep.txt	i removes case sensitive
# grep -A 2 -w 'is' example-grep.txt	A to print 2 lines after the match..line numbers can be any

# grep -B 2 -w 'is' example-grep.txt	B to print 2 lines before the match..line numbers can be any
# grep -C 2 -w 'is' example-grep.txt	C to print 2 lines before and after the match..line numbers can be any
# grep -r "arun" *	to search the pattern in all files under current directory
# grep -v "arun" *	to search the pattern expect the string "arun".
# grep -c "arun" example-grep.txt	counts no.of matching pattern
# grep –vc	to find out no.of lines that does not match
# grep "default=0" /etc/*	to search in particular directory
# grep -l "arun" *	print the file names alone containing the pattern "arun"
# grep -n "arun" *	print the matching string with the line numbers from the file
# egrep "arun|vishak" example-grep.txt	to search the pattern for more than 1 string arun and vishak (e extended) . | this is alternative operator in reg.exp
# egrep '^hi|hello$' example-grep.txt	to find the lines starting(^) with string "hi" or ending($) with string "hello"
# egrep '^.{47}3' example-grep.txt	to find a pattern from a position. 47th is a position
# fgrep "$" example-grep.txt	To find any spl character "$" is existing in a file

Meta characters - * ? [] ! ^ $

*	display string with one or more matching character
?	display string with single matching character
[afc]	display files/folders names starting with a or f or c
[0]	display files starting with 0
[01]	display files starting with 0 and 1
[a-f]	Range wise

[a-zA-Z1-9] display strings with combination of 3 characters for 1st position alone, a-z or A-Z or 1-9

[a-z][A-Z][1-9] display strings with combination of 3 characters for 1st,2nd,3rd positions a-z and A-Z and 1-9

a[\!]

[!a]*.txt display files and folders which are not starting with the letter a.

[!ap]*.txt

[!a-e]*.txt

#ls -l ar*
#ls -l ?ndia
#ls -l ??jan.txt
#ls -l 01??2015.txt
#ls -l [afc].txt
#ls -l [a-zA-Z1-9].txt
#ls -l [a-z][A-Z][1-9]report.txt
#2[0-9][0-9].fin

SORT:

```
[root@localhost ~]# cat emp
praveen m 22 analyst 30000 May 15-05-2016
saba f 23 analyst 40000 Jan 22-01-2016
lakshi f 42 manager 60000 Sep 05-09-2016
saks3 f 42 manager 60000 Sep 05-09-2014
sakshi m 3 anager 60000 Mar 22-03-2016
nilan m 36 assistant 800 Apr 05-04-2016
nilan m 36 assistant 800 Apr 05-04-2016
ankit m 28 architect 7000 Jun 11-06-2016
bavana f 30 tester 10000 Dec 02-12-2016
bavana f 30 tester 900 Dec 02-12-2016
rasheed m 32 developer 9000 Jul 16-07-2016
ghosh m 29 consultant 80000 Oct 08-10-2016
ghosh m 29 consultant 80000 Oct 08-10-2016
shreya f 33 vendor 5000 Nov 26-11-2016
shreya f 33 vendor 5000 Nov 26-11-1999
shreya f 33 vendor 5000 Nov 26-11-2009
shreya f 33 vendor 5000 Nov 06-11-2009
shreya f 33 vendor 5000 Nov 09-11-2009
```

sort filename	normal sort
sort -r filename	reverse sort
sort -u filename	prints unique records by eliminating duplicate records
sort -k2 filename	-k, column wise sort
sort -nk5 filename	sort 5th column by numeric
sort -Mk6 emp	sort 6th column by month wise
sort -nk7 emp | sort -rnk7.4	sort 7th column by numeric, then sort the 4th character from 7th column
sort -k4,4 -k5,5n emp	sort 4th first, then based 4th column sort the 5th column
sort -t',' -k2 filename	-t delimiter, when , is a delimiter

sort -t $'\t' -k1.3	when tab space is a delimiter

cut:

Nilan:m:36:assistant
Ankit:m:28:architect
bavana:f:30:tester
bavana:f:30:tester
rasheed:m:32:developer
ghosh:m:29:consultant
shreya:f:33:vendor
shreya:f:33:vendor

cut -cn filename	character wise
cut -cn-n filename	Range
cut -c10-15 filename	will cut from 10th to 15th position
cut -c10- filename	from 10th position to end of the line
cut -d':' -f3 filename	delimiter, column wise
cut -d':' -f1,2 filename	delimiter, column in range
cut -d' ' -f-3,6 filename	delimiter, column in range and individual
cut -d $'\t' -f1 bb	when tab space is a delimiter

tr:

Usage: tr [OPTION]... SET1 [SET2]
Translate, squeeze, delete characters from standard input writing to standard output.

-d, --delete delete characters in SET1, do not translate
-s, --squeeze-repeats replace each input sequence of a repeated character into single character.
Eg:

```
[root@server ~]# echo "hi boss, i ll be in office at 22-11-12" | tr -s 2
hi boss, i ll be in office at 2-11-12
[root@server ~]# echo "hi boss, i ll be in office at 22-11-12" | tr -s s
iihi bos, i ll be in office at 22-11-12
[root@server ~]# echo "hi boss, i ll be in office at 22-11-12" | tr -d 2
hi boss, i ll be in office at -11-1
[root@server ~]# echo "hi boss, i ll be in office at 22-11-12" | tr -d ' '
hiboss,illbeinofficeat22-11-12
```

```
[root@server ~
# echo "hi boss, i ll be in office at 22-11-12" | tr -s ' '
hi boss, i ll be in office at 22-11-12
[root@server ~]# echo "hii^    iii^    h^hh   hel$   llllo " | tr -d '^$'
hii    iii    hhh   hel   llllo
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr -s ' '
hi boss, i ll be in office at 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr [a-z] [A-Z]
HI BOSS, I  LL BE IN OFFICE AT 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr [:lower:] [:upper:]
HI BOSS, I  LL BE IN OFFICE AT 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr  [b] [bb]
hi boss, i  ll be in office at 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr  [" 'bb'
hi boss, i  ll be in office at 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr  'b' 'bb'
hi boss, i  ll be in office at 22-11-12
[root@server ~]#
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr 'b' 'bb'
hi boss, i  ll be in office at 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr 'hi' 'bb'
bb boss, b  ll be bn offbce at 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr 'h' 'bb'
bi boss, i  ll be in office at 22-11-12
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr '2' '3'
hi boss, i  ll be in office at 33-11-13
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr '2' '33'
hi boss, i  ll be in office at 33-11-13
[root@server ~]# echo "hi boss, i  ll be in office at 22-11-12" | tr 's' '@'
hi bo@@, i  ll be in office at 22-11-12
```

TEE:

Used to send the output to more than one stream.
As like name “T”, having one source of input and two dimensions of outputs.

```
# <cmd1> | tee <filename> | <cmd2>
#sort emp.dat  | tee tempfile | cut –f 3
# cut –d’ ‘ –f1 emp | tee log | head -3
```

WORD COUNT:

wc -lwc <filename>

l -- provides no.of lines in a file
w -- provides no.of words in a file
c -- provides no.of characters in a file

who | wc -l
head -1 /etc/passwd | wc -c

HISTORY

To execute the previously executed commands from a history file.

# history	to view previously executed command
# !!	to execute previous cmd
# !6	to execute previously executed cmd from the history
# history -c	clear the history for a current session only.
~/.bash_history	saves all the cmds in a file
# export PROMPT_COMMAND='history -a'	Append the new history lines to the history file.
# export HISTSIZE=5000	keep the last 5000 entries
# export HISTCONTROL=erasedups	ignore duplicate commands
# export HISTTIMEFORMAT='%F %H:%M'	changes the time format in history file
shopt -s histappend	append to the history instead of overwriting (good for multiple connections)

UNIQ:
It reports or filters out repeated lines in a file.
Filters out adjacent, matching lines from input file , writing the filtered data to output file.
It does not detect repeated lines unless they are adjacent. You need to **sort the input first**, or use sort -u instead of uniq.

# uniq <filename>	prints only unique lines for once
# uniq –d <filename>	prints only duplicate lines for once
# uniq –D <filename>	prints only duplicate lines with repeation

# uniq -c <filename>	give count of duplicate lines
# uniq –u <filename>	prints only unique lines

USER MANAGEMENT

useradd, passwd, chmod, chown, chgrp, setfacl, SUID, SGID, sticky bit , umask, acl, chage

User Types:

Root user
System user
Normal user

Root user (id = 0)
System users (id <= 499) rhel 6
System users (id <= 999) rhel 7
Normal users (id >= 500) rhel 6
Normal users (id >= 1000) rhel 7

```
[root@server ~]# ls -ltrh /etc/passwd
-rw-r--r-- 1 root root 1865 Jun 15 23:35 /etc/passwd
```

/etc/passwd -- user's basic info
/etc/shadow -- user's passwd details
/etc/group -- group details

/etc/passwd - 8fields

rahul:x:520:528::/home/rahul:/bin/bash

rahul	-	user name
x	-	encrypted password
520	-	user'id
528	-	group id
::	-	comments about the user.(analyst, manager)
/home/rahul	-	user's home directory.
/bin/bash	-	user's default login shell

bash, ksh, tch, csh

/etc/shadow - 8fields

halt:*:16413:0:99999:7::

halt - <username>
* - encrypted password
16413 - last password changed date
0 - min password changing date 30
60 - max password changing date 40
06/30/2017 - password expiry date
07/30/2017 - account expiry date

/etc/group - 4fields
daemon:x:2:root,bin,daemon
daemon - group name
x - encrypted passwd
2 - groupid
root,bin,daemon - members of the group

# useradd <username>	to add a user in a server
# useradd -c "Analyst" <username>	to create user with comments
# useradd -d /home/supp <username>	to create user with own home dir
# useradd -M test	to create user without home dir
# useradd -m -k /d1 <username>	to get contents from other dir in user's home dir
# useradd -e yyyy-mm-dd <username>	to create user with expiration date
# useradd -s login_shell <username>	to create user with assigned shell
# useradd –ou 0 –g 0 <username>	to create user as root user
# chage -l <username>	To check user passwd expiration details.
# chage -E yyyy-mm-dd <username>	To change expiration date for a user
# passwd <username>	to assign a passwd to a user
# passwd -l <username>	to lock user
# passwd -uf <username>	to unlock user
# passwd -d <username>	to remove passwd for user
# usermod -l <new-username> -m <old-username>	to change the <username>
# usermod -l <new-username> -d <new-dir name> -m <old-username>	to change the <username> with home dir
# usermod –g g4 <username>	to change the primary group for a User.
# usermod -G g1,g2,g3 <username>	to change a supplementary groups for a user
# usermod –a –G g4 <username>	To add supplementary group for a user
# usermod -d /var/www/html/ -s /bin/bash -e 2014-12-10 -c "This is Jack" -u 555 -aG apple jack	
# userdel <username>	to delete user

# userdel -r <username>	to delete user with user's home dir, mail, spool
# groupadd <groupname>	to add a group in a server
# gpasswd -d <username> <groupname>	to remove a user from a group
# gpasswd –M <user1,user2> <groupname>	to add users in a group
# groupmod -n <new_groupname> <old_groupname>	to change group name
# groupdel <groupname>	To delete a group
# lid <username>	To list groups containing particulat user
# lid -g <groupname>	To list group members
# chown <user_name> <dir_name>	to change the ownership for directory
# chown -R user_name dir_name	to change the ownership for directory and its sub directories
# chgrp grp_name dir_name	to change the group permission for directory
# chgrp -R grp_name dir_name	to change the group permission for directory and its sub directories
# chown -R user_name:grp_name dir_name	to change ownership and group ownership in single cmd.

#chown –R madan /d1
/d1
/d1/d2
/d1/d2/d3
==========
FILES
/etc/passwd -- User account information.
/etc/shadow -- Secure user account information.
/etc/group -- Group account information.

/etc/gshadow -- Secure group account information.
/etc/default/useradd -- Default values for account creation.
/etc/skel/ -- Directory containing default files.
==========

PERMISSION

Octal notation Permission:
#chmod 777 f1.txt
#chmod 640 f1.txt

Alphabet Permission:
chmod u+rwx,g+rwx,o+rwx f1.txt
chmod u+r f1.txt
chmod g-w f1.txt
chmod a+rwx f1.txt
chmod a-x f1.txt

Default permission:
root:
file : 644/666 -- 022
directory : 755/777 -- 022
normal user:
file : 664/666 -- 002
directory : 775/777 -- 002

Umask Value:

: Setting which determines what permissions are applied to a newly created file or directory is called a umask value
: By using this value, default value can be assigned.
: Umask value is inversely proportional to file/dir permission.

root:
file/directory : 022 - 644
normal user :
file/directory : 002 - 664

/etc/profile - file to edit umask value as permanent.
#umask 000

Umask value is inversely proportional to file/dir permission.

Sticky bit permission (t):

Used on directories alone.
Only the person who created the file within a directory may delete or rename it, even if other people have full permission.

You can turn it on by typing:

chmod +t <dirname>

chmod -t <dirname>

chmod 1700 <dirname>
chmod 700 <dirname>

Special Permission (s):

SUID - Set User ID
SGID - Set Group ID

Used on directories.
Used to add user ID and group ID permission to a file.

SUID:
File/process can run only under file owner name, not on the user who runs it.

chmod u+s <filename>
ls –l /usr/bin/passwd

SGID:
Group ownership provided to a directory, where all the file/folders created after setting SGID for a directory, will have same groupownership as parent directory.

chmod g+s <dirname>
chmod g-s <dirname>

chmod 2777 <dirname>

S.No	Special permissions	To add	To revoke
1	sticky bit	# chmod 1000 <dirname> # chmod +t <dirname>	# chmod -t <dirname>
2	set group id (sgid)	# chmod 2000 <dirname> # chmod g+s <dirname>	# chmod g-s <dirname>
3	sticky bit + sgid	# chmod 3000 <dirname> # chmod +t,g+s <dirname>	# chmod -t,g-s <dirname>
4	set user id (suid)	# chmod 4000 <filename> # chmod u+s <filename>	# chmod u-s <filename>
5	sticky bit + suid	# chmod 5000 <dirname> # chmod +t,u+s <dirname>	# chmod -t,u-s <dirname>

6	suid + sgid	# chmod 6000 <dirname> # chmod u+s,g+s <dirname>	# chmod u-s,g-s <dirname>
7	suid + sgid + sticky bit	# chmod 7000 <dirname> # chmod +t,u+s,g+s <dirname>	# chmod -t,u-s,g-s <dirname>

ACL: Access Control List:

As a System Admin, our first priority will be to protect and secure data from unauthorized access. We all are aware of the permissions that we set using some helpful Linux commands like **chmod, chown, chgrp**… etc. However, these default permission sets have some limitation and sometimes may not work as per our needs. For example, we cannot set up different permission sets for different users on same directory or file. Thus, Access Control Lists (ACLs) were implemented.

This permission sets have limitations. For example, different permissions cannot be configured for different users. Thus, Access Control Lists (ACLs) were implemented. From Red Hat Enterprise Linux 5 kernel provides ACL support for the ext3 file system and NFS-exported file systems.

```
# grep -i acl /boot/config*          -- to verify acl existence in server

# getfacl file/directory             -- to check ACL permission
# getfacl /root/fb

# setfacl -m  acl file/directory     -- to assign ACL permission for a user in file/directory
# setfacl -m  u:user1:rwx 1.py
# setfacl -m  u:arun:r 1.py
# setfacl -m  g:support:rwx 1.py

# setfacl -x  acl file/directory     -- to remove ACL permission for a user
# setfacl -x  u:arun: 3

# setfacl -b  file/directory         -- removing all ACL from file/directory
```

SUDO

To make normal user as a root user, we can sudo cmd.
To provide sudo access to a group, add the following line to the /etc/sudoers file.

```
sathiya          ALL=(ALL) ALL  -- sudo access to a user
%programmers     ALL=(ALL) ALL  -- sudo access to a group
```

sathiya : name of user to be allowed to use sudo
ALL : Allow sudo access from any terminal (any machine).
(ALL) : Allow sudo command to be executed as any user.
ALL : Allow all commands to be executed.

user1]# sudo fdisk /dev/sda

user1]# sudo –s cat /etc/passwd > /home/user1/passwd_bkp

su oracle	su – oracle
[root@localhost ~]# su oracle [oracle@localhost root]$	[root@localhost ~]# su - oracle [oracle@localhost ~]$
User "oracle" will be working on previous root user profile settings and not on oracle user's profile setting.	User "oracle" will be working on his own profile settings.

File Descriptors:

stdin 0 I/P
stdout 1 O/P
stderr 2 ERR

>	Overwrite the Contents or Redirecting
>>	Appending the previous data
<	Getting input from a file/user
1>	Re-directing output to a file
2>	Re-directing error to a file.
&	Re-directing to same terminal.
/dev/null	Error bucket, incoming data will be trashed immediately

#cat f1 f3 2> op1.txt

#cat f1 f2 2>/dev/null

#cat f1 f2 1>op.log 2>&1
1. output msg of f1 f2 is re-directed to op.log file
2. error msg is also re-directed to same file where output msg is getting redirected.

cat f1 f2 1>op.log 2>&1
First, the output got re-directed to the file op.log
2>&1 - This means re-directing the error(2) to the same terminal as the output(&1). Since output is re-directed to op.log, the error msg is also redirected to op.log.

#cat f1 f2 2>&1 1>op.log

This will not work as expected.
2>&1 - First, the error got re-directed to the same place as the output. Since the output re-direction is not defined yet, the error gets re-directed to its default terminal itself.
Secondly, the output is now re-directed to the file op.log. So, on running the above command, all the output gets re-directed to the output file op.log and the errors are captured in the terminal itself.

Process

When you execute a program on your UNIX system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.

Whenever you issue a command in UNIX, it creates, or starts, a new process. When you tried out the ls command to list directory contents, you started a process. A process, in simple terms, is an instance of a running program.

The operating system tracks processes through a five digit ID number known as the **pid** or process ID . Each process in the system has a unique pid.

Daemons - Daemon does not stand for Disk and Execution Monitor. They are the processes which run in the background and are not interactive. They have no controlling terminal. They perform certain actions at predefined times or in response to certain events. In UNIX, the names of most daemons end in d.

Services - In Windows, daemons are called services.

Process - Process is a running program. At a particular instant of time, it can be either running, sleeping, or zombie (completed process, but waiting for it's parent process to pick up the return value).

# ps	list process running in current shell
# ps -u	list current user's process
# ps -u raj	list only user raj process
# ps -ef	list full information of current user process
# ps –e	list all (root, normal users) process
# ps f	List processes in full format, with child process

pid	process id with 5 digit numbers
ppid	parent process id
# top	to get cpu/memory/process status
# kill –l	lists signals
# kill <pid>	graceful kill.
# kill -9 <pid>	immediate kill for a process id (-9 signal to terminate)

# pkill <pname>	to kill with a process name
# kill -STOP <pid>	to hold a process
# kill -CONT <pid>	to resume the process
Fg	foreground process
Bg	background process
&	to run a job in the background
# jobs –l	to list all the background(bg) running process with pid
# kill %2	to kill bg jobs with job number
# fg	this will bring recent bg job to fg.
# fg %20	this will bring required bg job using their job number to fg.
# nohup	to run a process directly in the server i.e without any terminal connection

# cat /dev/zero > /dev/null	to create a process
ctrl + z	To suspending a process
ctrl + c	to terminate a process
# jobs –l	to list background(bg) process for current terminal
	to bring background(bg) process to foreground(fg)

Difference between kill and kill -9:

Kill will generate a SIGTERM signal asking a process to kill itself gracefully i.e , free memory or take care of other child processes. Killing a process using kill will not have any side effects like unrelased memory because it was gracefully killed.

Kill -9 works similarly but it doesn't wait for the program to gracefully die. Kill -9 generates a SIGKILL signal which won't check the state of the process and kills the process immediately.

Zombie Process:

It derives from the common definition of zombie — an undead person.

On Unix and Unix-like computer operating systems, a zombie process or defunct process is a process that has completed execution (via the exit system call) but still has an entry in the process table: it is a process in the "Terminated state". This occurs for child processes, where the entry is still needed to allow the parent process to read its child's exit status: once the exit status is read via the wait system call, the zombie's entry is removed from the process table and it is said to be "reaped". A child process always first becomes a zombie before being removed from the resource table. In most cases, under normal system operation zombies are immediately waited on by their parent and then reaped by the system – processes that stay zombies for a long time are generally an error and cause a resource leak.

time for i in 1 2 3 4 5; do echo hello; sleep 2; done;

Nice & Renice:

Use to prioritize the process

Nice:

To assign a priority value for a new process.
Assigned by process name.

By default when a process starts, it gets the default priority of 0.

nice -n <nice value> <process-name>
-19 (highest) to 19 (lowest)
-19 -18 ...0....18 19

nice --4 "cat /dev/zero > /dev/null&" -- for +4 provide as -4, for -4 provide as --4

Renice: (prioritizing with PID, <username>, <groupname>)

Change priority value for already running process

nice -4 ls -ltr
renice -4 -p 3423 -- this will set the priority of process id no 3423 to -4, which will inturn increase its priority over others
renice 13 -u sarath -- this will set the priority of the process id 3564 to 13, and all the process owned by user "sarath" to the priority of 13
renice 14 -u sarath,satish -g custom -- this will set all process owned by "sarath","satish" and also the group "custom" to 14

```
[root@localhost Desktop]# cat /dev/zero > /dev/null&
[1] 5251

[root@localhost Desktop]# ps
PID TTY      TIME CMD
3059 pts/0   00:00:00 bash
5251 pts/0   00:00:10 cat
5255 pts/0   00:00:00 ps

[root@localhost Desktop]# kill 5251
[1]+  Terminated          cat /dev/zero > /dev/null
[root@localhost Desktop]# ps -j
PID  PGID   SID TTY      TIME CMD
3059  3059  3059 pts/0   00:00:00 bash
```

5263 5263 3059 pts/0 00:00:00 ps

KEY	LONG	DESCRIPTION
c	cmd	simple name of executable
C	pcpu	cpu utilization
f	flags	flags as in long format F field
g	pgrp	process group ID
G	tpgid	controlling tty process group ID
j	cutime	cumulative user time
J	cstime	cumulative system time
k	utime	user time
m	min_flt	number of minor page faults
M	maj_flt	number of major page faults
n	cmin_flt	cumulative minor page faults
N	cmaj_flt	cumulative major page faults
o	session	session ID
p	pid	process ID
P	ppid	parent process ID
r	rss	resident set size
R	resident	resident pages
s	size	memory size in kilobytes
S	share	amount of shared pages
t	tty	the device number of the controlling tty
T	start_time	time process was started
U	uid	user ID number
u	user	user name
v	vsize	total VM size in kB
y	priority	kernel scheduling priority

STAT - To describe the state of a process

D Uninterruptible sleep (usually IO)
R Running or runnable (on run queue)
S Interruptible sleep (waiting for an event to complete)
T Stopped, either by a job control signal or because it is being traced.
W paging (not valid since the 2.6.xx kernel)
X dead (should never be seen)
Z Defunct ("zombie") process, terminated but not reaped by its parent.

1st Row — top

This first line indicates in order:

current time (11:37:19)
uptime of the machine (up 1 day, 1:25)
users sessions logged in (3 users)
average load on the system (load average: 0.02, 0.12, 0.07) the 3 values refer to the last minute, five minutes and 15 minutes.

2nd Row – task
The second row gives the following information:

Processes running in totals (73 total)
Processes running (2 running)
Processes sleeping (71 sleeping)
Processes k (0 stopped)
Processes waiting to be stopped from the parent process (0 zombie)

3rd Row – cpu
The third line indicates how the cpu is used. If you sum up all the percentages the total will be 100% of the cpu. Let's see what these values indicate in order:
Percentage of the CPU for user processes (0.3%us)
Percentage of the CPU for system processes (0.0%sy)
Percentage of the CPU processes with priority upgrade nice (0.0%ni)
Percentage of the CPU not used (99,4%id)
Percentage of the CPU processes waiting for I/O operations(0.0%wa)
Percentage of the CPU serving hardware interrupts (0.3% hi — Hardware IRQ
Percentage of the CPU serving software interrupts (0.0% si — Software Interrupts
The amount of CPU 'stolen' from this virtual machine by the hypervisor for other tasks (such as running another virtual machine) this will be 0 on desktop and server without Virtual machine. (0.0%st — Steal Time)

4th and 5th Rows – memory usage
The fourth and fifth rows respectively indicate the use of physical memory (RAM) and swap. In this order: Total memory in use, free, buffers cached. On this topic you can also read the following article

6th Rows — Processes list
And as last thing ordered by CPU usage (as default) there are the processes currently in use. Let's see what information we can get in the different columns:

PID – l'ID of the process(4522)
USER – The user that is the owner of the process (root)

PR – priority of the process (15) , RT means a Real Time priority class – used for system processes)
NI – The “NICE” value of the process (0)
VIRT – virtual memory used by the process (132m)
RES – physical memory used from the process (14m)
SHR – shared memory of the process (3204)
S – indicates the status of the process: S=sleep R=running Z=zombie (S)
%CPU – This is the percentage of CPU used by this process (0.3)
%MEM – This is the percentage of RAM used by the process (0.7)
TIME+ –This is the total time of activity of this process (0:17.75)
COMMAND – And this is the name of the process (bb_monitor.pl)

Processes states that ps indicate:

D Uninterruptible sleep (usually IO)
R Running or runnable (on run queue)
S Interruptible sleep (waiting for an event to complete)
T Stopped, either by a job control signal or because it is being traced.
W paging (not valid since the 2.6.xx kernel)
X dead (should never be seen)
Z Defunct ("zombie") process, terminated but not reaped by its parent.
and the additional characters are:

< high-priority (not nice to other users)
N low-priority (nice to other users)
L has pages locked into memory (for real-time and custom IO)
s is a session leader
l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
+ is in the foreground process group

Press 'M' - to sort the process by memory usage.
Press 'P' - to sort the process by cpu usage.
Press 'N' - to sort the process by process id
Press 'T' - to sort the process by the running time.
Press 'R' - to sort the process by the reverse order
Press 'b' - to highlight sorted column
Press 'c' - to display the full command path
Press ‘O’ - to display parameters of top command
Press 'b' - to display the running process
Press 'c' - to display the sleeping process

top –u <username>
top –u admin
top –p 23658, 2365
c
z
shift + p
shift + m

ARCHIVING:

tar -- to create archive (tape archive)

C	Create
T	List
X	Extract
R	Append
--delete –force	Delete
V	Verbose
F	File

tar <options> <tar-file-name> <files/folders>

# tar -cvf bkp.tar folder1 folder2 file1 file2	to create the tar file
# tar -tvf bkp.tar	to view the tar file contents
# tar -xvf /root/omr/bkp1.tar	to extract the tar file.
# tar –rvf bkp.tar file3	to append a file in b file.
# tar delete -f tue.tar file3	to delete particular file from tar file
# tar -xvf apache-tomcat-7.0.69.tar.gz	we can also untar a compressed file directly
# tar -zcvf report.tar.gz /boot	we can also create tar file with compression

COMPRESSION:

gzip bkp.tar -- to compress a tar file
gunzip bkp.tar.gz -- to uncompress a zipped file.

bzip2 op.txt -- to compress .txt file
bunzip2 op.txt.bz -- to uncompress .txt file

FIND

To search the files and directories with the required options in whole hard disk.
User has to have rx permission on searching directory.

find <path> <options>..... <actions>

using –name	
# find / -name Tecmint	finds both file and directory
# find / -name *.sh -o -name *.log	find two file names in a same time.
using –type	
# find / -type d -name Tecmint	finds directory
# find / -type f -name Tecmint	finds files
# find / -type f -iname Tecmint	remove case sensitive
# find . -type f -not -name "*.d"	find filename excluding *.d files
# find / -type f -empty	find empty files
# find / -type d -empty	find empty directories.
# find / -type f –links +1	
# find / -type l	find link files alone.
# find / -type b	find block files alone.
using –size	
# find / -size +50M -size -100M	
# find / -size +100M	
using –mtime	
+5 -- **1 2 3 4 5 6 7 8 9 10 11** 12 13 14 15 16 **-5** -- 1 2 3 4 5 6 7 8 9 10 11 **12 13 14 15 16** **-2** -- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 **15 16** **2** -- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 **15** 16 **3** -- 1 2 3 4 5 6 7 8 9 10 11 12 13 **14** 15 16	
# find /dev -mtime +5	(modified time) before 5days (+5 -- before 6th)
# find /var/spool -mtime -5	within 5days from the current date
# find /var/spool -mtime 5	with exact 5th day from the current date
using –mmin	
# find / -mmin 120	
# find / -mmin +200	
# find / -mmin -150	
using –perm	
# find / -type f -perm 666	
# find / -type f -perm 642	
# find / -perm +2000	to find SGID files - Sticky bit group id
# find / -perm +4000	to find SUID files - Sticky bit user id
using –user	
# find /home -group ftpusers	
# find /data/project -group ftpusers -name "*.c"	
using –group	
# find /data/project -group ftpusers -iname "*.c"	
# find directory-location -user {owner} -name {file-name}	
Execlude directories	
# find / -type f ! -path "/proc/*su" -size +100M	Finding files sizes in whole hard disk which has

	more than 100mb size by excluding /proc directory
Action:	
: -exec "action" {} \;	
# find . -type f -empty -exec /bin/rm -rf {} \;	to delete empty files in PWD
# find . size +50M -exec cp {} ./Desktop \;	to cp files having size more than 50MB to Desktop folder
# find / -type f -exec "grep -H 'gulshan'" {} \;	
# find . -type f -empty -exec rm -rf {} \; -print	find a string in which file in exists in whole hd, H print filenames
# find / -type f -name '*.tar.gz' -mtime +5 -exec rm {} \;	

ALIAS:

Instructs the shell to replace one string with another when executing commands.
Aliases are used to customize the shell session interface.
Using alias, frequently-used commands can be invoked using a different, preferred term; and complex or commonly-used options can be used as the defaults for a given command.
Aliases persist for the current session. They can be loaded at login time by modifying the shell's .rc file. The invocation and usage of alias differs depending on the shell;

# alias a ='cat /etc/passwd'	create a temporary alias as "a"
# a	execute the command as "a"
# alias b='cat /etc/group'	create a temporary alias as "b"
# b	execute the command as "b"
# alias dir='cd /d1/d2/d3/d4/d5/d6'	add this entry in .bashrc to make alias permanent
# dir	execute the command as "dir"

SLEEP:

The sleep command is used to delay for a specified amount of time.
The sleep command pauses for an amount of time defined by NUMBER.

SUFFIX may be "s" for seconds (the default), "m" for minutes, "h" for hours, or "d" for days.

```
[root@localhost ~]# cat aa
date
    echo "  perl    "
    sleep 2 #wait for 2seconds
    echo
```

```
date
      echo "  python  "
      sleep 1m #wait for 1minute
      echo
date
      echo "  ruby    "
      sleep 2m 30s
      echo
date
      echo " splunk   "
      sleep 2m 30s
      echo
date
```

SCHEDULERS:

To automate a job.
Run a job in a specified timing.

AT, CRONTAB

At :

One time job scheduler.
Can't be edited.

Commands used with AT:

```
# at                          --      execute commands at specified time.
# atq                         --      lists the pending jobs of users.
# at -c <job.no>              --      to view the content of a particular job
# cat /var/spool/mail/root --         to view output of the job
# atrm   <jobnumber>    --            delete jobs by their job number.
```

1. Schedule first job using at command

Below example will schedule “ls -l” command to be executed on next 9:00 AM once.

```
# at 9:00 AM
at> ls -l
```

at> ctrl+d -- to exit from at prompt.
job 3 at 2013-03-23 09:00

2. List the scheduled jobs using atq

When we list jobs by root account using atq , it shows all users jobs in result. But if we execute it from non root account, it will show only that users jobs.

```
# atq
3      2013-03-23 09:00 a root
5      2013-03-23 10:00 a rahul
1      2013-03-23 12:00 a root
```

Fileds description:

First field	: job id
Second field	: Job execution date
third field	: Job execution time
Last field	: User name, under which job is scheduled.

3. Remove scheduled job using atrm

Yo can remove any at job using atrm using there job id.

```
# atrm 3
# atq
5      2013-03-23 10:00 a rahul
1      2013-03-23 12:00 a root
```

4. Check the content of scheduled at job

atq command only shows the list of jobs but if you want to check what script/commands are scheduled with that task, below example will helps you.

```
# at -c 5
```

In above example 5 is the job id.
Examples of at Command:

Example 1: Schedule task at comming 10:00 AM.

```
# at 10:00 AM
```

Example 2: Schedule task at 10:00 AM on coming Sunday.
at 10:00 AM Sun

Example 3: Schedule task at 10:00 AM on coming 25'th July.
at 10:00 AM July 25

Example 4: Schedule task at 10:00 AM on comming 22'nd June 2015.
at 10:00 AM 6/22/2015

Example 5: Schedule task at 10:00 AM on same date at next month.
at 10:00 AM next month

Example 6: Schedule task at 10:00 AM tomorrow.
at 10:00 AM tomorrow

Example 8: Schedule task to execute just after 1 hour.
at now + 1 hour

Example 9: Schedule task to execute just after 30 minutes.
at now + 30 minutes

Example 10: Schedule task to execute just after 1 and 2 weeks.
at now + 1 week
at now + 2 weeks

Example 11: Schedule task to execute just after 1 and 2 years.
at now + 1 year
at now + 2 years

Example 12: Schedule task to execute at mid night.
at midnight

root user can also remove other users AT jobs.

```
[root@localhost ~]# atq
4	2015-09-30 20:00 a root
5	2015-09-30 21:00 a training

[root@localhost ~]# atrm 5
4	2015-09-30 20:00 a root
5	2015-09-30 21:00 a training
```

Crontab:

Job Scheduler to automate the jobs.
Its user specific.
Minimum time cron can assign a job in a minute.

Basic syntax

* * * * * cmd
* mins 0-59
* hr 0-23
* date of month 1-31
* month of year 1-12
* day of week 0,7-6 sun-sat

# crontab -e	to edit crontab file
# crontab -l	to list the contents crontab file
# crontab –l –u user1	to list other user's cron settings

30 08 10 6 * /root/backup.sh	10th june 8.30am
00 12,21 * * * /root/monitoring.sh 1> mon.op 2>mon.err	twice in a day, 12pm 9pm
00 9,12,15,18 * * 1-5 cmd	only in weekdays b/w 9am - 6pm for every 3hr
*/2 * * * *	for every 2mins
*/10 * * * *	for every 10hrs
0 */1 * * *	for every 1hr.

Need not to reboot the pc to make the changes in effect, bcoz after exiting crontab, by automatically it will get refreshed.

```
[root@client ibm]# crontab -l

#* * * * * echo hi > /dev/pts/3

42 14 * * * "cp /ibm/file1 /ibm/file2" > op.log 2>er.log

# this job is for create directory
52 14 * * * /ibm/script.sh

#this is report auditing
57 14 * * * /ibm/script.sh  >op.log 2>op.err
```

SHELL SCRIPTING

Shell

It's a user interface to communicate with OS for executing commands.
User - > shell - > kernel -> hardware
Types –
sh, bash (bourne again shell),
ksh(korn shell),
zsh,
C (csh, tsh)

ps
echo $SHELL

Shell Script

Its program file with many commands are placed in, to execute one by one.
It is useful to run many task by accepting input from user and provide output.
Both interactive and non-interactive shell script can be prepared.
Everyday task can be automated by shell script.

# ps	prints current shell
# echo $SHELL	prints current shell
# printenv SHELL	prints current shell
# echo $BASH_VERSION	prints bash version

Bash Versions:

3.2.25(1) - release EL5
4.1.20(1) - release EL6
4.2.20(1) - release EL7

Variables:

variable in capital letters are **SYSTEM DEFINED** variables. echo $SHELL
variable in small letters are **USER DEFINED** variables. echo $a

Type of Variables:

1 Local variables
2 Environmental or Global variables

3 Shell variables

Local variables:
Variables used within the shell.
Variables not used for all(parent and child) the shells running in the system.

a=apple -– assigning a variable
echo $a -- calling a variable

b=batch -- this type of variable is called as **scalar** variable
echo $b

files=(/etc/passwd /etc/shadow /etc/group) -- this type of variable is called as **array** variable
echo ${files[0]}
echo ${files[1]}
echo ${files[2]}

Environmental or Global variables:
An environment variable is available to any child process of the shell.

echo $PATH
echo $HOME
echo $PS1
echo $USER
echo $SHELL

# export <variable-name>	to set the variable available in all shells(parent, child)
# export –n <variable-name>	to remove variable temporarily from export list.
# printenv	prints all env variables
# env	prints all env variables

To make variable permanent.
vi ~/.bash_profile

a1=linux.com
export a1=linux.com

If variable settings needs to be global (i.e ALL users to have them) modify system profiles such as /etc/profile.

PATH:

- The PATH is an environment variable.
- It is a colon (:) delimited list of directories that your shell searches through when you enter a command.
- All executables are kept in different directories on the Linux and Unix like operating systems.

To check current values of PATH variable:

```
# echo $PATH

/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/root/bin:/oracle
```

To add directories permanently in PATH variable:

```
# cat   ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi;

# User specific environment and startup programs
export PATH=$PATH:$HOME/bin:/oracle
```

Shell variables:

```
# vi firstscript.sh

#!/bin/bash
echo "File Name                          : $0"
echo "First Argument                     : $1"
echo "Second Argument                    : $2"
echo "Quoted Values                      : $*"
echo "Total Number of Arguments          : $#"
echo “PID of current script                : $$”
echo “PID of last background process       : $!”
echo “Exit status of this script           : $?”

[root@localhost ~]# sh firstscript.sh java linux
File Name                  : firstscript.sh
First Argument             : java
Second Argument            : linux
```

Quoted Values : java linux
Total Number of Arguments : 2
PID of current script : 2545
PID of last background process :
Exit status of this script : 0

#echo $?
0 success
#echo $?
1 error

Prompt String:

- PS1 (Prompt String 1) is one of the prompts available in Linux/Unix.
- When you try to login to any machine, you have to enter user name and password.
- Once you are done with this you are presented with some info like who logged in, on what machine he logged in, what is his present working directory and if the logged in user is a super user or a normal user.
- This is done by using PS1 prompt which is a inbuilt shell variable.

```
# export PS1=”[\u@\h \w]\$ “
# export PS1=”[\h \d \t \u:\w]\$ "
# export PS1="\u@\h[ \$(date +%k:%M) ]\w:"
# export PS1=”\[\e[1;32m\][\u@\h \W]\$\[\e[0m\] “
# export PS1=”\s-\v\$”
```

If you want to show this prompt show after you reboot, add the same variable to your ~/.bash_profile or ~/.profile with respect to your currect shell.

.bash_profile -- to add the variable
.bashrc -- contains system environment settings

 # cat /etc/profile
cat /etc/bashrc

```
i=idea
echo "hi .........welcome..........$LOGNAME"
a=`date +%r | tr ' ' : | awk -F: '{print $1":"$2" "$4}'`
export PS1="\u@\h[\$(date +%d-%h) \$a]\w:"
```

~/.bash_profile is the place to put stuff that applies to your whole session, such as programs that you want to start when you log in (but not graphical programs, they go into a different file), and environment variable definitions.

~/.bashrc is the place to put stuff that applies only to bash itself, such as alias and function definitions, shell options, and prompt settings. (You could also put key bindings there, but for bash they normally go into ~/.inputrc.)

Understanding parent shell, child shell:

```
# ps -l
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0   2244   2240  0  80   0 - 27062 wait   pts/0    00:00:00 bash
4 R     0   2368   2244  0  80   0 - 26467 -      pts/0    00:00:00 ps

#  bash
#  ps -l
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0   2244   2240  0  80   0 - 27062 wait   pts/0    00:00:00 bash
0 S     0   2369   2244  0  80   0 - 27062 wait   pts/0    00:00:00 bash
4 R     0   2379   2369  0  80   0 - 26467 -      pts/0    00:00:00 ps

#  bash
#  ps -l
F S   UID    PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0   2244   2240  0  80   0 - 27062 wait   pts/0    00:00:00 bash
0 S     0   2369   2244  0  80   0 - 27062 wait   pts/0    00:00:00 bash
0 S     0   2380   2369  0  80   0 - 27062 wait   pts/0    00:00:00 bash
4 R     0   2389   2380  0  80   0 - 26467 -      pts/0    00:00:00 ps
```

Understanding parent shell, child shell with export command:

Export command is used to export a variable or function to the environment of all the child processes running in the current .

Case1:

```
# a=linux.com
# echo $a
linux.com
#bash              -- getting into subshell
#echo $a          --- this will print empty line
```

Case2:

```
# a=linux.com
# echo $a
linux.com
# export a        --- now variable "a" will be available across the all sub shells.
# bash
# echo $a         --- this will print value of "a"
 linux.com
```

Different ways of running the script

# ./1.sh	using it's relative path	It ll fork a sub shell
# /home/user1/1.sh	using its absolute path	It ll fork a sub shell
# sh 1.sh	by specifying the interpreter(sh)	It ll fork a sub shell
# bash 1.sh	by specifying the interpreter(bash)	It ll fork a sub shell

```
echo " enter your-name "
read v1
echo $v1
echo ============
read -p " enter your-name " v2
echo $v2
echo ============
read -s -p "Enter Password : " v3
echo "Your password - $v3"
```

COMMENTING :

```
single line # echo "hi"
multiple lines
:<< 'tag'
        echo "hi"
        echo "hello"
        echo "linux"
tag
        echo "windows"
```

QUOTES:

;	command separator	#date; cal; who; uname -a; id

"" double quote	to make the statement as a single string	#echo "this script is used for monitoring"
\ backslash	to suppress the spl meaning of metacharacters.	#echo "this script is used for monitoring which is bought for \$5"
``back quotes	to make the statement as a command in a string	#echo "this script is used for monitoring which is bought for $5" on `date`"
‘’ single quotes	to make the statement suppressing effect of spl characters	# echo ‘Am $x $y $z’
-e	enables interpretation of backlash	#echo -e "this script is used \n for monitoring"
\n	creating new line	#echo -e "this script is used \n for monitoring"
\t	creating tab space	#echo -e "\t this script is used \n for monitoring"
()	Parentheses to evaluate first	#echo $(a)

DATE

```
# date +"%FORMAT"
# date +"%FORMAT%FORMAT"
# date +"%FORMAT-%FORMAT"
# date +%d_%m_%y_%H_%S_%r_%R_%h
# date -s "22:09"
# date –d 20170101 +%A
```

Arithmetic Substitution:

/ Division
* Multiplication
+ Addition
- Subtraction
() Parentheses to evaluate first
% Reminder

# expr 6 + 3	# $((6:3))	9	Addition
# expr 6 – 3	# $((6-3))	3	Subtraction
# expr 3 – 6	# $((-6+3))	-3	Subtraction
# expr 6 * 3	# $((6*3))	18	Multiply
# expr 6 / 4	# $((6/4))	1	Quotient
# expr 6 % 4	# $((6%4))	2	Remainder/Modulus

Relational Operators	
-eq	Equal to
-lt	Less than

-le	Less than or Equal to
-gt	Greater than or Equal to
-ge	Greater than or Equal to
-ne	Not equal to
File related tests	
-f file	True if file exists and is a regular file.
-r file	True if file exists and is readable.
-w file	True if file exists and is writable.
-x file	True if file exists and is executable.
-d file	True if file exists and is a directory.
-s file	True if file exists and has a size greater than zero
-e file	True if file exist
String tests	
-n str	True if string str is not a null string.
-z str	True if string str is a null string.
str1 = str2	True if both strings are equal.
Str	True if string str is assigned a value and is not null.
str1 != str2	True if both strings are unequal.
Conditional Operators	
[ ] Test also permits the checking of more than one expression in the same line	
-a, &&	Performs the AND function
-o, ||	Performs the OR function

Control Statements:

1. Basic syntax and logic
2. Clear requirement.

if <condition> ; then <action> fi ;
if <condition> ; then <action> else <action> fi;
if <condition> ; then <action> elif <condition> ; then <action> else then <action> fi;

if ..fi
if ..else..fi
if ..elif..else..fi

if then using [ ] test condition
eg1

```
a=1
b=2

if [ $a = 1 ];
then c=`expr $a + $b`;
echo "$c";
fi;
```

eg2:

```
echo “Enter the value for A”
read A # to get input from user
echo “Enter the value for B”
read B

C=`expr $A + $B`;
echo "Addition of $A and $B is $c";
```

if then else

```
if  ls  ff ;
then
echo "file ff has been  found"
else
echo "file ff not found"
fi;
```

if then else using [] test condition

```
if [ -f ff ];
then echo "file ff found"
else echo "file ff not found"
```

if then elif

```
if ls ff ;then
echo "file ff found";
elif ls ss;then
echo "file ss found"
else echo "file ff not found"
fi;
```

```
if [ -f ff ];then
echo "file ff found";
elif [ -f ss ];then
echo "file ss found"
elif [ -f ram ]; then
echo "file ram was found"
else echo "None of the files are  found"
fi;
```

AND -a &&

1 1 = 1
1 0 = 0
0 1 = 0
0 0 = 0

[$a –a $b] , [$a] && [$b]

OR -o ||

1 1 = 1
1 0 = 1
0 1 = 1
0 0 = 0

```
if [ -f sony -a -f onida ]
then touch lg
echo "file lg is created"
echo "`ls -l lg`"
else
echo "either sony or onida is not available to create a file lg"
fi;
```

AND operator (-a)

```
if [ -f aa -a -f bb -a -f cc ];
then echo "file aa bb cc are found"
elif  [ -f aa -a -f bb ]
then echo "file aa bb are found"PS1
```

```
elif [ -f aa ];
then echo "file aa is found";
fi;
```

eg: (&&)

```
[root@localhost ~]# cat 10.sh
#!/bin/bash

if  [ -f nov ] && [ -f dec ]
      then touch jan
else echo "file jan is not created"
fi;
```

OR operator (||)

Eg: (-o)

```
if  [ -f nov  -o  -f dec ]
      then touch jan
else echo "file jan is not created"
fi;
```

Eg: (||)

```
cd /tmp/arun/ || ls -ltr condition;
touch /tmp/sax;
```

STRING COMPARE

```
if  [ $1=male ];
then echo "given gender is correct"
fi;
```

! negation

```
if [ ! -f fb] ;
then touch fb
fi;
```

case

```
case pattern in
pattern1) action1 ;;
pattern2) action2 ;;
pattern3) action3 ;;
pattern4) action4 ;;
...
patternN) actionN;;
*) action;;
esac

read a
case $a in
prabal) echo "request for interrupt interface" ;;
ravi) echo "always blush" ;;
gowri) echo "itna paise denge";;
manisha) echo "punjab is like canada";;
esac
```

Which is better and easier , IF THEN or CASE statement ??

Loops:

for loop
while loop
nested while loop
until loop
infinite while

for:
syntax

```
for var_name in var_values
do
        action
done

for var in 1 2 3
do
echo "Welcome $var"
```

```
done;

for var in {1..10}
do
echo "Welcome $var"
done;
```

while:

```
$var=x




while    <condition>
do
         <action>
done;

a=1;
while [ $a -lt 10 ];
do
a = `expr $a +1'
echo $a
done

a=2;
b=3;

while [ $a = 2 ] ; # condition checking
do
c=`expr $a + $b`;
echo $c;
#   a=`expr $a + 1`;
done
```

Nested while loop:

```
read a
while [ $a -lt 10 ];  # 1st loop
```

```
do
        b=$a;
        while [ $b -gt 0 ];   # 2nd loop
                do
                        echo -n "$b"  #     -n     do not output the trailing newline
                        b=`expr $b - 1`;
                done
        echo
        a=`expr $a + 1`
done;

10
210
3210
43210
543210
6543210
76543210
876543210
```

infinite while loop

```
while :
do
                echo "press ctrl+c to stop"
done ;
```

While loop for arithmetic operation:

until loop

This loop would continue forever because a is alway greater than or equal to 10 and it would never become less than 10.

```
a=10

until [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
```

```
done
```

How to compare two dates:

```
echo enter the 1st date in format yyyy-mm-dd
read a;
a1=$(date -d $a +"%Y%m%d");# a1 = 20171202

echo enter the 2nd date in format yyyy-mm-dd
read b;
b1=$(date -d $b +"%Y%m%d");# b1 = 20161202

if [ "$a1" -ge "$b1" ];
then  echo $a ;
else echo $b;
fi;
```

FUNCTION

Function is series of instructions/commands.
Function performs particular activity in shell.
To define function use following

Syntax:

```
function-name ()
{
        <action1>
        <action2>
..
..
        <actionn>

        return
};
```

Simple function:

```
[root@node1 ~]# cat -n f1.sh
    # define your functions
    f1 ()
    {
```

```
        echo "hi"
        echo "hello"
    }

    # invoke your functions
    f1
```

[root@node5 ~]#bash f1.sh – to run a function.

```
[root@node1 ~]# cat -n f2.sh
    today()
    {
    echo "Today is `date +%A`, `date +"%d %h %Y"`, `date +%r`"
    }

    today;
```

Function with arguments:

```
#cat file1
f2 ()
{
        echo hi hello $1 $2
}

f2  $1 $2
```

```
#bash file1 arun anu
hi hello arun anu
```

To call a function at login page, add it to .bashrc file

```
[root@node5 ~]#tail -20 .bashrc

f1 ()
{
        echo "hi"
        echo "hello"
}
f1;
```

Function for adding numbers

```
#!/bin/bash
add(){
   sum=$(($1+$2))
   return $sum
}

read -p  "Enter an integer: " int1
read -p  "Enter an integer: " int2
add $int1 $int2
echo "Output is: " $?
```

Which input is greater

```
max_two ( ) {
      if [ "$1" -eq "$2" ] ; then
            echo Equal
      elif [ "$1" -gt "$2" ] ; then
            echo $1
      else
            echo $2
      fi
      }
max_two $1 $2
```

Function with while loop condition

```
  echo "This is a funky function."

fun ()
{   i=0
  REPEATS=5

  echo
  echo "And now the fun really begins."
  echo

  while [ $i -lt $REPEATS ]
```

```
  do
            sleep 5    # Wait for 5  second!

    echo "<----------FUNCTIONS----->"
    echo "<------------ARE------------->"
    echo "<------------FUN------------>"
    echo
    let "i+=1"
  done
};

  # Now, call the functions.

fun
```

Funtion for taking backups

```
#!/bin/bash

LOGFILE="/root/backupscript.log"
echo "Starting backups for `date`" > "$LOGFILE"

report_bkp()
{
      DIR="/report/"
      TAR="report.tar"
      BZIP="$TAR.bz2"

      cd "$DIR"
      echo "Archiving files under /report" >> "$LOGFILE"
      tar -cvf "$TAR" *.xml *.png *.xls
      echo "Compressing $TAR..." >> "$LOGFILE"
      bzip2 "$TAR"
      echo "backup of /report is taken " >> "$LOGFILE"
};
```

```
user_data_bkp()
{
	TAR="users.tar"
	BZIP="$TAR.bz2"
	FILES="/home"

	echo "Archiving files under /home" >> "$LOGFILE"
	tar -cvf "$TAR" "$FILES"
	echo "Compressing $TAR..." >> "$LOGFILE"
	bzip2 "$TAR"
		echo "Backup of /home is taken " >> "$LOGFILE"
}

DAY=`date +%w`

if [ "$DAY" = "0" ]; then
  echo "It is `date +%A`, only backing up report." >> "$LOGFILE"
	report_bkp
 else  echo "It is `date +%A`, only backing up /home dir." >> "$LOGFILE"
	user_data_bkp
fi;

echo -e "Backup `date` is SUCCESS "  >>  "$LOGFILE"
```

Interactive script 1:

```
[root@client ~]# cat -n interactive

while :
do
clear
		echo "###################"
		echo "  MAIN MENU              "
		echo "###################"
		echo "[1] show free memory info";
		echo "[2] show current users connected to the server";
		echo "[3] show cpu info";
		echo "[4] exit";

read a
```

```
        case $a in
                1) echo -e "`free -m` \n   press enter key"; read  ;;
                2) echo -e "`who am i` \n press enter key" ;read     ;;
                3) echo -e "`cat /proc/cpuinfo` \n press enter key" ; read  ;;
                4) exit ;;
        esac;
done
```

Try executing below shell scripts to understand how to run a script in its parent shell without forking a subshell.	
# cat 1.sh echo "This PID is from 1.sh $$"; ps –l; # cat 2.sh echo "This PID is from 2.sh $$"; sh /root/1.sh; ps -l; # sh 2.sh	invoking shell scirpt from another shell script
# cat 1.sh echo ""This PID is from 1.sh $$""; ps –l; # cat 2.sh echo ""This PID is from 2.sh $$""; source /root/1.sh ; ps -l; # sh 2.sh	this will run the script with same pid of 2.sh

# cat 3.sh 	
f1()
{
echo hi;
};
 	
# cat 4.sh 	
sh /root/1.sh
f1;
# sh 4.sh 	
	you will get error for this script.
	
	F1 command not found:

# cat 3.sh 	
f1()
{
echo hi;
};
# cat 4.sh 	
source /root/1.sh
f1;
 	
# sh 4.sh 	
	calling a function from a same shell

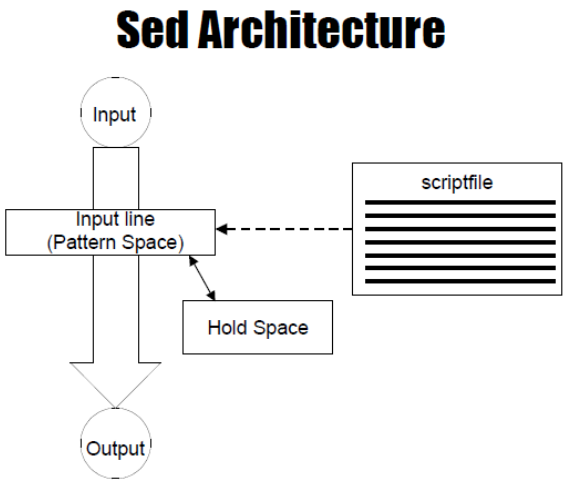
# cat 1.sh 	
f1()
{
echo hi this is f1
};
 	
f2()
{
echo hi this is f2
};
 	
f3()
{
echo hi this is f3
};
 	
export -f f1 f2 f3
sh /root/2.sh
 	
# cat 2.sh 	
 	
f1
f3
 	
# sh 1.sh 	
	1. Defining 3 functions in 1.sh file
	2. Calling only f1, f3 functions from 2.sh
	3. Execute 1.sh to check the output

Two more ways to execute a shell script:

# . 1.sh	execute the script in the current shell without forking a sub shell	It wont fork a sub shell
# source ./1.sh	source command is synonym for the . (dot)	It wont fork a sub shell

SED :
Stream oriented
Non interactive
Text Editor

The name sed is an abbreviation for stream editor and the utility derives many of its commands from the ed line-editor (ed was the first UNIX text editor). This allows you to edit multiple file or to perform common editing operations without ever having to open vi or emacs.



Syntax:
-[addr1] [addr2] [addr..] [command][arguments]

Address can be either a line number or a pattern, enclosed in slashes /PATTERN/
A pattern is regular expressions (grep)
If no pattern is specified, the command will be applied to all lines in a file.

Command is a single letter

[addr1] [addr2] [addr..] d

 # sed -n 'ADDRESS'p filename

 # sed -n '/PATTERN/p' filename

Print	
# sed -n 1p songs.txt	print 1st line from the file
# sed -n '$'p songs.txt	print the last line
# sed -n '1p;4p' marks	print 1st and 4th line
# sed -n '4,8'p songs.txt	print from 1st 4th line in a range
# sed -n '4,$'p songs.txt	print from 4th line last line
# sed -n '1~3'p songs.txt	print from 1st line and jumping 4th, 7th , 10th ..Nth rd line
# sed -n '3~2'p songs.txt	print from 3rd line and leaving one after
# sed -n -e '1,5p' -e '10,15p'	print lines from 1 to 5 and 10 to 15
# sed -n '/John/'p songs.txt	print lines containing the word "John"
# sed -n '/John/'Ip songs.txt	print lines containing the word without case sensitive "John"
# sed -n '/ntp/p;/gdm/p' passwd	print lines containing the words 'ntp' and 'gdm'
# sed -n '/ntp/,/gdm/p' passwd	print lines containing the words 'ntp' and 'gdm'
# sed -n '/X/!p' file	print lines which does not contain 'X':
# sed -n '/[@!]/p' file	print lines which contain the character 'u' or 'x'
# sed -n '/bash$/p' file	print lines which end with 'bash'
# sed -n '/[xX]$/p' file	print lines which end with 'x' or 'X'
# sed -n '/^[AL]/p' file	print lines beginning with either 'A' or 'L':
# sed -n '/^root\\|bash$/p'	print lines starting with 'root' and ending with 'bash'
# sed -n '/^root.*bash$/p'	print lines starting with 'root' and same line ending with 'bash'
Delete	
# sed 6d /etc/passwd	deletes line 6, (-e expression)
# sed '1d;4d' marks	deletes 1st and 4th line
# sed '$d' costumerA	delete last line in file
# sed '/^$/d' costumerA	delete the empty lines in file costumerA
# sed '1,10d' file	deletes lines 1 through 10 in range wise
# sed '2,4!d' file	delete lines other than the specified range
# sed '1d;$d' file	delete 1st and last line from the file
# sed -e /^ya*/,/[0-9]$/d /etc/passwd	deletes from the first line that begins with ya or ends with a digit
# sed '/x$/d' file	delete all lines ending with a character 'x'
Substitue and Replace	
# sed s/<find>/<replace>/g	(s substitute, g global), replace the "word" day by "night".
# sed s/day/night/g	(s search, g global), replace the "word" day by "night".
# echo "/root/abc" | sed 's/\/root\/abc/\/s1\/s2/' / as delimiter	replacing a string with /
# sed -e 's/ \/root\/abc / \/hemanth\/ankit /g' sed.txt	replacing a string with /
# sed -e 's/ \s\+/ /g' p.log	replace all the tabs/spaces in between the records with just a single space, + repeats the previous item more times.
# sed 's/^/@@/g' passwd	insert @@ at starting of every line
# sed -i '1i\hi there\' passwd	add this pattern to first line

# sed -i '1c\hi there\' passwd	change this pattern in first line with old one.
# cat 1.sed [root@localhost ~]# cat 1.sed 1,5p 25,28p /bash$/p # sed -n –f 1.sed input-filename	Using sed script file, when we have many instuctions, so it can be called from a file.

AWK:

- Named after Aho, Weinberger, Kernigham.
- It's a powerful programming language.
- Can access, transform and format individual fields in records.
- It get inputs from files, redirection and pipes
- Convenient numeric processing.
- Convenient way of accessing fields from a lines.
- Flexible printing with printf function.
- Builtin arithmetic and string functions.
- C-like syntax.

#awk –version

Syntax

An awk program consists of:
– An optional BEGIN segment
    • For processing to execute prior to reading input
– *pattern - action pairs*
    • *Processing for input data*
    • *For each pattern matched, the corresponding action is taken*
– An optional END segment
    • Processing after end of input data

BEGIN {action}
    pattern {action}
    pattern {action}
    .
    .
    pattern {action}
END {action};

BEGIN and END are special patterns and optional.
They are not used to match input records.
Rather, they are used for supplying start-up information to your awk script.
A BEGIN rule is executed, once, before the first input record has been read.

An END rule is executed, once, after all the input has been read.

AWK Variables List:

$0	Contains content of the full records.
$1..$n	Holds contents of individual fields in the current record.
NF	Contains number of fields in the input record.
NR	Contains number of record processed so far.
FS	Holds the field separator. Default is space or tab.
OFS	Holds the output field separator.
RS	Holds the input record separator. Default is a new line.
ORS	Holds the output record separator.
FILENAME	Holds the input file name.

Operators	
Unary operators	Operator which accepts single operand is called unary operator.
Binary operators	Operator which accepts more than one operand is called binary operator.
Unary Operator	
+	Positivate the number
–	Negate the number
++	Auto Increment
--	Auto Decrement
Binary operators	
Arithmetic Opertors	
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Modulo Division/Reminder
(space)	String Concatenation
Assignment Operators	
=	Assignment
+=	Shortcut addition assignment
-=	Shortcut subtraction assignment
*=	Shortcut multiplication assignment
/=	Shortcut division assignment
%=	Shortcut modulo division assignment
Conditional Operators	
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to
==	Is equal to

!=	Is not equal to
&&	Both the conditional expression should be true
||	Any one of the conditional expression should be true
Regular Expression Operator	
~	Match operator
!~	No Match operator

```
#cat bss
aardvark 555-5553 1200/300 B
alpo-net 555-3412 2400/1200/300 A
barfly 555-7685 1200/300 A
bites 555-1675 2400/1200/300 A
camelot 555-0542 300 C
core 555-2912 1200/300 C
fooey 555-1234 2400/1200/300 B
foot 555-6699 1200/300 B
macfoo 555-6480 1200/300 A
sdace 555-3430 2400/1200/300 A
sabafoo 555-2127 1200/300 C

# cat inv
Jan 13 25 15 115
Feb 15 32 24 226
Mar 15 24 34 228
Apr 31 52 63 420
May 16 34 29 208
Jun 31 42 75 492
Jul 24 34 67 436
Aug 15 34 47 316
Sep 13 55 37 277
Oct 29 54 68 525
Nov 20 87 82 577
Dec 17 35 61 401
Jan 21 36 64 620
Feb 26 58 80 652
Mar 24 75 70 495
Apr 21 70 74 514

# cat students
Jones 2143 78 84 77
```

Gondrol 2321 56 58 45
RinRao 2122 38 37
Edwin 2537 87 97 95
Dayan 2415 30 47

Printing	
# awk '{}'	single quotes around makes shell doesn't interpret any awk characters as special shell characters
# awk '{action}' filename	
# awk '{print}' filename	print contents of whole file
# awk '/pattern/' filename	
# awk '{print $0}' filename	print contents of whole file
# awk '{print $1}' filename	print 1st column of file
# awk '{print $1 $2 $3}' filename	print 1st 2nd 3rd columns of file
# awk '{print $1 " " $2 " "$3}' filename	print 1st 2nd 3rd columns of file with space as delimiter
# awk '{print $1 " ," $2 " ,"$3}' filename	concatenates using ","
# awk '/foo/ {print}' bss	to search a pattern
# awk '/core/ {print} /fooey/ {print}' bss	to search a pattern more than one pattern
# uptime | awk '{print $4,$5}'	
# awk '$2 ~ /12/ {print} ' bss	
# awk '$1 ~ /foo/ {print} ' bss	find a pattern in respective column
# awk '{$5="exam"; print}' students	to replace a column with particular value
# awk '$1 == "foo" { print $2 }' bss	
# awk '$1 ~ "foo" { print $2 }' bss	
# awk '/2400/ && /foo/' bss	using conditional parameter AND
# awk '/2400/ || /foo/' bss	using conditional parameter OR
# awk ' ! /foo/' bss	
# awk ' $1 == "core", $2 == "B" ' bss	
# awk ' $1 == "core", $4 == "B" ' bss	
# awk '/45/ {print "70:", $0} ; /95/ {print "95:", $0 }' students	print with matching strings
# awk '{ print "Field number one: " $1 }' bss	string concatenation

Functions	
# awk ' tolower($0) ~ /dayan/ {print}' students	using inbuild tolower function, to search a pattern without case-sensitive
# awk '{ print "The square root of", $2, "is", sqrt ($2) }' inv	using inbuild sqrt function

# echo 'productiondata_12' | awk 'sub("data","org",$0)'	sub() function in awk replaces part of strings
# awk '{print length}' emp.lst	
Field separator	
# awk –F : '{print $1 " " $2 " " $3}' /etc/passwd	print 1st 2nd 3rd columns of file when input file has delimiter "|"
# awk -F : '{print $2}' 0077wd	
# awk -F: '$2 == "x"' /etc/passwd	
# awk -F ':' '$3==$4' /etc/passwd	Print every line which has the same USER ID and GROUP ID
# awk -F ':' '$3>=100 && $NF ~ /\/bin\/bash/' /etc/passwd	Print user details who has USER ID greater than or equal to 100 and who has to use /bin/sh
# awk -F ':' '$5 == "" ' /etc/passwd	Print user details who doesn't have the comments in /etc/passwd file
# awk -F ':' '$3 > maxuid { maxuid=$3; maxline=$0 }; { print maxuid, maxline }' /etc/passwd	Find the user details who is having the highest USER ID
# awk -F ':' '$3 > maxuid { maxuid=$3; maxline=$0 }; END { print maxuid, maxline }' /etc/passwd	Find the user details who is having the highest USER ID
Variables	
# awk -F: '{print $1 " " $2 " " NF}' /etc/passwd	print 1st 2nd total_no_of_columns of file
# awk '{print NF, $0}' /etc/passwd	print no.of fields in each record
# awk 'NF==4' /etc/passwd	print records where no.of fields is 4
# awk '{print NR,$0}' /etc/passwd	print record numbers with contents of file
# awk 'NR==4' /etc/passwd	print only 4th record of file
# awk 'NR % 2 == 0' /etc/passwd	prints the even-numbered lines
# awk '{print NR, $0}' file1 file2	prints records numbers with two files as input
# awk '{print NR, FNR ,FILENAME , $0}' file1 file2	prints with record number, file number record, filename,
# awk '{print $NR}' bss	
# awk '$5==23000' filename	print only record where 23000 value is matching in 5th column
# awk '{print $0}' RS="/" bss	making column into rows
# awk '{print $0}' RS=" " bss	
# awk 'BEGIN { RS = "/" } ; { print $0 }' bss	the right time to do this is at the beginning of execution, before any input has been processed, so that the very first record will be read with the proper separator.
# awk '{ OFS = ":"; $2 = "" ; print ; print NF }' bss	
# cat numbers one,two,three!four,five,six!seven,eight,nine!ten	O/P of the script: two five

# awk 'BEGIN {RS="!";FS=","} {print $2}' numbers	eight
awk ' BEGIN { "date" | getline current_time print "Report printed on " current_time }'	Getline inbuild awk variable
# awk '{ print $n }' n=4 inv n=2 bss	using user defined variables
Arithmetic	
# awk 'BEGIN{a=1;b=3; print a + b}'	
# awk 'BEGIN {a=1;b=3; print a b}'	
# awk 'BEGIN {a=1;b=3; print a/b}'	
# awk 'BEGIN {a=1; b=2; c=3; print a b * c}'	
# awk 'BEGIN {a=1; b=2; c=2; print (a b) * c}'	
# awk '{ $2 = $2 + 10; print $0 }'	
# awk '{$3=$2-10; print $2,$3}' bss	for incrementing any column values from the input file.
# ls -l | awk '$6 == "Jul" {sum += $5 } END {print sum}'	
# awk '{print"total pay for " $1 " is " $5 }' emp	Finding total size of JUL month files
# awk -F: '$NF ~ /\/bin\/bash/ {n++}; END {print n}' /etc/passwd	
# awk -F: '{ a += NF } ; END {print a}' /etc/passwd	Count number of users who is using /bin/bash shell
Print	
#awk 'BEGIN {print "hi"}'	
#awk 'BEGIN {print "hi\nhello"}'	
awk 'BEGIN { print "line one\nline two\nline three" }' awk 'BEGIN { print "Month Cost" print "----- ------" } { print $1, $2 }' inv	
awk 'BEGIN { print "Month Crates" print "----- ------"} { print $1, $2 }' inv	
ls -l | awk ' BEGIN { print "\t List of html files:" } /.html/ { print } END { print "\t There you go!" }'	
awk ' { OFS = ";" ; ORS = "\n" } { print $1, $2 }' bss	
awk ' { OFS = ";" ; ORS = "" }	

{ print $1, $2 }' bss	
awk 'BEGIN { OFMT = "%d" print 17.23 }'	
awk 'BEGIN { OFMT = "%s" print 17.23 }'	print numbers as decimal
awk 'BEGIN { OFMT = "%f" print 17.23 }'	print numbers as string
awk '{ print $1 > "names.unsorted" print $1 | "sort -r > names.sorted" }' bss	print numbers as floating
Printf	
printf (“format”, value...)	
# awk -F, '{printf("%s\t%s\t%d\n", $1, $2, $3)}' emp	
# awk -F, '{printf("%20s %20s %3d\n", $1, $2, $3)}' emp	
# awk -F, '{printf("%-20s %-20s %-3d\n", $1, $2, $3)}' employees	
# awk '{ printf "%-10s %s\n", $1, $2 }' emp	
awk ’BEGIN { print "Name Number" print "---- ------" } { printf "%-10s %s\n", $1, $2 }’ bss	
awk 'BEGIN { printf "%-10s %s\n", "Name", "Number" printf "%-10s %s\n", "----", "------" } { printf "%-10s %s\n", $1, $2 }' bss	
awk 'BEGIN { format = "%-10s %s\n" printf format, "Name", "Number" printf format, "----", "------" } { printf format, $1, $2 }' bss	
awk 'BEGIN { print "Analysis of 'foo'" } /foo/ { ++a } END { print "foo appears " a " times." }' bss	To find the no of records in the input file ‘BBS-list’ that contain the string ‘foo’. The BEGIN The second rule increments the variable foobar every time a record containing the pattern ‘foo’ is read. The END rule prints the value of foobar at the end of the run.
Control statements	

if (condition)
{
Action
	If then syntax

```}; ```	
```x=2 awk ' BEGIN{ if (x % 2 == 0) print "x is even" else print "x is odd" }' ```	

``# awk '{
if (NF < 8) {print "short", $0}
else {print "long", $0}
}'emp ```	

```awk '{ if (NF > max) max = NF } END { print max }' file ```	

``awk '{
if ($3 == "" || $4 == "" || $5 == "")
print "Some score for the student",$1,"is missing";
}' students; ```	

``for (initialization, condition, increment)
{
body
} ; ```
	for loop syntax

``# awk ' {
for (i=1; i<=3; i++)
{print "Line " NR ", field "i": "$i;}
}' emp ```	

``while (condition)
{
action
}; ```
	while condition syntax

``awk ' BEGIN { i = 1
while (i <= 3) filename
{ print $i i++ }
} '; ```	

Standard way of creating shell scripts

Better file handling
Use only the necessary commands inside a loop
Gain more with less
One file to multiple files? Walk over the entire script
Using right conditions at right places in AND or OR
Wherever needed, go for internal commands
Avoid using lengthy commands
sed/awk parses the entire file, by default:
Best option vs any option.

1. Better file handling: Knowingly or unknowingly, a lot of files are created or deleted in a shell script. Due to the use of large number of files, handling of files become very important. Even a simple echo statement which re-directs output to a file has to open the file first, write data into the file and close the file. Let us look at an example:

```
#!/usr/bin/sh

cnt=1
while [ $cnt -ne 100 ]
do
	echo $cnt >> file
	let cnt=cnt+1
done
```

This script is simple to comprehend. Inside a while loop, at every increment of the count value, the variable is written or appended to the file. The loop is run for 100 times. Every time, the output file is opened, data is written and the file is closed. And yes, this happens for 100 times. In practical cases, where the loop is running on millions of records, this could be a huge time elapsed.

Now, look at the example below which is an improved version of the above:

```
#!/usr/bin/sh

cnt=1
while [ $cnt -ne 100 ]
do
	echo $cnt
	let cnt=cnt+1
done > file
```

Instead of writing the count at every instance, it is written once at the end of the while loop. Every time when the echo statement is executed, the output printed by echo remains in the buffer. When the while loop finishes, the entire buffer contents gets written into the file. Compare this solution to the above and just imagine the performance improvement.

Tip: Whenever you have a print statement in a loop, try checking whether they can be put in a better place.

2. Use ONLY the necessary commands inside a loop: It means not to use those commands which are not needed inside the loop. But, its common sense right? The point here is: Do not use commands inside loop which could very well have been outside loop. Let us consider the below example:

```
$ cat test.sh
#!/usr/bin/sh

cnt=1
for i in `cat file`
do
	DT=`date '+%Y%m%d'`
	FILE=${i}_${DT}
	echo $cnt > $FILE
	let cnt=cnt+1
done
```

What the script does is: A loop is run on the contents of file. For every entry in the file, a new filename is prepared which is nothing but the concatenation of the entry in the file along with the date. And a value is written to this new file. It looks simple and without any issues. No issues? Look at it again.

Why is the date command inside the loop? The date command is fetching the year, month and date which is going to remain the same (unless until script run through midnight in which case I do not think will be a genuine requirement here). The date could have very well served the purpose being outside the loop as well. If the input file on which the loop is running is to contain some 1000's of records, the date command is going to run 1000's of times where actually we wanted it only once. A huge performance issue. The improvement here would be to simply move the highlighted line before the for loop.

Tip : Always make it a point to have only the relevant commands at the right places.

3. One file to multiple files? Walk over the entire script: Whenever we have a requirement to write a script which is going to run on lots of files, the developer mindset is to write the script and make it work for one file. Once it works with one file, make it work for multiple files either by adding a loop or by passing command line arguments. Let us take the earlier example for now. The developer could well have written the lines inside the loop by hardcoding the variable "i", and once it is working, just enclosed the code inside a for loop. Due to this approach, the date command remained inside. . Lots of performance issue happens due to this reason.

Tip : Always make it a point, to walk over the entire script the moment a loop is put on an existing set of codes. You will be surprised to see many lines of code being irrelevant inside the loop.

4. Best option vs any option: The beauty in Unix or Unix flavors is it gives multiple options to achieve anything in it. Regular readers of this blog would have come across the umpteen articles on stuff wherein we explained the different ways in which a particular output can be achieved.

Let us consider a file. The requirement is to parse the file and read the 2 values into 2 different variables:

```
$ cat file
Solaris 25
Linux 21
AIX 40
```

Approach 1: Using the while loop:

```
$cat test.sh
#!/bin/bash

while read line
do
	OS=`echo $line | awk '{print $1}'`
	VALUE=`echo $line | awk '{print $2}'`
	echo "OS : $OS"
	echo "VALUE: $VALUE"
done < file
```

The while loop reads every line into a variable, and then using awk filters out each variable and stores it separately into two variables, OS and value. Frankly speaking, we wasted 2 awk commands here. The awk was not needed at all. Check out the next approach.

Approach 2: Using the same while loop, but without awk:

```
$cat test.sh
#!/bin/bash

while read OS VALUE
do
	echo "OS : $OS"
	echo "VALUE: $VALUE"
done < file
```

As seen in one of our articles, while loop has all the properties to read a text file or CSV file effeciently as shown above. And hence, the awk is not needed at all. Both the above methods provide the right result. Which is better? Very simple, the second one because we achieved the result with much lesser number of commands.

Tip : Whenever, you achieve a result with a series of commands piped to each other, try looking for different options to see whether it can be improved.

Note: It always does not mean that the one with more commands will take lot of time than the one with lesser number of commands. If the option with a single command is written poorly, it can equally mess-up.

5. Wherever possible, internal command always: In one of our articles, we saw the difference between internal and external commands. Internal commands are internal to the shell which the shell executes without creating any process whereas for the external commands, a process is created. Due to this, internal commands are always much much faster compared to external command.

Example 1 to find the length of a string:

```
$ x="welcome"
$ expr $x : '.*'
7
$ echo ${#x}
7
```

2 different commands are used. One using expr which is an external command, other using echo which is an internal command. The echo will be much better in performance than the expr.

Example 2 to read a file line by line in Shell:

Option 1:

```
$ cat file | while read line
> do
>  echo $line
> done
```

Option 2:

```
$ while read line
> do
>   echo $line
> done < file
```

In the first option, we use the cat command and pipe the output to the while command. However, in the Option 2, it is purely internal where the file is read using the input file descriptor.

Tip : Always prefer internal commands if possible.

6. Avoid Useless use of any command: There is a popular term known as UUC which stands for Useless use of cat. This means using cat command when actually it is not needed at all. For example:

```
$ cat file | grep Linux
```

This command could very well have been 'grep Linux file'. This is called UUC. Actually, if you look a little carefully, we might use many other commands which are pretty useless. Many more instances like these, many such useless commands we will come across. Let us look at another example of same kind:

```
$ grep Linux file | awk '{print $2}'
```

could very well have been:

```
$ awk '/Linux/{print $2}' file
```

Never use commands which are not needed at first place. Having said, this can be improved by getting exposed to lot of other commands available in the Unix flavor in which you are working.

Tip : Whenever you try with any command at the command prompt, be it for any small activity, always look for many different ways in which it can be achieved. This way you will get exposed to different options to achieve with which you can choose the right and the better option.

7. Achieve more with less: Say, you want to have 3 variables one containing the year, next containing the current month and the third containing the day. So, the commands for it could be:

```
$ year=`date '+%Y'`
$ mon=`date '+%m'`
$ dt=`date '+%d'`
```

Well, we used 3 date commands which,actually, could very well have been done with a single date command.

```
$ DATE=`date '+%Y%m%d'`
$ year=${DATE:0:4}
$ mon=${DATE:4:2}
$ day=${DATE:6:2}
```

See the difference!!! All we did is, in one date command, got the year, month and day. And used the shell substring function to extract them into different variables. Now, we have one external date command, and 3 internal shell substring operations. And imagine the performance improvement it brings when this is run many times in a script.

Tip: While using a command more than once, check for the possibility of getting the result with least number of commands.

8. Do not use ls always for file listing: Keep in mind, ls is not the only command to list files, there are different options to list the files. Just do a "echo *" at the prompt and see what happens!!!. When we want to process many files in a loop, say to process all the .txt files in a loop:

Option 1:

```
#!/usr/bin/sh

for file in `ls *.txt`
do
    echo $file
done
```

Option 2:

```
#!/usr/bin/sh
for file in  *.txt
do
    echo $file
done
```

As you saw now, ls was not needed at all. Shell has many its own properties which we can discuss in some other thread. But, when it comes to using ls, use only if it really needed.

Tip: Use ls only if needed.

9. sed/awk parses the entire file, by default: Say, you want to print the first 2 lines of a file using sed:

$ sed -n '1,2p' file

This command starts reading the file. When the first line is read, it prints, and when second line is read, it prints. From 3rd line onwards, it reads, but does not print anything. The point here is: sed still reads the entire file. Assume a huge file with millions of records in it. For printing 2 lines, the command reads the entire file which is bad performance. .

So, the solution for this is below:

$ sed '2q' file

This command on printing the 2nd line quits the file and comes out. The same is applicable for awk as well. For the same requirement, if one has to do in awk using better performance:

$ awk 'NR==3{exit;}1' file

Tip: Always make sure you process only what is needed.

10. Using right conditions at right places in AND or OR: In shell scripting also, we can have logical AND and OR conditions. In an AND condition involving 2 conditions, if the first condition is not true, the second condition is not even evaluated since the result is anyway going to be false. Always make sure to put the condition with more failure chances as the first in the AND condition. In this way, you will avoid the second condition most of the times. In case of an OR condition, it is just the reverse.Always put the highest success possibility condition in the beginning.

Booting Sequence

POST -- power on self test (will make sure power is supplied to all hardware parts in the CPU)
CMOS -- to check date/time – Battery (permanent, non volatile memory)
BIOS -- basic i/p o/p system -- to check boot order (1st hdd, cd, usb, network)
MBR -- it presents in hard disk
-- master boot record (boot loader (grub) -> partition table -> magic number
-- magic number 0 is clean.
-- magic number 1 is corrupted.
-- grub, grand unified bootloader , /boot/grub/grub.conf
kernel -- /sbin/init (/boot/vmlinuz-2.6.32-71.el6.x86_64 -- kernel file)
init -- executes run level (its a first process running when OS gets booted, PID as 1)
runlevel -- /etc/rc.d/rc*.d/rc* --
0 6
kdump -- /etc/sysconfig/kdump

RHEL Versions	Boot Loader
rhel 4	LILO
rhel 5	GRUB
rhel 7	GRUB2

RUN LEVELS:

It ranges from 0 to 6

/etc/inittab -- configuration file

0 Used to halt the system. This runlevel is reserved and cannot be changed.
1 Used to run in a single-user mode. This runlevel is reserved and cannot be changed.
2 Multiuser, without NFS (The same as 3, if you do not have networking) in CLI
3 Used to run in a full multi-user mode with a command-line user interface.
4 Not used by default. You are free to define it yourself.
5 Used to run in a full multi-user mode with a graphical user interface.
6 Used to reboot the system. This runlevel is reserved and cannot be changed.

Important files in RHEL	
/dev/null	A pseudo device, that don't exist. Sometime garbage output is redirected to /dev/null, so that it gets lost, forever.
/etc/bashrc	Contains system defaults and aliases used by bash shell.
/etc/crontab	A shell script to run specified commands on a predefined time Interval.
/etc/exports	Information of the file system available on network.
/etc/fstab	Information of Disk Drive and their mount point.
/etc/group	Information of Security Group.
/etc/init.d	Service startup Script.
/etc/hosts	Information of Ip addresses and corresponding host names.
/etc/hosts.allow	List of hosts allowed to access services on the local machine.
/etc/inittab	INIT process and their interaction at various run level.
/etc/issue	Allows to edit the pre-login message.
/etc/motd	motd stands for Message Of The Day, The Message users gets upon login.
/etc/mtab	Currently mounted blocks information.
/etc/passwd	Contains password of system users in a shadow file, a security implementation.
/etc/profile	Bash shell defaults
/etc/profile.d	Application script, executed after login.
/etc/rc.d	Information about run level specific script.
/etc/rc.d/init.d	Run Level Initialisation Script.
/etc/resolv.conf	Domain Name Servers (DNS) being used by System.
/etc/securetty	Terminal List, where root login is possible.

/etc/skel	Script that populates new user home directory.
/usr/bin	Normal user executable commands.
/usr/include	Contains include files used by ‘c‘ program.
/usr/share	Shared directories of man files, info files, etc.
/usr/lib	Library files which are required during program compilation.
/usr/sbin	Commands for Super User, for System Administration.
/proc/cpuinfo	CPU Information
/proc/filesystems	File-system Information being used currently.
/proc/interrupts	Information about the current interrupts being utilised currently.
/proc/ioports	Contains all the Input/Output addresses used by devices on the server.
/proc/meminfo	Memory Usages Information.
/proc/modules	Currently using kernel module.
/proc/stat	Detailed Statistics of the current System.
/proc/swaps	Swap File Information.
/var/log/lastlog	log of last boot process.
/var/log/messages	log of messages produced by syslog daemon at boot.
/var/log/wtmp	list login time and duration of each user on the system currently.

 # netstat -punta | grep 80 , to check whether port 80 in use
 # lsof , to list the open files
 # lsof -i :80 , to check whether port 80 in use