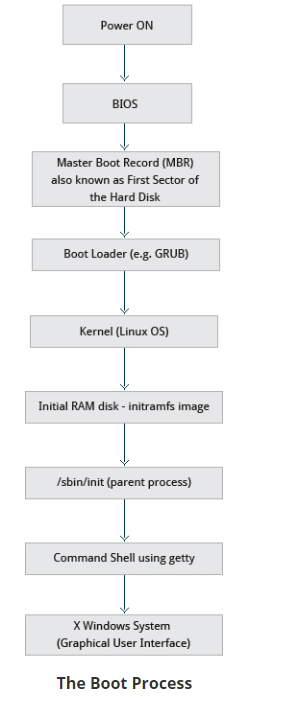
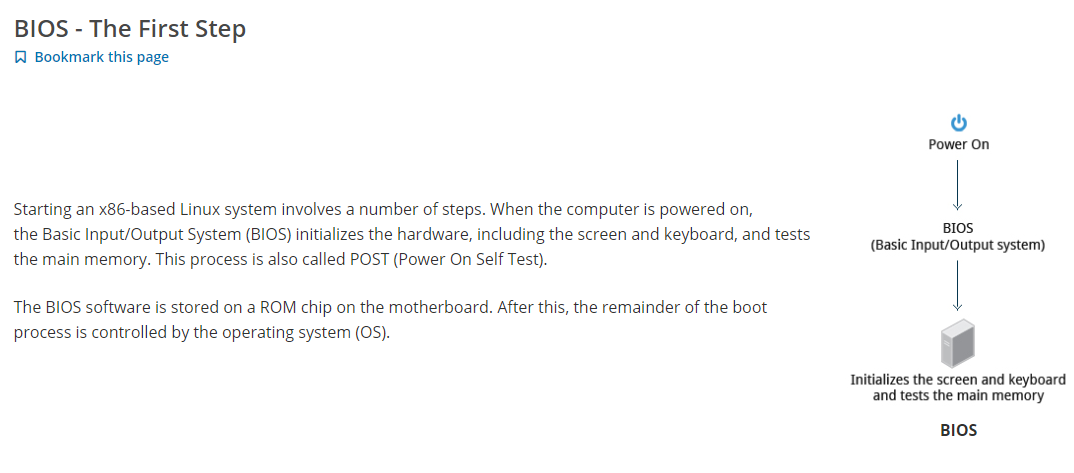
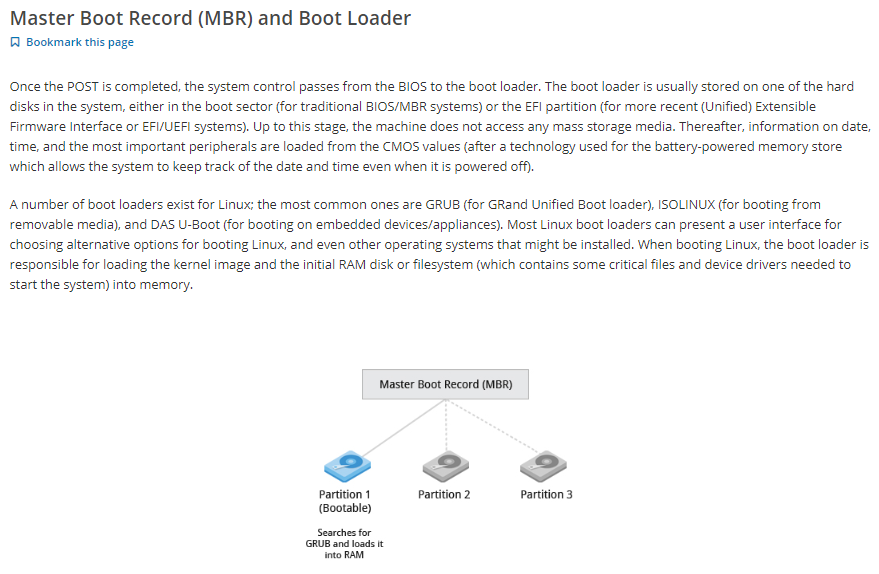
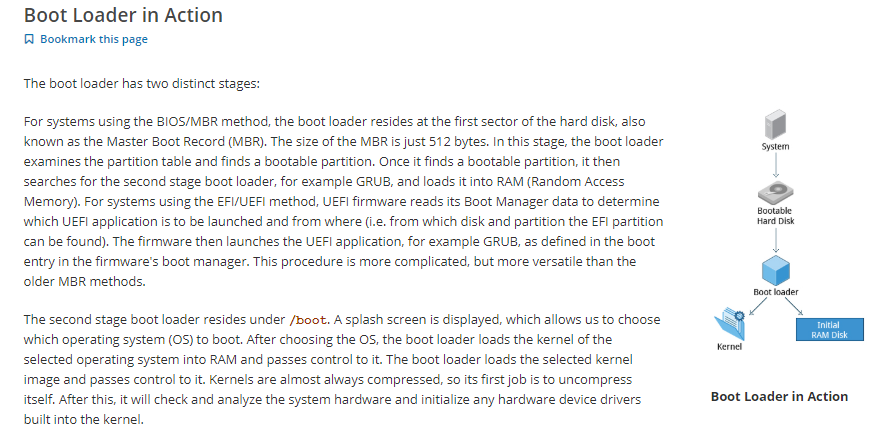
Linux system administration :

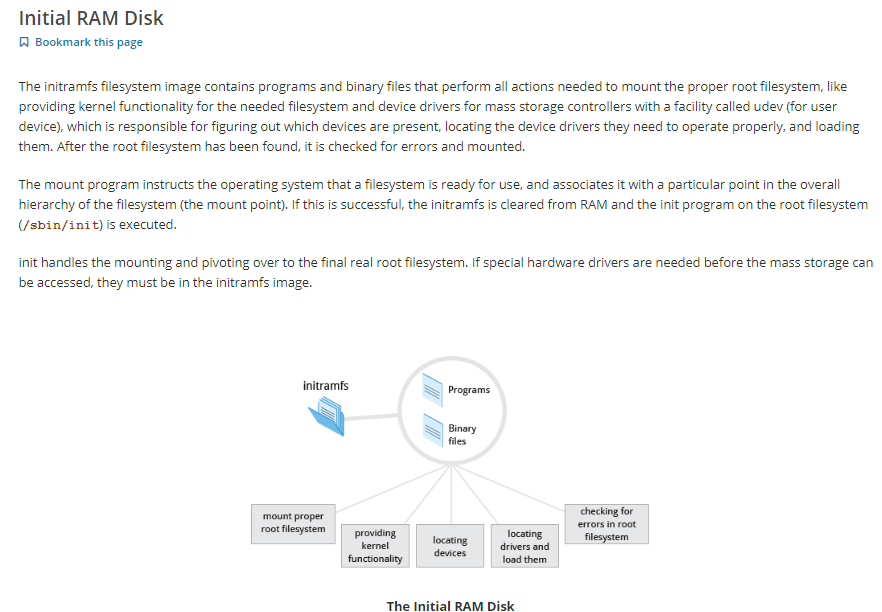
Linux boot process:

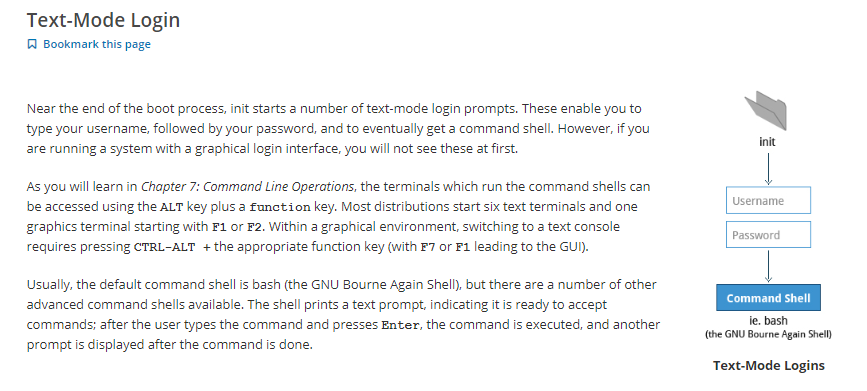


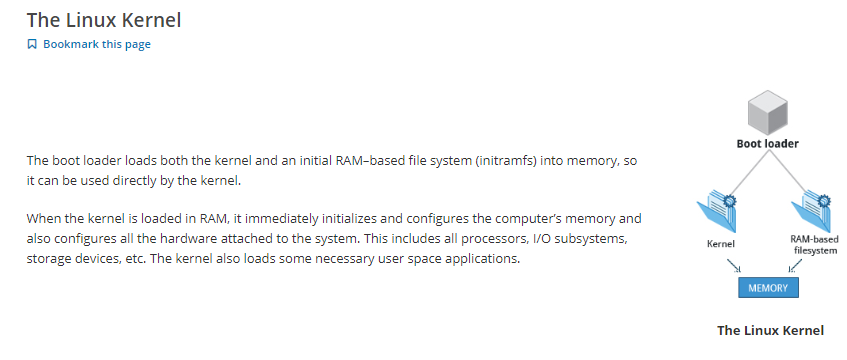


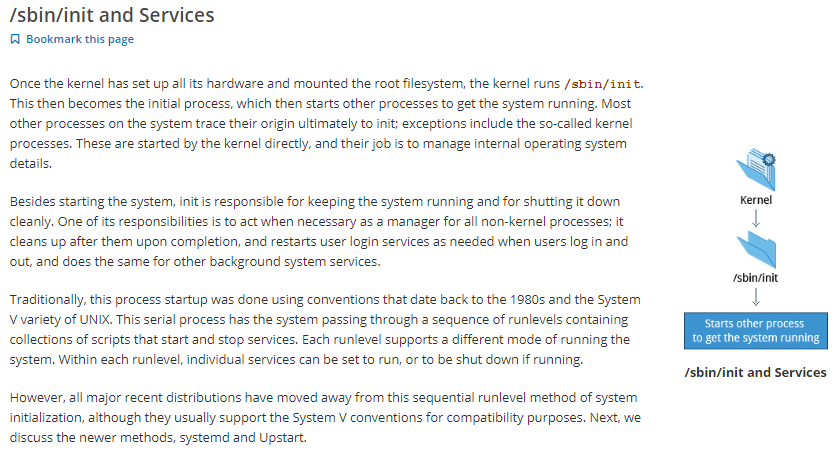


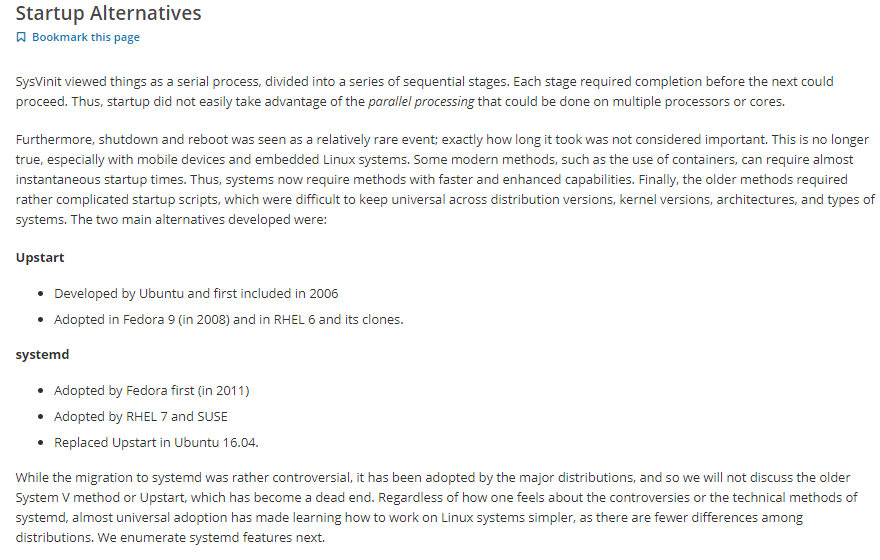


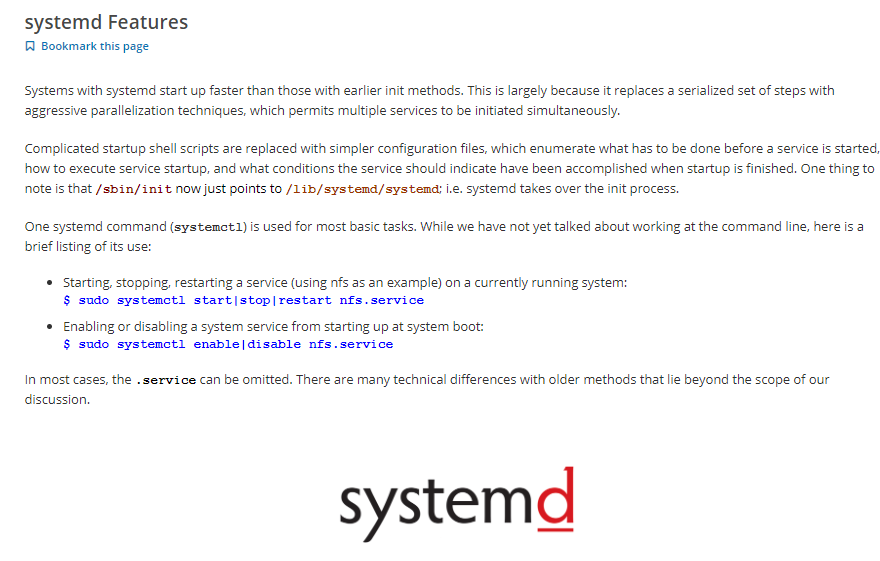


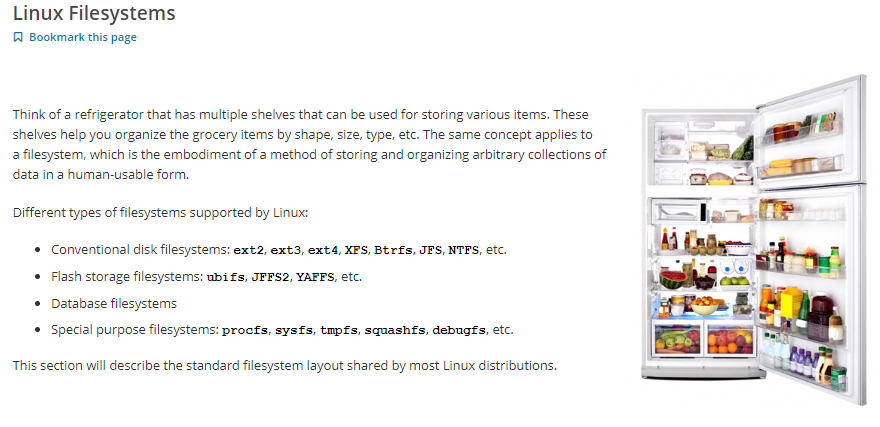


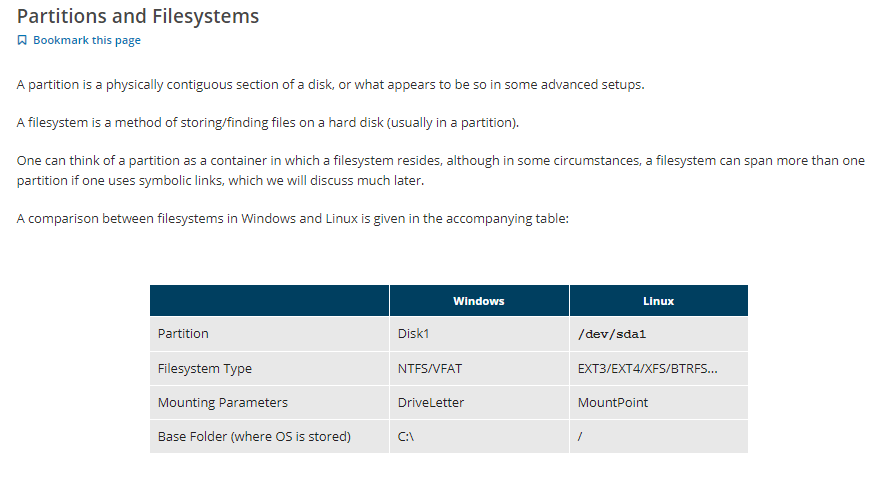


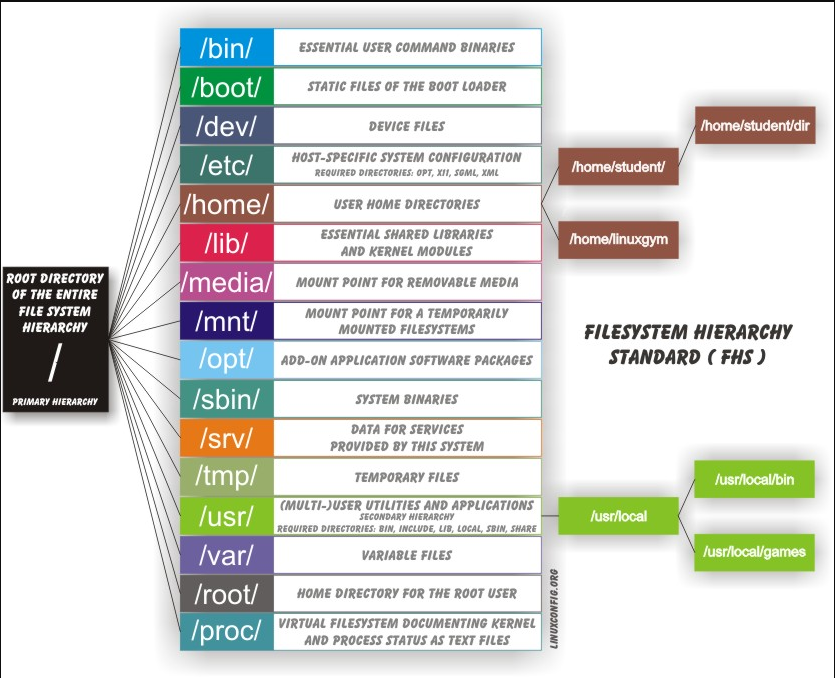


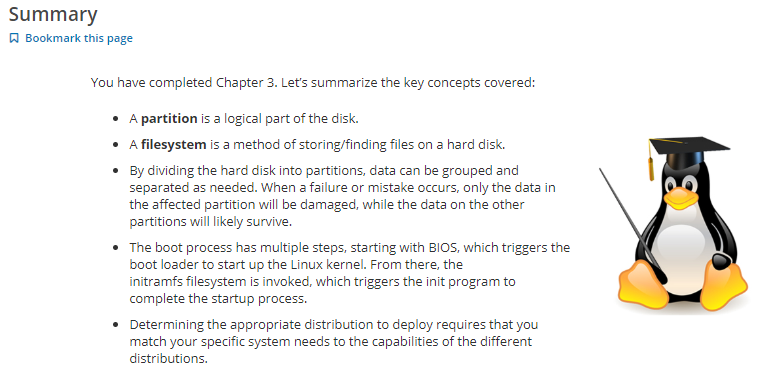




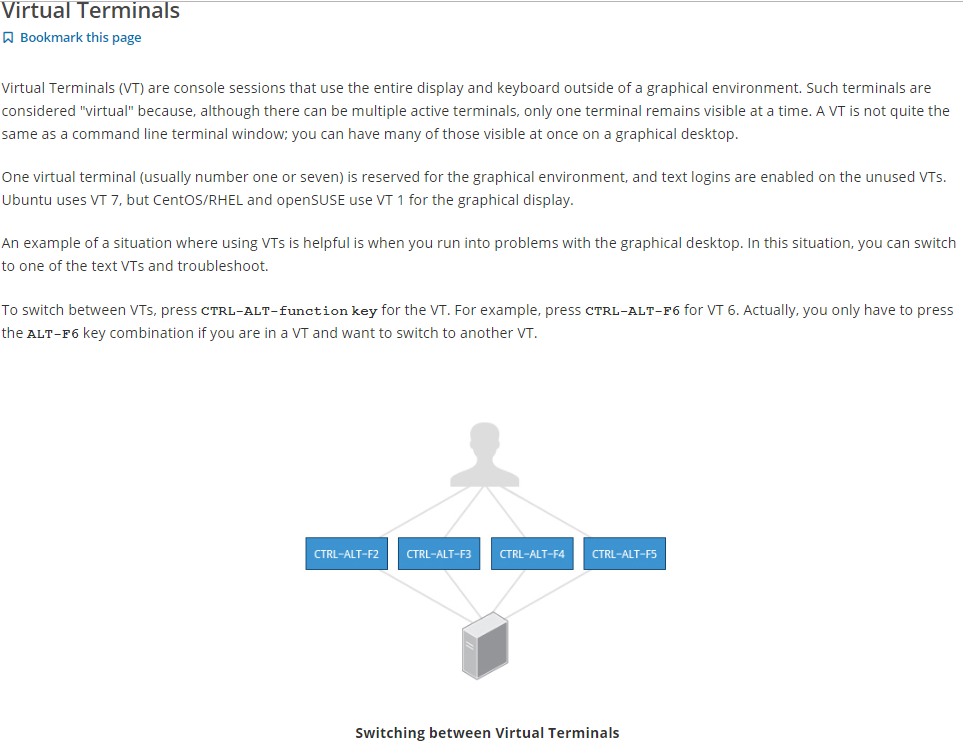


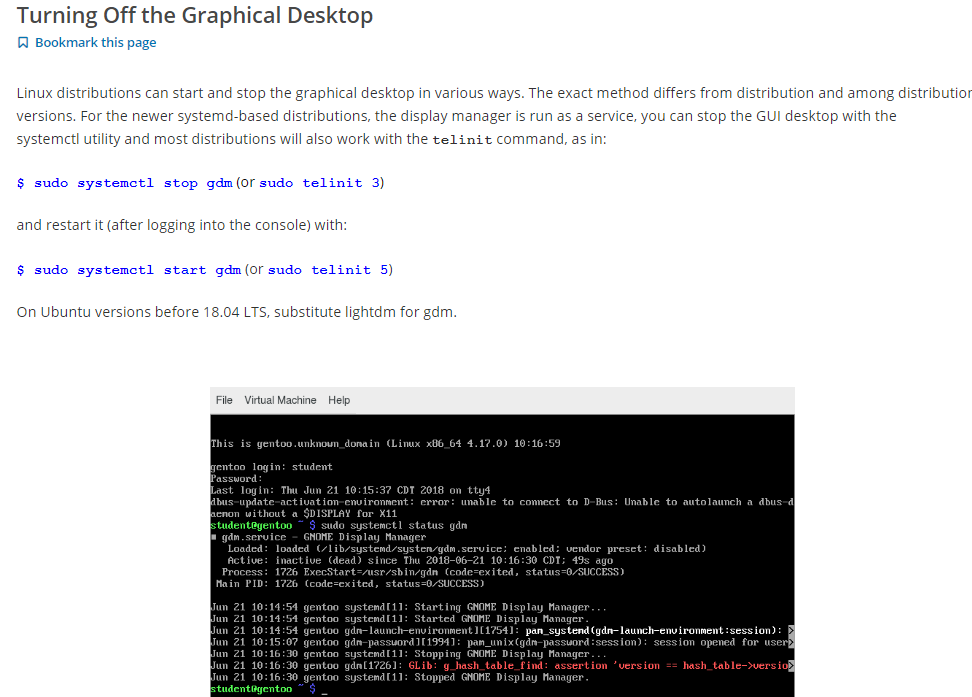


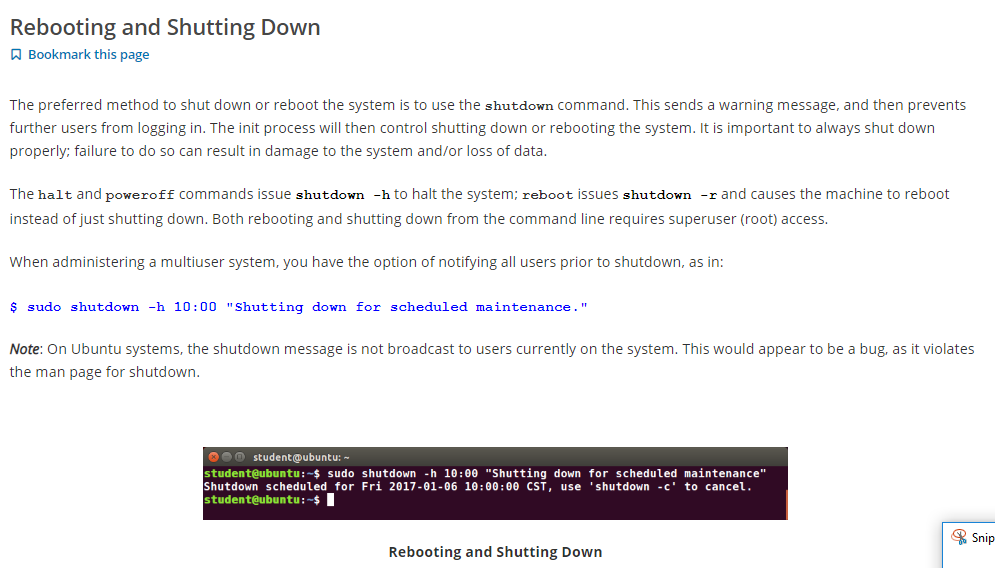


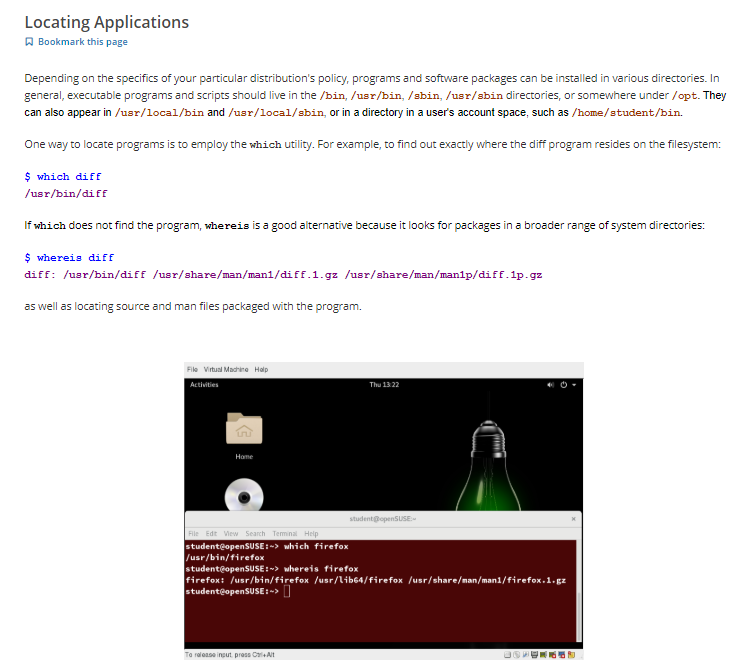


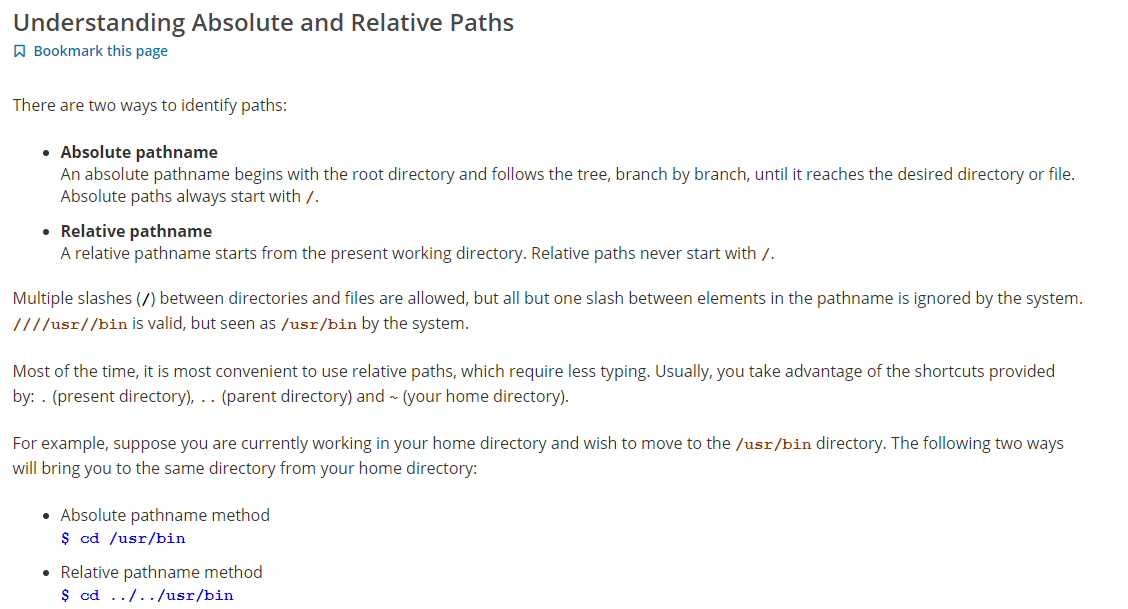


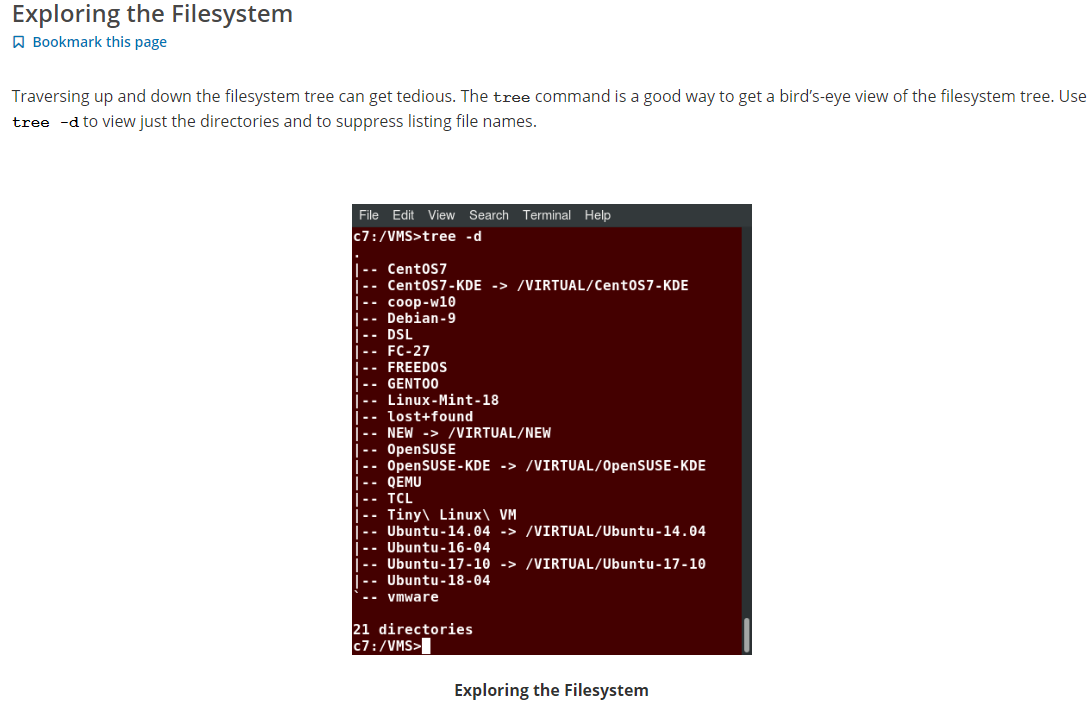


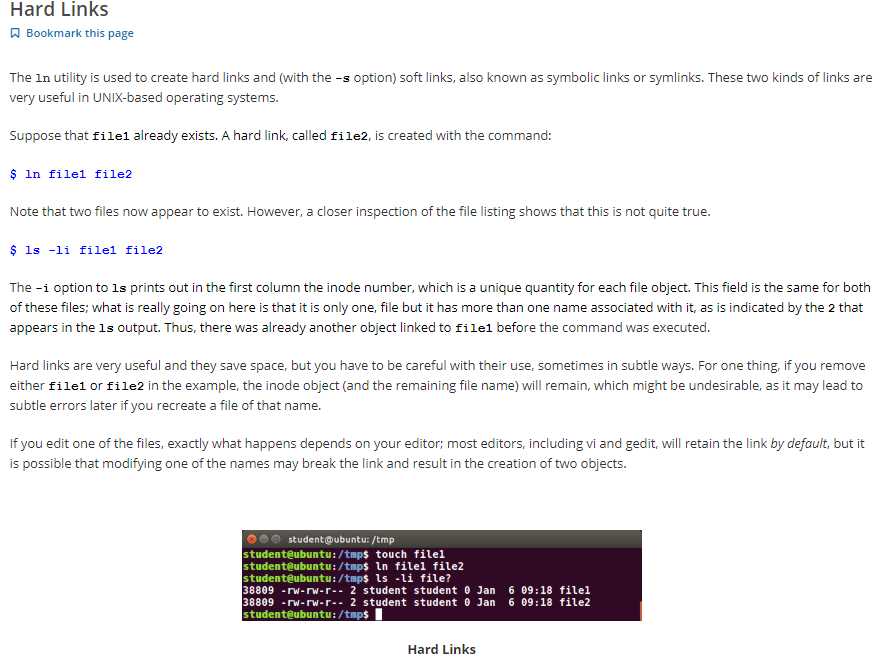


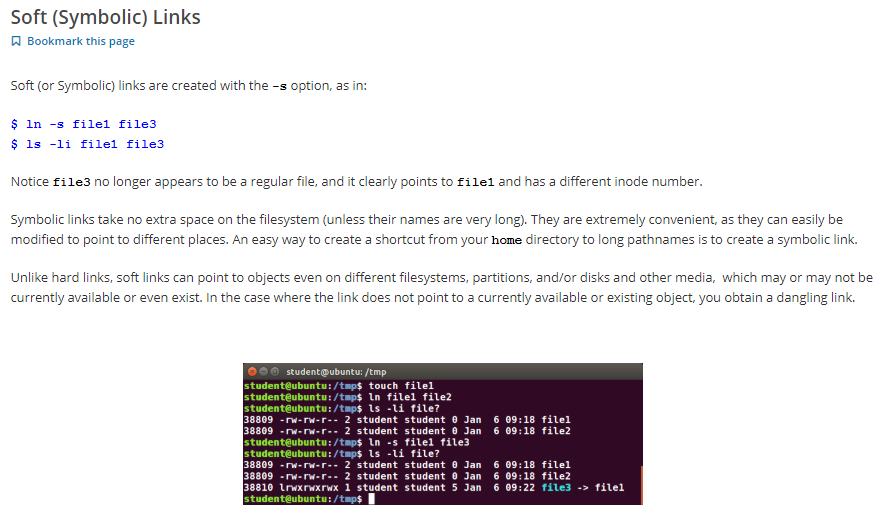


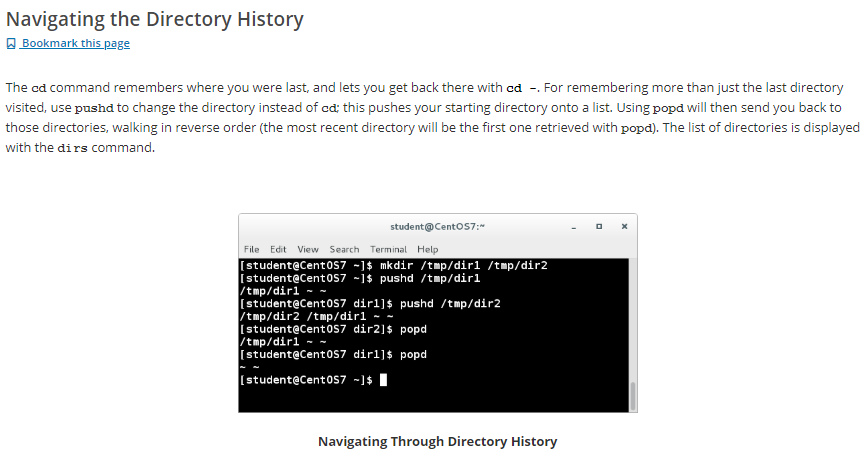


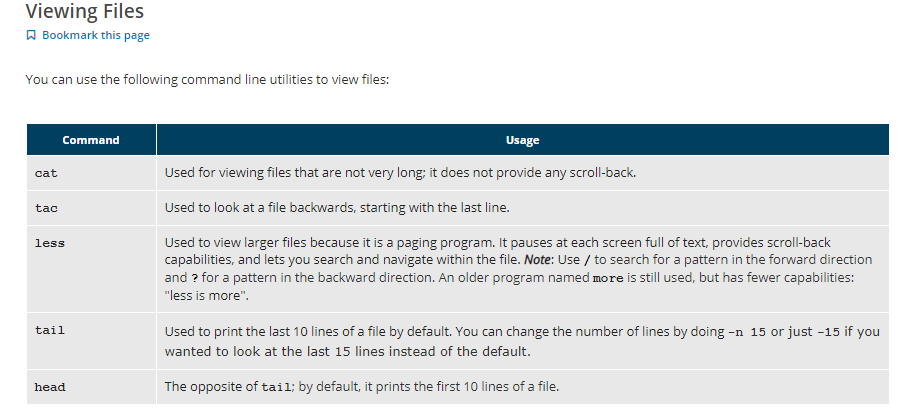


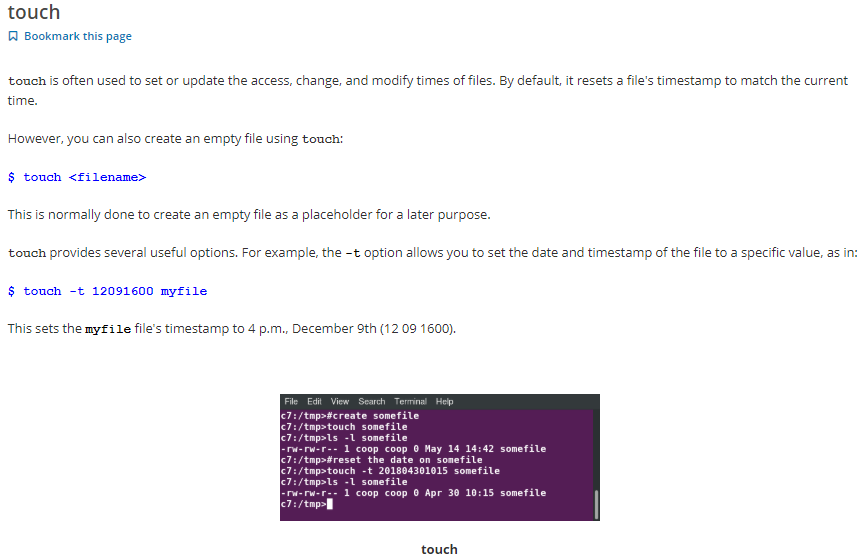




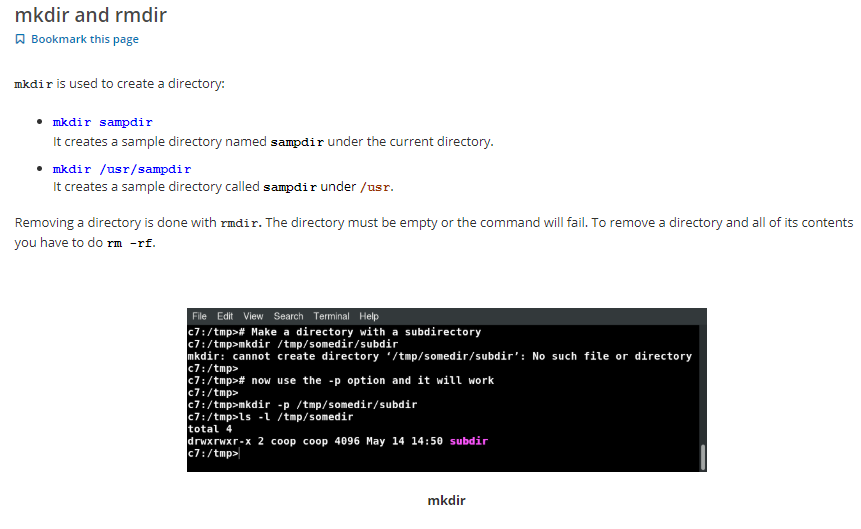


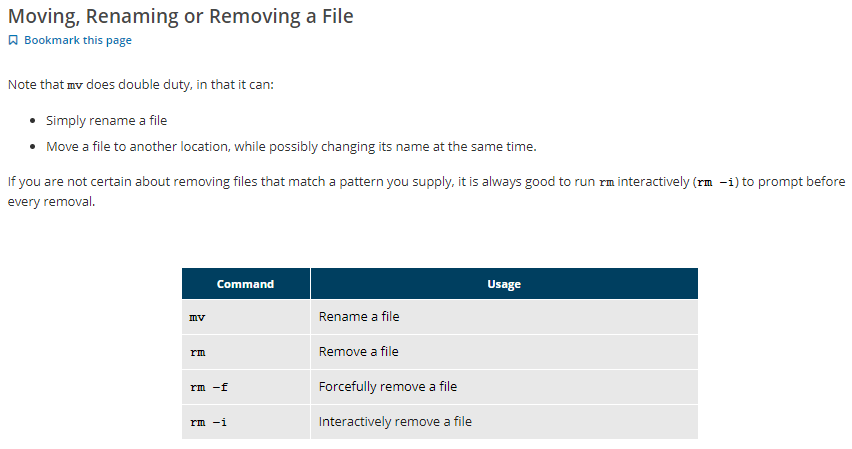


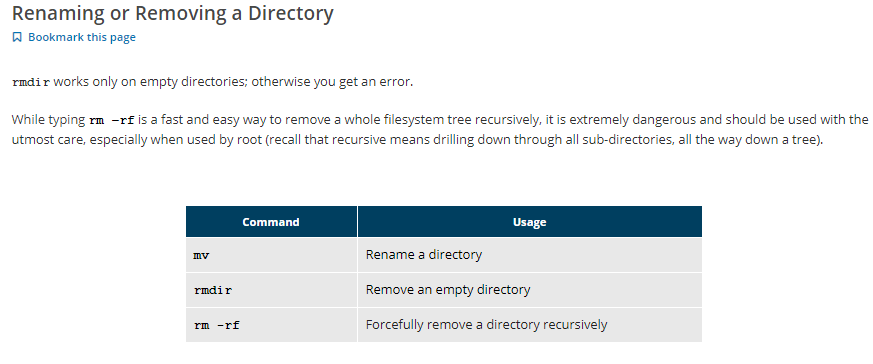


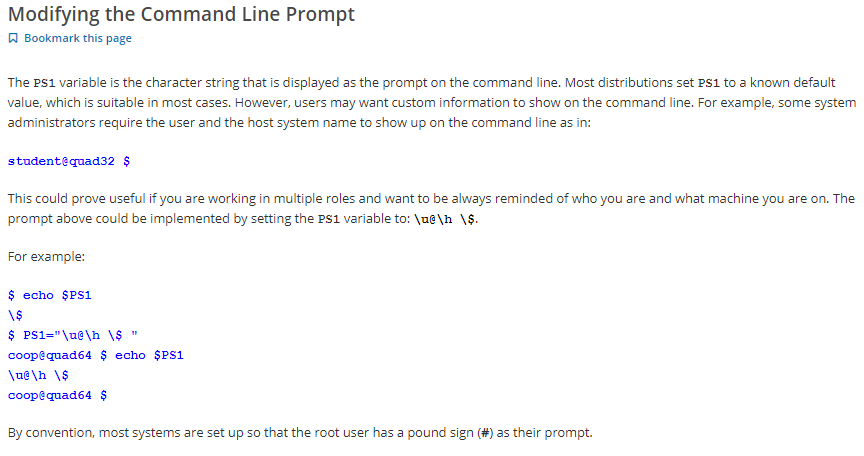


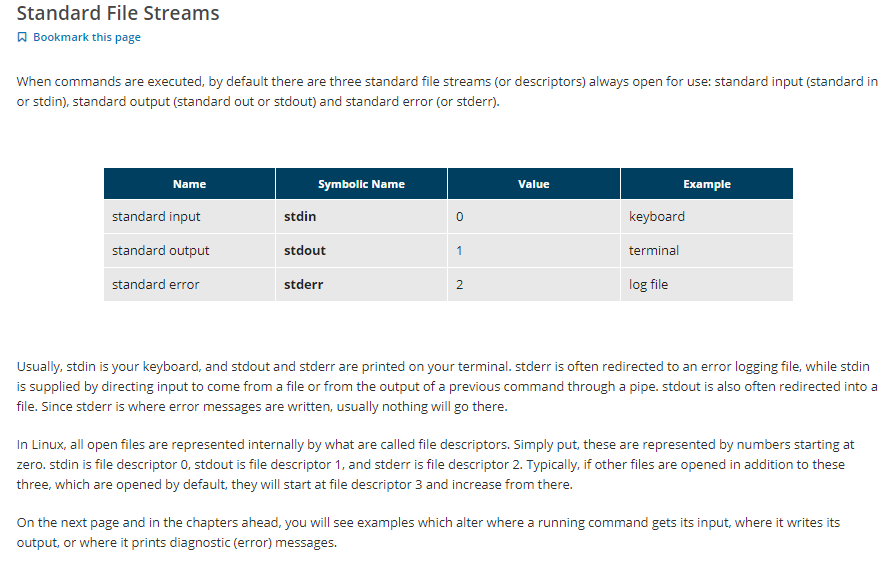
\*we can set the time stamp of any file using format (yymmddhhmm).





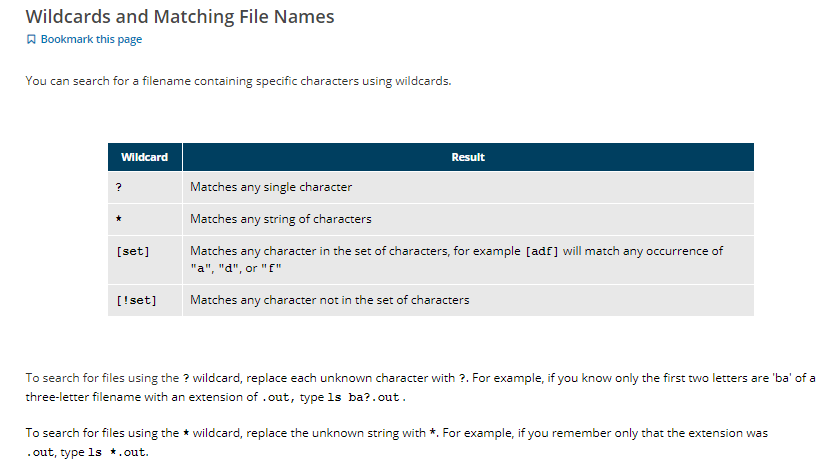








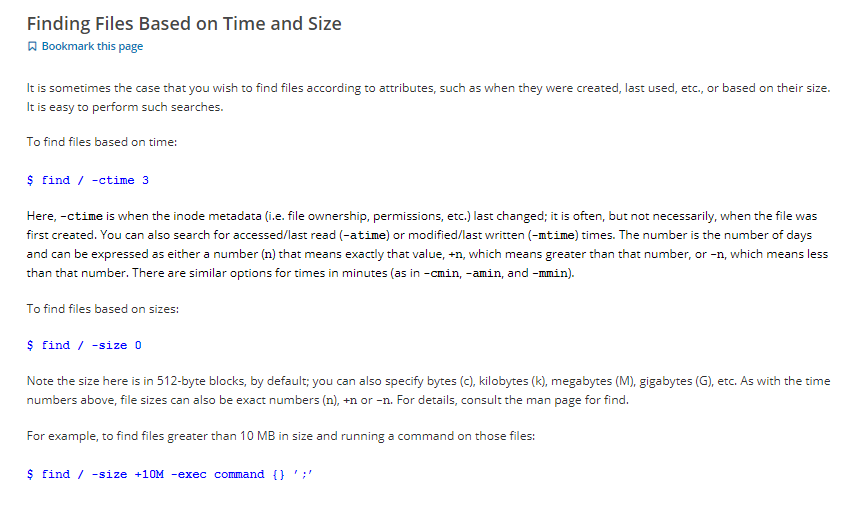


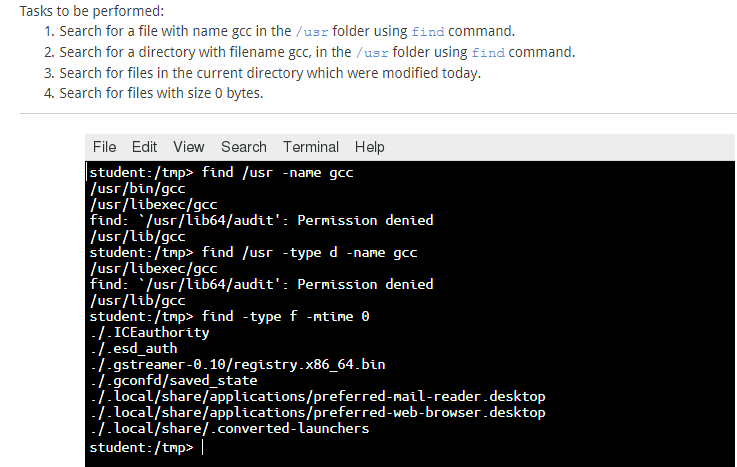


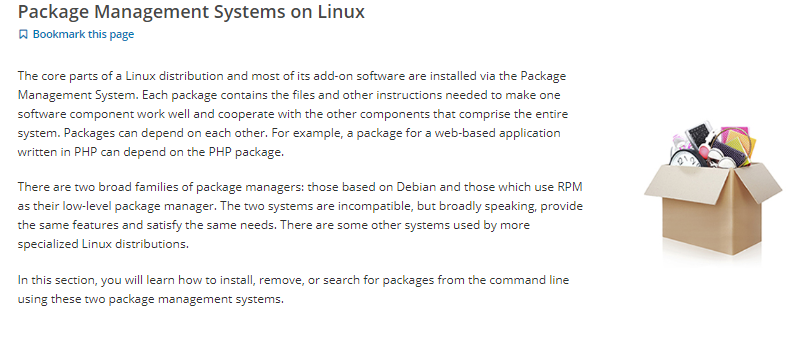


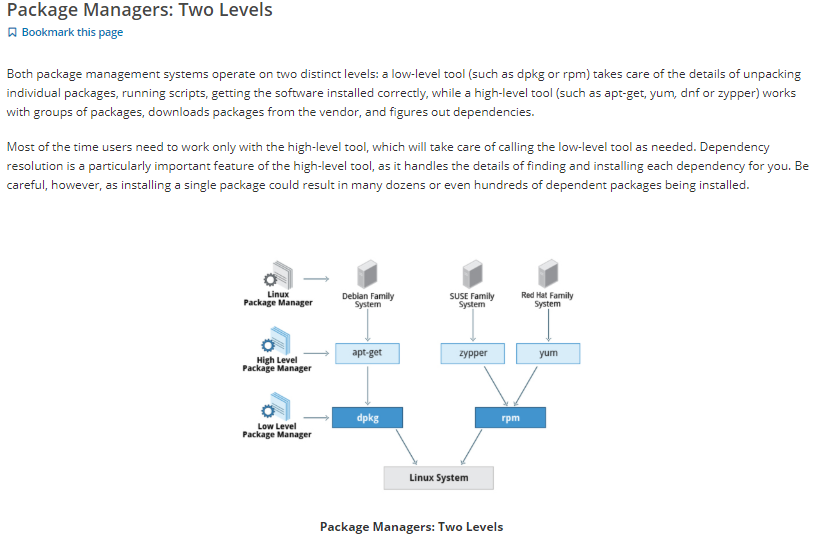


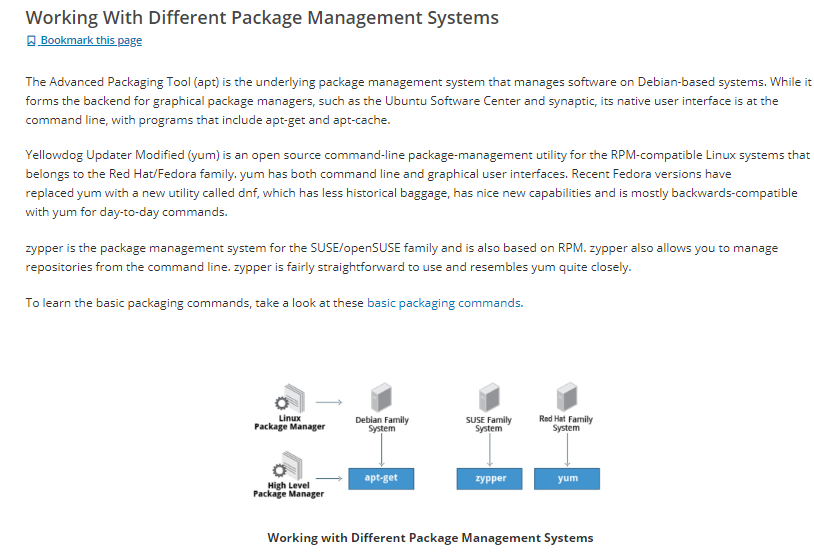












**Debian :**

dpkg(low –level package manager)

dpkg --list | less ----- to check all the packages

dpkg --list | grep zip ---- to check info about particular package

dpkg --listfiles zip | less --- to see the files inside package

dpkg --remove zip -- to remove package

apt( High level package manager)

apt-cache search wget -- to search package

apt-get install wget --- to install package

apt-get remove wget – to remove package

**RPM** (low level package manager)

rpm -qa ---to check all packages

rpm –qa | grep python --- to find python packages

rpm -qil python --- to check the details of package

rpm –e python –to remove package

yum (High level package manager)

yum list python ---to see the package

yum info python ---to see the details of package

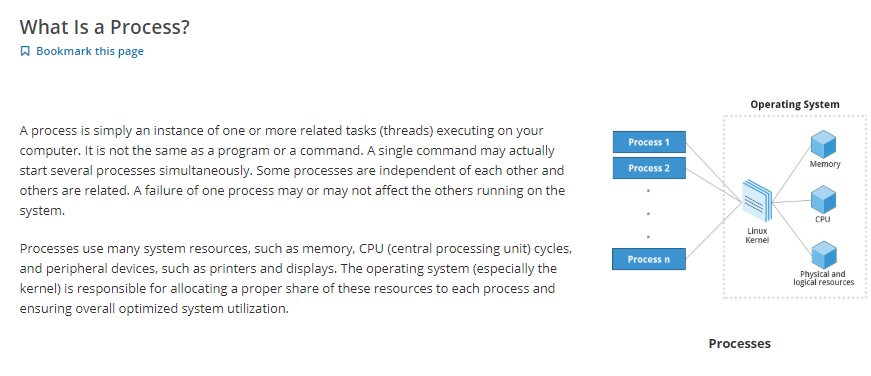
yum install python – to install package

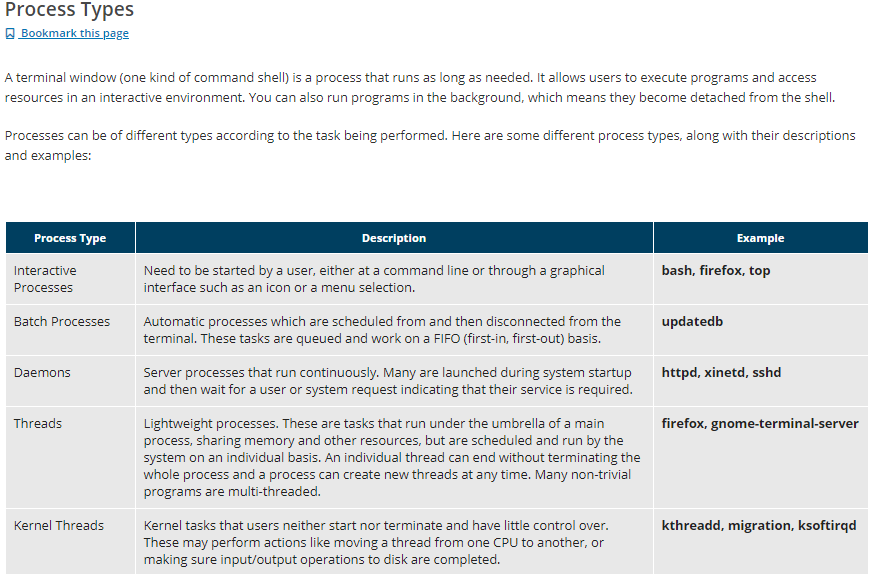
yum remove python –to remove package

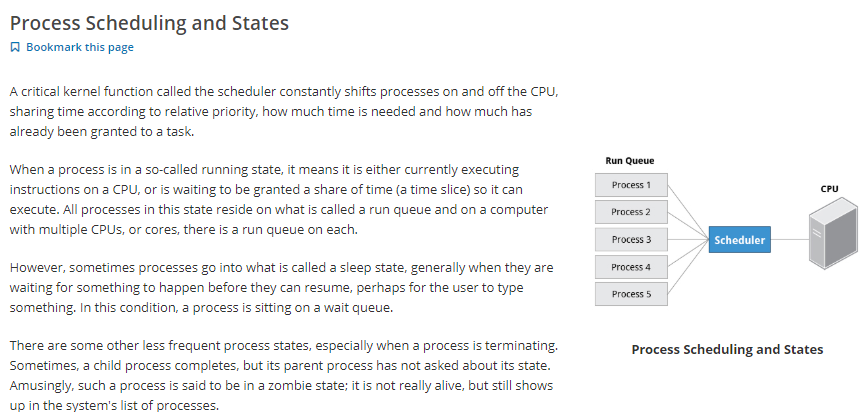


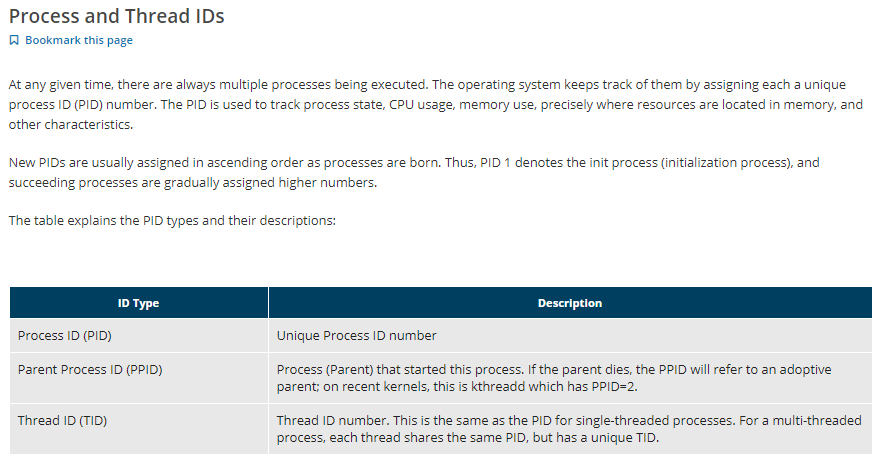
* To check all pages give cmd - man –a program name.eg man –a ls .

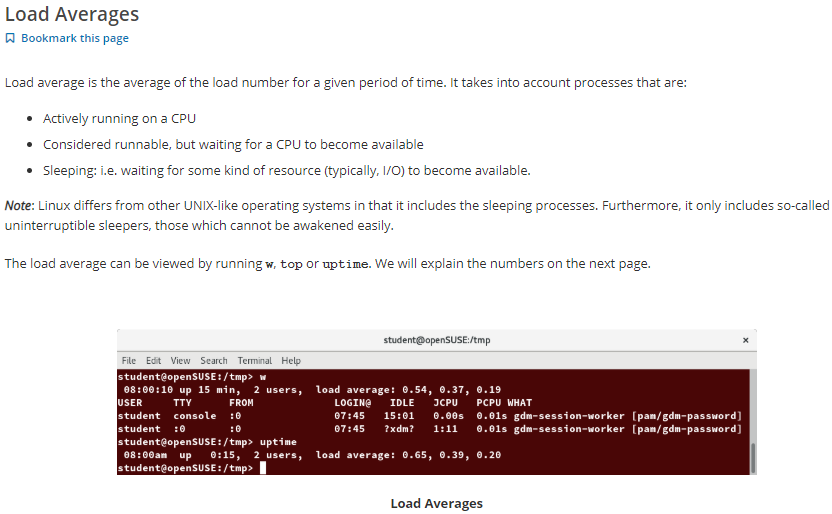
\*We can also find details by cmd : info ls \

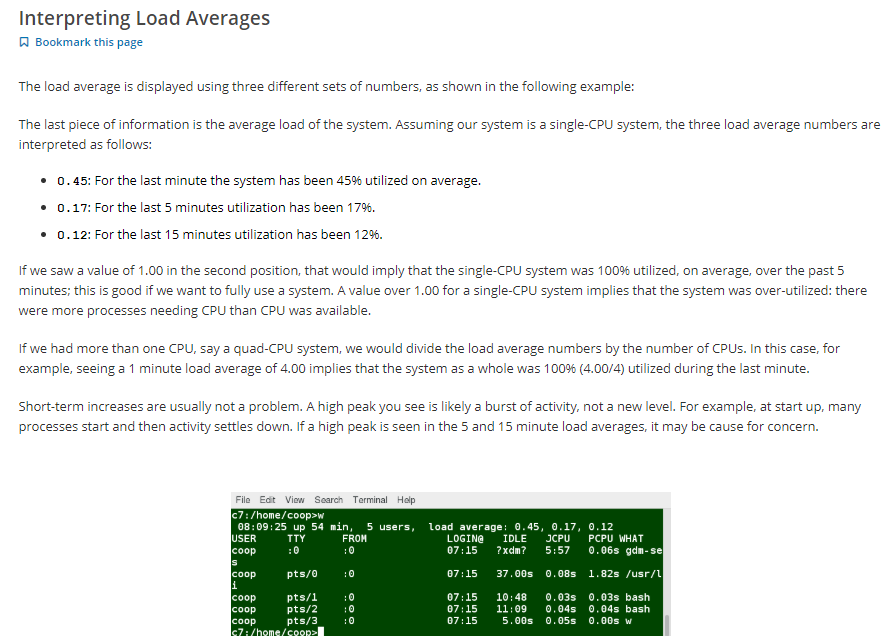


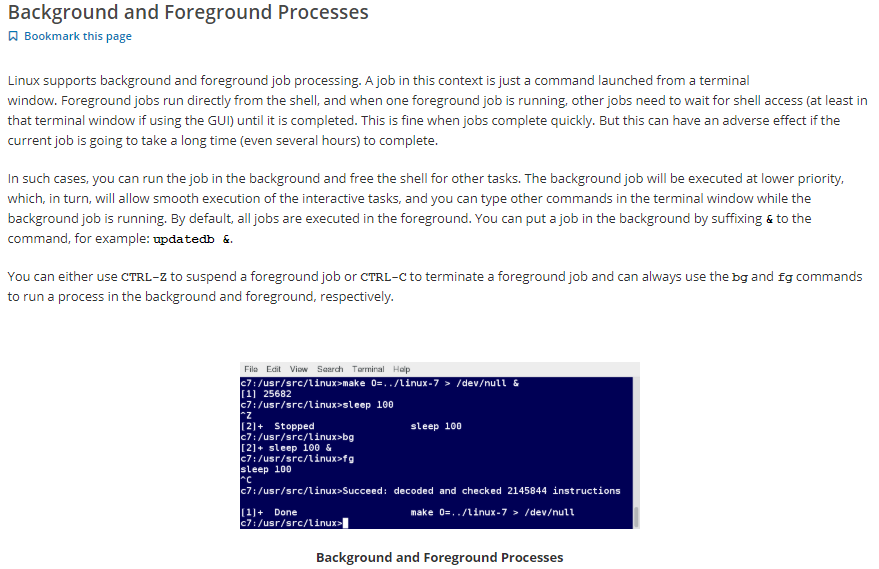




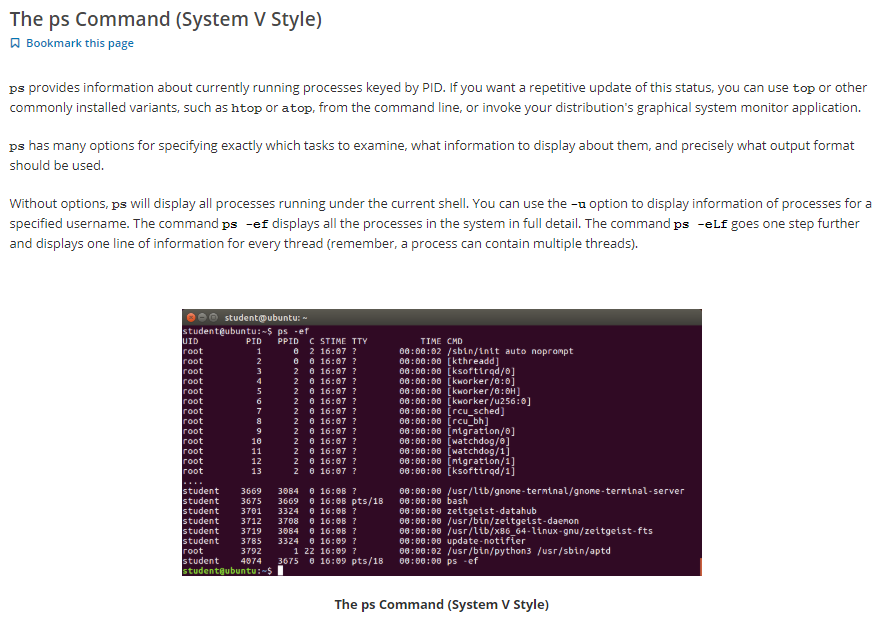


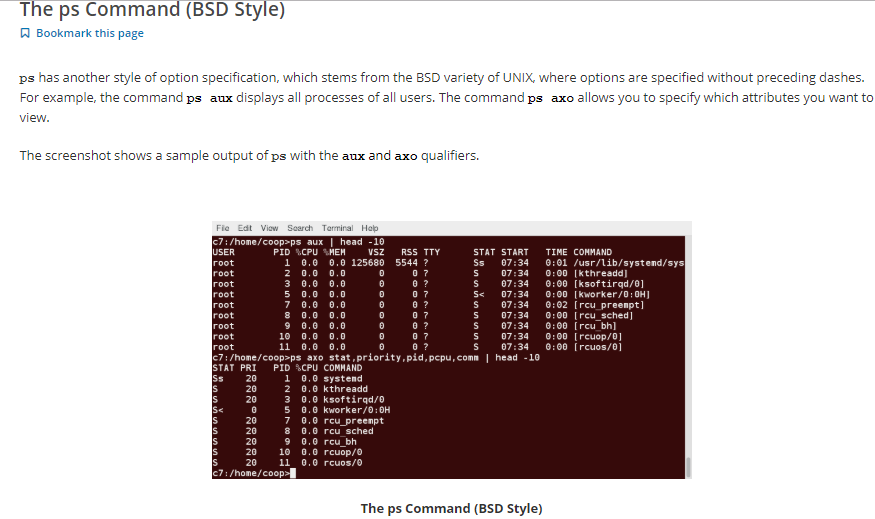


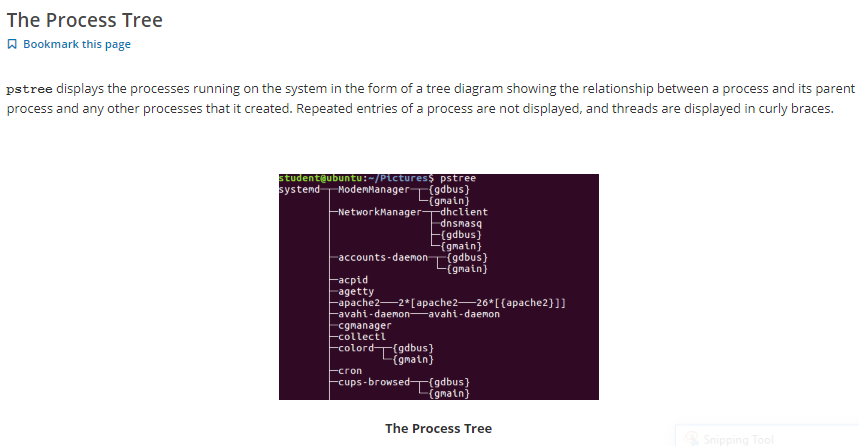


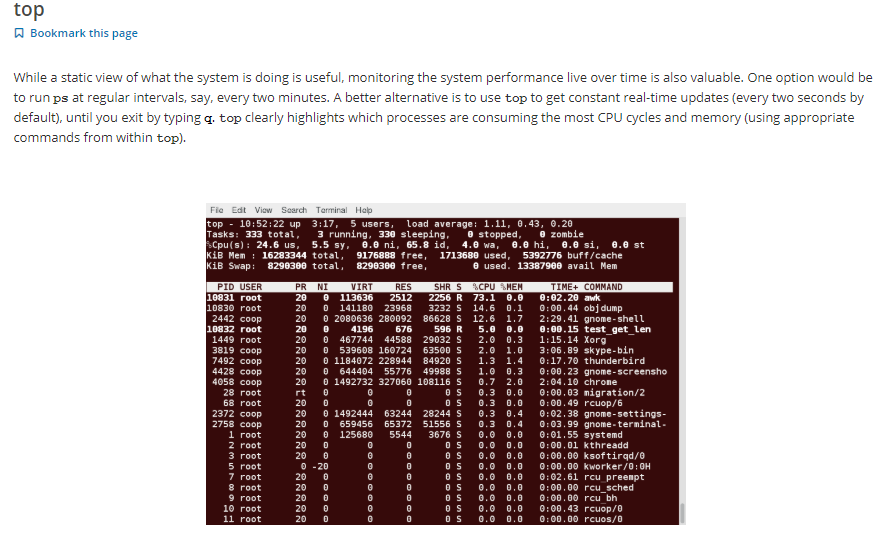


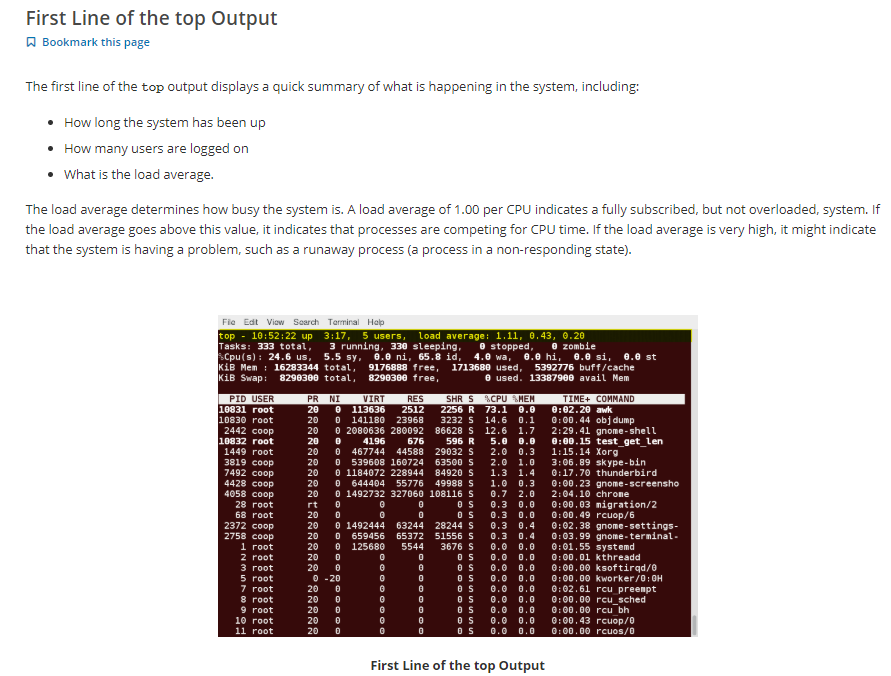


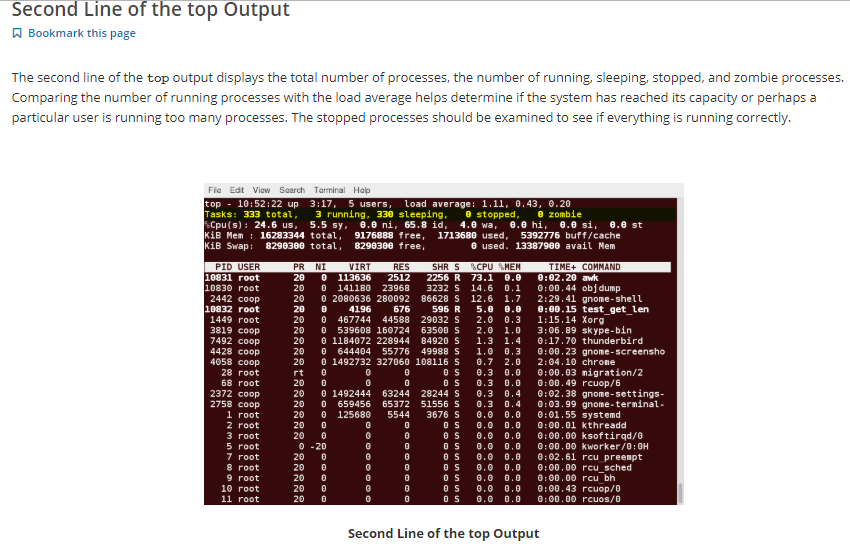


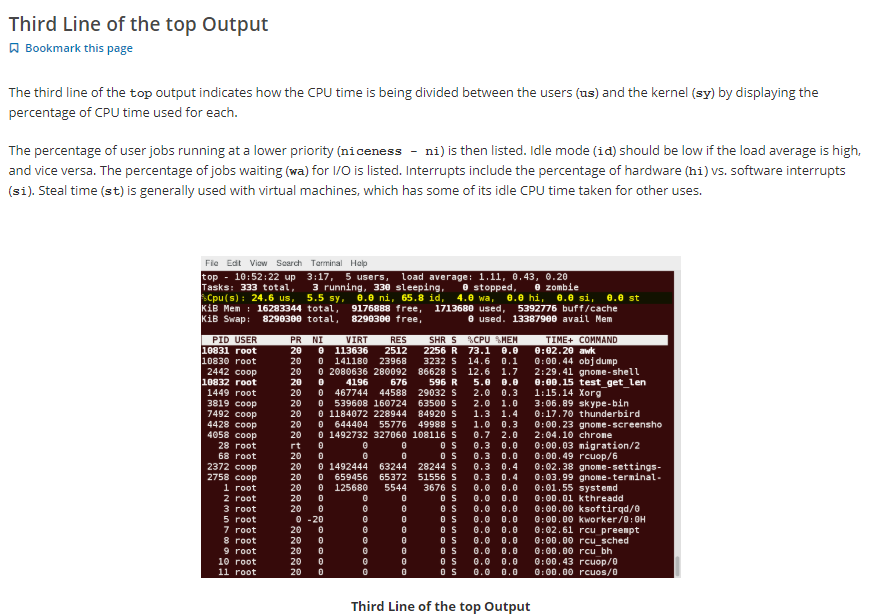


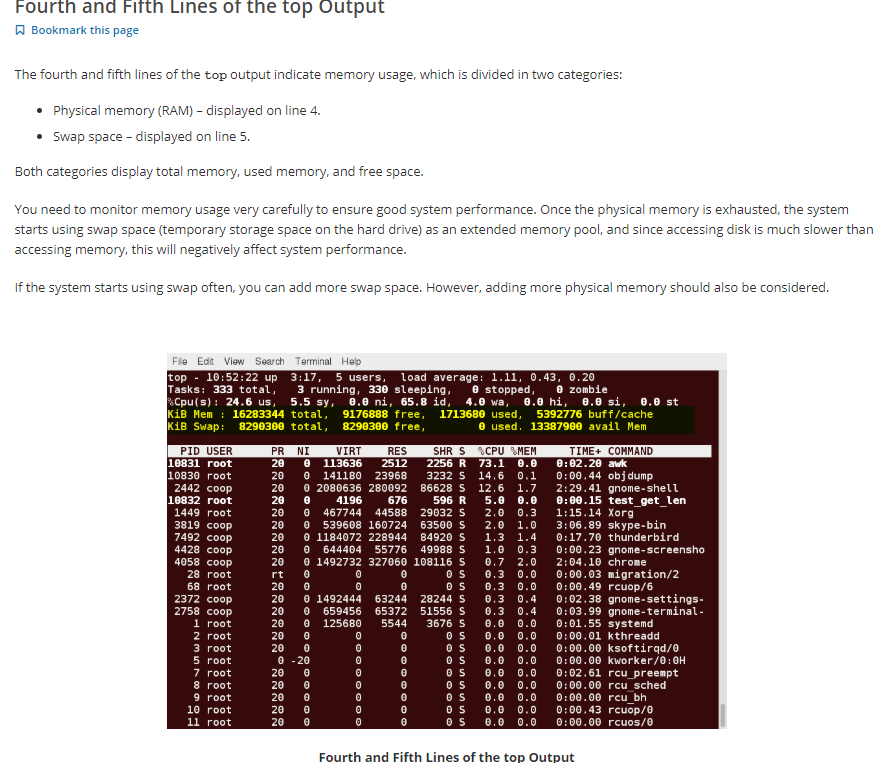


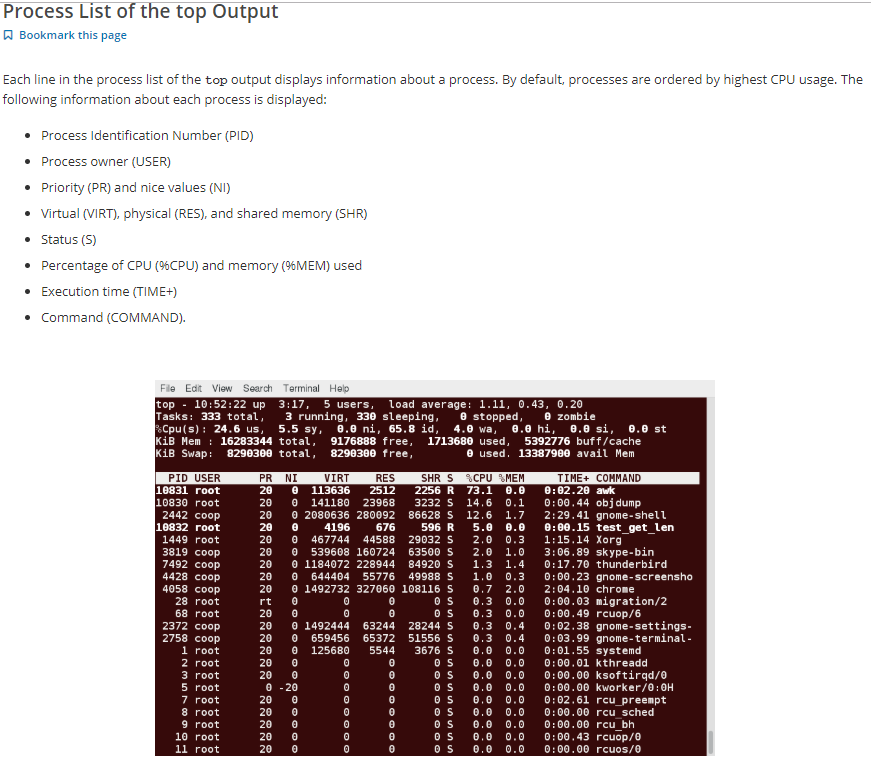


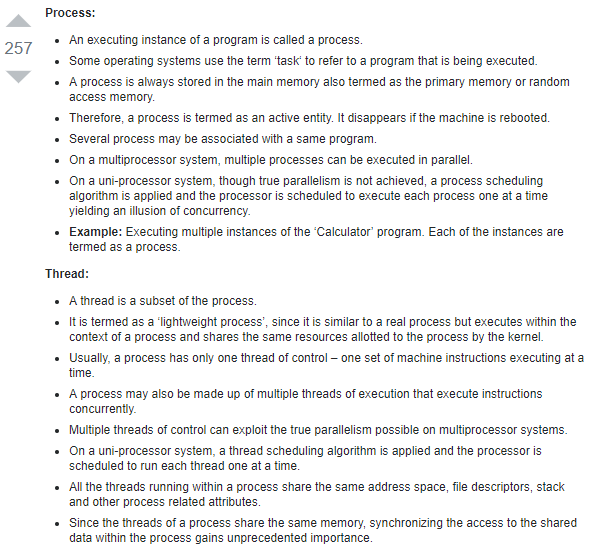


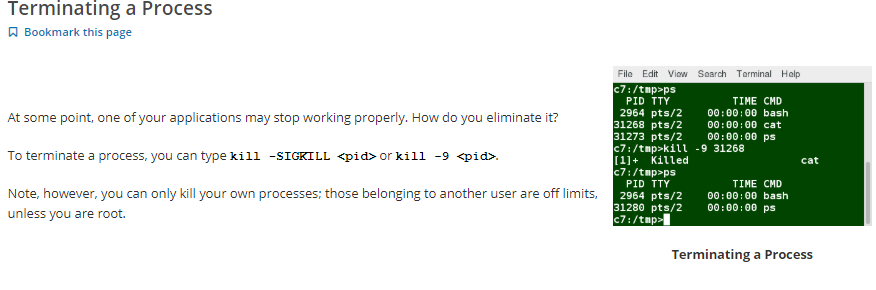


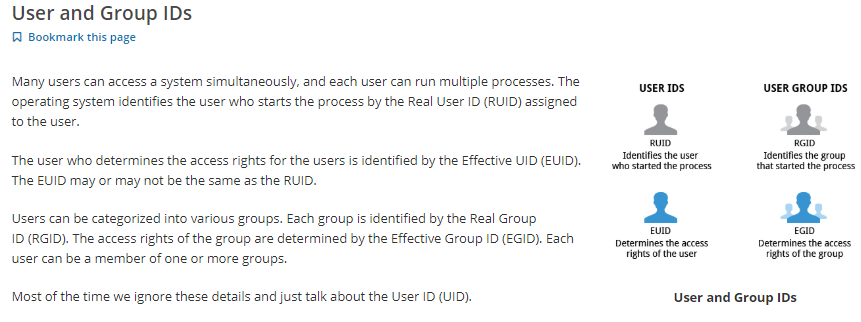












directories:

programs:

bin ----programs binary files

sbin ----programs binary files for su

lib----libraries of code used by programs

configuration :

etc---config files for the system and other miss stuff

init.d -----config files for booting linux

rc.d ------config files for booting linux

hardware :

dev-----device files for interfacing with disks and other hardware

mnt ----mountpoint directories that provides access to the disks

runtime files :

var ----files specific to this computer ,created and updated as system runs

log ----log files that trackimportatant system events ,containing error,warning

ls command

(List directory contents)

The ls command will list the files and directories within the current working directory (the directory you are currently in).

There are a few options you can use with ls, and the format, or syntax of the command is....

ls [options] [file]

EXAMPLE

>ls -l /home/rich/www

Lets break that down....

ls is the command

-l is the option which will give you a long listing format (which shows more info than just the file names - the owner, size, date last modified etc)

/home/rich/www is the directory we want to see a listing of (if you omit this part, ls will print the contents of the directory you are in).

Options

Some useful options are -l, -a, -s, -h and -R

-l will give you a long listing (as explained above)

-a will show you ALL the files in the directory, including hidden files

-R will the subdirectories recursively, which means it will show all the directories and files within the specified directory.

-s will also show you the size of the files (in blocks, not bytes)

-h will show the size in "human readable format" (ie: 4K, 16M, 1G etc). Of course you must use this option in conjunction with the -s option.

You can combine as many of these options as you wish.

EXAMPLES

>ls -la /home

Lists ALL the files and directories in the /home directory, in the long listing format.

>ls -ash

Lists ALL the files in the current directory (no directory was specified so it lists the contents of the current directory), and the size of the files/directories, written in 'human readable' format.

There are many more options for ls, but these are a few you may want to use. To see the total list of options for the ls command, you can type man ls at the prompt.

Back to command index

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME

rm - remove files or directories

SYNOPSIS

rm [OPTION]... FILE...

EXAMPLES

DESCRIPTION

This manual page documents the GNU version of rm. rm removes each specified file. By default, it does not remove directories.

If a file is unwritable, the standard input is a tty, and the -f or --force option is not given, rm prompts the user for whether to remove the file.

If the response does not begin with `y' or `Y', the file is skipped.

-d, --directory

unlink FILE, even if it is a non-empty directory (super-user only)

-f, --force

ignore nonexistent files, never prompt

-i, --interactive

prompt before any removal

-r, -R, --recursive

remove the contents of directories recursively

-v, --verbose

explain what is being done

--help

display this help and exit

--version

output version information and exit

To remove a file whose name starts with a `-', for example `-foo', use one of these commands:

rm -- -foo

rm ./-foo

Note that if you use rm to remove a file, it is usually possible to recover the contents of that file.

If you want more assurance that the contents are truly unrecoverable, consider using shred.

SEE ALSO

shred(1)

The full documentation for rm is maintained as a Texinfo manual. If the info and rm programs are properly installed at your site, the command

info rm

should give you access to the complete manual.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ln - creates links between files.

Description

ln creates a link to file TARGET with the name LINKNAME. If LINKNAME is omitted, a link to TARGET is created in the current directory,

using the name of TARGET as the LINKNAME.

ln creates hard links by default, or symbolic links if the -s (--symbolic) option is specified. When creating hard links, each TARGET must exist.

\*A link is an entry in your file system which connects a filename to the actual bytes of data on the disk. More than one filename can "link" to the same data.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

mput ----to ftp multiple files from local machine to the remote machine.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

vi editor :

To Get Into and Out Of vi

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

\* vi filename edit filename starting at line 1

vi -r filename recover filename that was being edited when system crashed

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

\* :x<Return> quit vi, writing out modified file to file named in original invocation

:wq<Return> quit vi, writing out modified file to file named in original invocation

:q<Return> quit (or exit) vi

\* :q!<Return> quit vi even though latest changes have not been saved for this vi call

Adding, Changing, and Deleting Text

Unlike PC editors, you cannot replace or delete text by highlighting it with the mouse. Instead use the commands in the following tables.

Perhaps the most important command is the one that allows you to back up and undo your last action. Unfortunately, this command acts like a toggle, undoing and redoing your most recent action. You cannot go back more than one step.

\* u UNDO WHATEVER YOU JUST DID; a simple toggle

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

\* i insert text before cursor, until <Esc> hit

I insert text at beginning of current line, until <Esc> hit

\* a append text after cursor, until <Esc> hit

A append text to end of current line, until <Esc> hit

\* o open and put text in a new line below current line, until <Esc> hit

\* O open and put text in a new line above current line, until <Esc> hit

Deleting Text

The following commands allow you to delete text.

\* x delete single character under cursor

Nx delete N characters, starting with character under cursor

dw delete the single word beginning with character under cursor

dNw delete N words beginning with character under cursor;

e.g., d5w deletes 5 words

D delete the remainder of the line, starting with current cursor position

\* dd delete entire current line

Ndd or dNd delete N lines, beginning with the current line;

e.g., 5dd deletes 5 lines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

to check running procceses o linux and unix:

ps -e -----it will display all the running process

ps -ag -----to see all running processes

ps -ef | grep (process name) ----to see the process id of particular process

ps aux ---to see the processes info with cpu utilization

kill pid -----to kill the current running process

kill 0 -------ti kill all running processes

top ----to see the current runing process with memory and other info with real time upadte

pstree--- to display the tree of processes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

syntax

cp sourcefile targetd

mv sourse target

strings ----uses :

TCS-Test1-unit0(active):/> which ls

/usr/bin/ls

TCS-Test1-unit0(active):/> strings /usr/bin/ls

SUNW\_OST\_OSCMD

RaAdC1hxmnlogrtucpFbqisfHL@eEvV

usage: ls -1RaAdCxmnlhogrtuvVcpFbqisfHLeE@ [files]

COLUMNS

total %d

ls: cycle detected for %s

total %llu

%llu

%10llu //\*it shows the readable ascii trings found in binary files\*//

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

$wc -l -------to list the word count of dir

$ls --help ------cmd to check usage of that command .

$ls --version ------will show cmd version

$echo arg -----to display the content which you want to print on the screen .

$man -ls ----to see the manual of the paticular cmd

$man -ls | less ----if the output is more than the page,we can pipeline by cmd less .type Q,to get out.

$sudo ls file name -----this cmd will run the cmd as super user and return to the original user, after executing cmd .

$echo $HOME --------------to check the home dir

$echo $DISPLAY ------------to check the display variable

echo Command

Use echo command to display text or value of variable.

echo [options] [string, variables...]

Displays text or variables value on screen.

Options

-n Do not output the trailing new line.

-e Enable interpretation of the following backslash escaped characters in the strings:

\a alert (bell)

\b backspace

\c suppress trailing new line

\n new line

\r carriage return

\t horizontal tab

\v Vertical tab

\\ backslash

combining cmd:(operatores)

Ampersand Operator (&)---Runs cmds in background.We can run multiple cmds using &.

It will run in different subcells.

Semi colon opertaor : cmd1;cmd2 ---- will run cmds sequentially

And operator(&&) :The AND Operator (&&) would execute the second command only, if the execution of first command SUCCEEDS,

i.e., the exit status of the first command is 0.

for eg:mtc@typhoon-base-unit0:/$ ls lrt && ./sumscript.sh

ls: lrt: No such file or directory

OR operator(||): The OR Operator (||) is much like an ‘else‘ statement in programming.

The above operator allow you to execute second command only if the execution of first command fails, i.e., the exit status of first command is ‘1‘

jobs :

$jobs ----list the jobs running in your current shell

$buinavapp & ------& causes the cmd to run as background job

$suspend -----suspend the current job and return to orignal shell

$cltr d or exit ---to get out of shell

basic file operation :

syn :ls --- option l,r,t,a,h

$ls -s | sort -n ----will show sort the files acc to there size.

cp :

cp file1 file 2 ----------sourse des

or cp file1 file2 file3 des dir

cp command main options:

option description

cp -a archive files

cp -f force copy by removing the destination file if needed

cp -i interactive - ask before overwrite

cp -l link files instead of copy

cp -L follow symbolic links

cp -n no file overwrite

cp -R recursive copy (including hidden files)

cp -u update - copy when source is newer than dest

cp -v verbose - print informative messages

mv:

$mv options source target --options i & f

$move file1 file2

rm :

rm options files | dir

rm -rf -----removes recursively forcefully

ln:

syn :ln options source target

a link is reference to another file.two types soft hand hard.soft links refers to another file by its path.

eg:$ ln -s myfile mysoftlink (options s,f,d ,,d for dir to carete a hard link...but you will need su for this )

mkdir:

syn:mkdir options dir

$mkdir dir1 dir2 dir3

$mkdir -p dir1/dir2/dir3 ----it will create parrent dir ...dir3 inside dir2 and dir2 in dir1 and dir1 itself

rmdir:

syn:rmdir /tmp/junk (options p)

file viewing :

cat --view files in their entirety

less ----one page at a a time

head ----first line of file

syntax:

head [options] [file(s)] ## will display the first ten lines of the file name##

head -n5 ---will print the 5 lines from starting

tail -----last line of file(options f ---keep the file open and watch activaely while another prog is writing to it)

syntax:

tail [options] [filenames] ## will display the last ten lines of the file name##

tail -5 filename ---will display the last 5 lines of file

tail +5 filename will print the content of file from 5th line

eg:tail -f /var/log/messages

strings :

syn:$strings /usr/bin/who ----to see the readable contents of binary files)

file:

syn :file option files

touch :

$ touch myfile (options -t & d ---t for time d for date )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

change owner :

chown :

syn :chown user\_spec file1 file2 file3

user\_spec maybe anything like:

$chown smith myfile ---to change owner of file

$chown smith:users myfile -----to cahnge file ownership to users

$chown :users myfile ------to set the owneship of group

chgrp:

syn :chgrp options group\_spec files

chmod:

file permission bit :binary 000 111 111 111 =chmod 777 ,,,which will give read,write and execute to all users.

sst rwx rwx rwx (consecuitvely for user ,group and others ,,,u,g,o,a--all)

+ ---to add permission

- -----to remove permission

= ------to set absolute permission ,ignoring existing ones

eg:

ug+rw ,,whould set read and write permissions for uesr and group

a-x ,,whould remove execute permission for eveyone

u=r ,,,would first remove all exixting permissions and then make file readable only to user.

$chmod u=rwx,g=rx,o=r myfile

1-the user can read, write, and execute it;

2-members of your group can read and execute it;

3- others may only read it.

file location :

find --locate files in a dir hier

syn : $ find location comparison-criteria search-term

$find / -type f -name myfile -print

$find / -type d -name

eg :$ find . -type d ----will print dir

$find . -type f ---will print files

TCS-Test1-unit0(active):/> find ./opt -name "images"

./opt/nortel/sam21em/html/images

./opt/nortel/NTsesm/gwchelp/images

./opt/nortel/NTsesm/servlet/XMLServlet/tmm/images

./opt/nortel/NTsesm/servlet/XMLServlet/images

./opt/nortel/NTsesm/www/lmmhelp/images

./opt/oracle/network/tools/images

./opt/oracle/uix/cabo/images

./opt/oracle/wwg/admin/images

./opt/oracle/jdk/jre/lib/images

./opt/NTnpm/www/npmhelp/images

./opt/apache/htdocs/images

./opt/tk/lib/tk8.3/demos/images

./opt/tk/lib/tk8.3/images

./opt/NTIpSec/www/images

./opt/jakarta/dist/tomcat/webapps/ipsec/images

./opt/usp/ntssgusp/jre/lib/images

./opt/SUNWlwact/misc/pacifico/images

which:

syn: which file

cmd tells you the location of the executable file.ex:

TCS-Test1-unit0(active):/bin> which mkdir

/usr/bin/mkdir

type:

syn:type file

it also tells the location of the files.for ex :

TCS-Test1-unit0(active):/bin> type mkpasswd which

mkpasswd is /usr/bin/mkpasswd

which is hashed (/usr/bin/which

file text manipulation :

grep:

syn: grep [options] pattern [files] ( option: n , i ,c )

-c : This prints only a count of the lines that match a pattern

-n : Display the matched lines and their line numbers.

-i : Ignores, case for matching

Syntax:

grep -i "string" FILE

ex: TCS-Test1-unit0(active):/var/log> grep -i "root" auditlog | less

Dec 9 03:45:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 04:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 05:15:03 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 06:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 06:45:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 07:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 08:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 09:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 10:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 10:45:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 11:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 12:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

Dec 9 13:15:04 TCS-Test1 root: "sync\_files: Cluster nodes out of Sync"

eg:TCS-Test1-unit0(active):/var/log> grep -i "root" \* .gz

//to search by wildcard ,,in the dir ending with .gz//

eg:TCS-Test1-unit0(active):/var/log> grep -i "root" \*

// searching all sub-dir in current dir //

egrep:(extended grep)

$ egrep -C 0 '\(f\|g\)ile' check\_file ----to search meta-character,when they are escaped.

output :(f|g)ile

fgrep:(fixed grep)

$ fgrep -C 0 '(f|g)ile' check\_file ------------to search meta-charcater when they are not escaped.

output :(f|g)ile

cut:

syn:cut -(b|c|f)range [options] [files] -----b-byte c-character and f--filed l

eg:

mtc@typhoon-base-unit0:/$ cat sname

1 devendra

2 krishna

3 anima

mtc@typhoon-base-unit0:/$ cut -d " " -f1 sname

1

2

3

mtc@typhoon-base-unit0:/$ cut -d " " -f2 sname

devendra

krishna

anima

eg:

TCS-Test1-unit0(active):/> cat vicky

devendra pratap singh

devendra pratap singh

devendra [pratap singhi

TCS-Test1-unit0(active):/> cut -b1 vicky

d

d

d

TCS-Test1-unit0(active):/> cut -c1 vicky

d

d

d

paste:

syn: paste [option] [files]

General Syntax of paste utility:

Syntax:

paste {file1} {file2}

Use of paste utility:

Putting lines together.

TCS-Test1-unit0(active):/> cat letter

a b c d e

TCS-Test1-unit0(active):/> cat numbers

1 2 2 3 4:

TCS-Test1-unit0(active):/> paste letter numbers

1 2 2 3 4:

a b c d e

Join:-

General Syntax of join utility:

Syntax:

join {file1} {file2}

Use of join utility:

The join utility joins, lines from separate files.

eg:

$join sname smark

11 Vivek 67

12 Renuka 55

13 Prakash 96

14 Ashish 36

15 Rani 67

tr:transform or translate

syn:tr [options] charset1 charset2

eg:

TCS-Test1-unit0(active):/> cat vicky | tr 'a' '@'

devendr@ pr@t@p singh

devendr@ pr@t@p singh

devendr@ [pr@t@p singhi

eg :

$ tr "[a-z]" "[A-Z]" < sname > cap\_names

$ cat cap\_names

VIVEK

ASHISH

ZEBRA

BABU

tr command is used to translate all lower case characters to upper-case letters.

It take input from sname file, and tr's output is redirected to cap\_names file.

awk(data manipulation):

Syntax:

awk 'pattern action' {file-name}

eg:

mtc@typhoon-base-unit0:/$ cat inv

egg order 4

cacke good 10

cheese okay 4

pen good 12

floppy good 5

mtc@typhoon-base-unit0:/$ awk '/good/ { print $3 }' inv

10

12

5

\*\*awk utility, select each record from file containing the word "good" and

performs the action of printing the third field (Quantity of available goods.)

uniq:

Syntax:

uniq {file-name}

eg:

mtc@typhoon-base-unit0:/$ cat uniqname

Devendra

Devendra

123

123

mtc@typhoon-base-unit0:/$ uniq uniqname

Devendra

123

##uniq cmd removes the duplicates,only in consuctive lines,,so if duliactes are not in same line;

then use sort with uniq.

for eg:

mtc@typhoon-base-unit0:/$ uniq uniqname

Dev Singh

123

Dev Singh

123

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ sort uniqname | uniq

123

Dev Singh

sort :

syn:sort [option] [files] [options ---n -numeric ,f--case insensitive sorting ]

eg:

TCS-Test1-unit0(active):/> cat vicky | sort -r

devendra pratap singh

devendra pratap singh

devendra [pratap singhi

TCS-Test1-unit0(active):/> cat vicky | sort

devendra [pratap singhi

devendra pratap singh

devendra pratap singh

file compression and packaging :

tar:

syn : tar -[options] tarfile name target dir or file

[options ---cvf --to create ,,tvf---to list content ,xvf--to extract content ,rvf --to add files or dir to exixting tar file

eg:tar -cvf dev.tar file1

TCS-Test1-unit0(active):/> tar -tvf dev.tar (tar.gz,tar.bz2)

tar: blocksize = 4

-rw-r--r-- 0/0 22 Jan 16 19:36 2017 file1

TCS-Test1-unit0(active):/> cat dev.tar

file1000064400000000000000000000000261303726341500120130ustar00rootroot00001250000002devendra pratap singh

TCS-Test1-unit0(active):/> tar -xvf dev.tar

tar: blocksize = 4

x file1, 22 bytes, 1 tape blocks

disks & filesystems :

df:disk free

syn:df [options]

eg:

TCS-Test1-unit0(active):/> df -k

Filesystem kbytes used avail capacity Mounted on

/dev/md/dsk/d2 9294317 2047445 7153929 23% /

/devices 0 0 0 0% /devices

ctfs 0 0 0 0% /system/contract

proc 0 0 0 0% /proc

mnttab 0 0 0 0% /etc/mnttab

swap 6750688 1568 6749120 1% /etc/svc/volatile

objfs 0 0 0 0% /system/object

sharefs 0 0 0 0% /etc/dfs/sharetab

TCS-Test1-unit0(active):/> df -h

Filesystem size used avail capacity Mounted on

/dev/md/dsk/d2 8.9G 2.0G 6.8G 23% /

/devices 0K 0K 0K 0% /devices

ctfs 0K 0K 0K 0% /system/contract

proc 0K 0K 0K 0% /proc

mnttab 0K 0K 0K 0% /etc/mnttab

swap 6.4G 1.5M 6.4G 1% /etc/svc/volatile

objfs 0K 0K 0K 0% /system/object

sharefs 0K 0K 0K 0% /etc/dfs/sharetab

/platform/SUNW,Netra-T5220/lib/libc\_psr/libc\_psr\_hwcap2.so.1

8.9G 2.0G 6.8G 23% /platform/sun4v/lib/libc\_psr.so.1

/platform/SUNW,Netra-T5220/lib/sparcv9/libc\_psr/libc\_psr\_hwcap2.so.1

8.9G 2.0G 6.8G 23% /platform/sun4v/lib/sparcv9/libc\_psr.so.1

fd 0K 0K 0K 0% /dev/fd

/dev/md/dsk/d8 5.9G 142M 5.7G 3% /var

swap 512M 6.6M 505M 2% /tmp

MOUNT:

syn: mount device dir

ex : mount /dev/sda1 /mnt/vicky

this will mount block sda1 to the dir vicky inside mnt...so that it will be accesible

Mount CDROM

If you do not belong into cdrom group and thus do not posses permissions to mount CDROM, switch to root user and execute a command below:

# mount -t iso9660 /dev/scd0 /media/cdrom

mount: block device /dev/sr0 is write-protected, mounting read-only

unmount:

syn: unmount dir

eg: unmount /dev/sda1

to unmount all mounted devices :unmount -a

shutdown:

syntax: shutdown [options] time [messege]

eg:shutdown -h +2 "maintinance"

here time is in min.

options ----r ---reboot h -halt c-cancel the shutdown

logname:

root@typhoon-base-unit0:/root> logname

logname: no login name

whoami:

root@typhoon-base-unit0:/root> whoami

root

who:

root@typhoon-base-unit0:/root> who

root ttyS5 Feb 2 20:32

users:

root@typhoon-base-unit0:/root> users

root

passwd:

syn: passwd [options] [username]

#passwd dev

sudo:superuser do ----gives access to cmds,which dont have root,access

sudo gives the root perssions and do the task

eg: when we run cmd ---apt-get update it gives error

while when we give sudo option L:sudo apt-get update then it does the work.

uname:gives info about system ,,,like operting system and version

root@typhoon-base-unit0:/root> uname -a

Linux typhoon-base-unit0 2.6.14-pne #1 SMP Fri Oct 17 21:22:30 EDT 2014 i686 GNU/Linux

to know the RAM :

cat /proc/meminfo

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

types of shells:

BASH ( Bourne-Again SHell ) Brian Fox and Chet Ramey

Free Software Foundation

Most common shell in Linux. It's Freeware shell.

CSH (C SHell) Bill Joy University of California (For BSD) The C shell's syntax and usage are very similar to

the C programming language.

KSH (Korn SHell) David Korn AT & T Bell Labs

--

TCSH See the man page.

Type $ man tcsh

--

TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

Tip: To find all available shells in your system type following command:

$ cat /etc/shells

Tip: To find your current shell type following command

$ echo $SHELL

cmd to change shell :chsh

my 1st shell script:

to start a shell script we will 1st crate a script file using vi and will start writing our cmds atfer she-bang.

Shebang is "#!".various examples of shebang is as below :

#!/bin/sh – Execute the file using the Bourne shell, or a compatible shell, with path /bin/sh

#!/bin/bash – Execute the file using the Bash shell.

#!/bin/csh -f – Execute the file using csh, the C shell, or a compatible shell, and suppress the execution of the user’s .cshrc file on startup

#!/usr/bin/perl -T – Execute using Perl with the option for taint checks

#!/usr/bin/env python – Execute using Python by looking up the path to the Python interpreter automatically via env

ex:sample script for name display

vi script.sh

#! /bin/sh

echo "devendra Pratap singh"

~

TCSF-CBM-unit0# chmod 777 script.sh

TCSF-CBM-unit0#

TCSF-CBM-unit0#

TCSF-CBM-unit0#

TCSF-CBM-unit0# ./script.sh

devendra Pratap singh

shell varibale: we can also write scrip using variable.for ex

CSF-CBM-unit0# vi script.sh

"script.sh" 5 lines, 89 characters

#! /bin/sh

#name display program

echo "what is your name"

read name

echo "welcome $name" #here name is variable which will contain the input data like name

TCSF-CBM-unit0# ./script.sh

what is your name

dev

welcome dev

another varible script:

vi varscript.sh

#! /bin/sh

#variable name script

name="devendra prtap singh" #here quotes are important otherwise shell will not understand.

echo $name

TCSF-CBM-unit0# chmod 777 varscript.sh

TCSF-CBM-unit0#

TCSF-CBM-unit0#

TCSF-CBM-unit0#

TCSF-CBM-unit0# ./varscript.sh

devendra prtap singh

VARIABLES IN SHELL:

`n Linux (Shell), there are two types of variable:

(1) System variables - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.

(2) User defined variables (UDV) - Created and maintained by user. This type of variable defined in lower letters.

You can see system variables by giving command like "$ set", some of the important System variables are:

System Variable

Meaning

BASH=/bin/bash Our shell name

BASH\_VERSION=1.14.7(1) Our shell version name

COLUMNS=80 No. of columns for our screen

HOME=/home/vivek Our home directory

LINES=25 No. of columns for our screen

LOGNAME=students students Our logging name

OSTYPE=Linux Our Os type

PATH=/usr/bin:/sbin:/bin:/usr/sbin Our path settings

PS1=[\u@\h \W]\$ Our prompt settings

PWD=/home/students/Common Our current working directory

SHELL=/bin/bash Our shell name

USERNAME=vivek User name who is currently login to this PC

You can print any of the above variables contains as follows:

$ echo $USERNAME

$ echo $HOME

USER defined variables:

Syntax:

variable name=value

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

Example:

$ no=10# this is ok

$ 10=no# Error, NOT Ok, Value must be on right side of = sign.

To define variable called 'vech' having value Bus

$ vech=Bus

To define variable called n having value 10

$ n=10

Rules for Naming variable name (Both UDV and System Variable):

(1) Variable name must begin with Alphanumeric character or underscore character (\_), followed by one or more Alphanumeric character. For e.g. Valid shell variable are as follows

HOME

SYSTEM\_VERSION

vech

no

(2) Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In following variable declaration there will be no error

$ no=10

But there will be problem for any of the following variable declaration:

$ no =10

$ no= 10

$ no = 10

(3) Variables are case-sensitive, just like filename in Linux. For e.g.

$ no=10

$ No=11

$ NO=20

$ nO=2

Above all are different variable name, so to print value 20 we have to use $ echo $NO and not any of the following

$ echo $no # will print 10 but not 20

$ echo $No# will print 11 but not 20

$ echo $nO# will print 2 but not 20

(4) You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.

$ vech=

$ vech=""

Try to print it's value by issuing following command

$ echo $vech

Nothing will be shown because variable has no value i.e. NULL variable.

(5) Do not use ?,\* etc, to name your variable names.

Read-only variables:

# if we had defined any varibales as read-only then its value can't be changed.script will give error.

eg:mtc@typhoon-base-unit0:/$ sh -v read.sh

#! /bin/sh

name=Devendra

readonly name

name=vicky

read.sh: line 4: name: readonly variable

Unsetting varibales name:

syn: unset variablename

Linux arrays:

syn: array\_name[index]=value

accessing array values:

${array\_name[index]}

# we can access all array values by giving "\*" or "@".

eg:

#! /bin/sh

#array initialization

Name[0]=Dev

Name[1]=vicky

Name[2]=Devendra

echo -e "first name is ${Name[0]}"

echo -e "Second name is ${Name[1]}"

echo -e "third number is ${Name[2]}"

echo -e "all names are ${Name[\*]}"

output:

mtc@typhoon-base-unit0:/$ ./array.sh

first name is Dev

Second name is vicky

third number is Devendra

all names are Dev vicky Devendra

script two add two numbers: ## we use expr(external program) for opertaion like addition,substraction etc.

#! /bin/sh

# sum script for two no

echo "enter two number"

read num1 num2

sum=$(expr $num1 + $num2)

echo "the sum is $sum"

script to devide two numbers:

#! /bin/sh

# sum script for two no

echo "enter two number"

read num1 num2

dev=$(expr $num1 / $num2)

echo "the devision is $dev"

Note: while doing echo of expr pute it in backquote(` `),otherwise output will not be printed.

for ex:

$ echo "expr 6 + 3" # It will print expr 6 + 3

$ echo 'expr 6 + 3' # It will print 9

here are three types of quotes

Quotes

Name

Meaning

" Double Quotes "Double Quotes" - Anything enclose in double quotes removed meaning of that characters (except \ and $).

' Single quotes 'Single quotes' - Enclosed in single quotes remains unchanged.

` Back quote `Back quote` - To execute command

exit status:

if we run any cmd and it runs successfully then it will return value 0 ,otherwise it will return non-zero value.

To know it type "$?" and check the response.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The read Statement

Use to get input (data from user) from keyboard and store (data) to variable.

Syntax:

read variable1, variable2,...variableN

Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable fname.

$ vi sayH

#

#Script to read your name from key-board

#

echo "Your first name please:"

read fname

echo "Hello $fname, Lets be friend!"

Run it as follows:

$ chmod 755 sayH

$ ./sayH

Your first name please: vivek

Hello vivek, Lets be friend!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Wild cards (Filename Shorthand or meta Characters)

Wild card /Shorthand Meaning Examples

\* Matches any string or group of characters. $ ls \* will show all files

$ ls a\* will show all files whose first name is starting with letter 'a'

$ ls \*.c will show all files having extension .c

$ ls ut\*.c will show all files having extension .c but file name must begin with 'ut'.

? Matches any single character. $ ls ? will show all files whose names are 1 character long

$ ls fo? will show all files whose names are 3 character long and file name begin with fo

[...] Matches any one of the enclosed characters $ ls [abc]\* will show all files beginning with letters a,

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Command line arguments:

Try to determine command and arguments from following commands

$ ls foo

$ cp y y.bak

$ mv y.bak y.okay

$ tail -10 myf

$ mail raj

$ sort -r -n myf

$ date

$ clear

Answer:

Command No. of argument to this command (i.e $#) Actual Argument($\*or $@)

ls 1 foo

cp 2 y and y.bak

mv 2 y.bak and y.okay

tail 2 -10 and myf

mail 1 raj

sort 3 -r, -n, and myf

date 0

clear 0

$# holds number of arguments specified on command line. And $\* or $@ refer to all arguments passed to script.

$ myshell foo bar

Shell Script name i.e. myshell

First command line argument passed to myshell i.e. foo

Second command line argument passed to myshell i.e. bar

In shell if we wish to refer this command line argument we refer above as follows

myshell it is $0

foo it is $1

bar it is $2

Here $# (built in shell variable ) will be 2 (Since foo and bar only two Arguments), Please note at a time such 9 arguments can be used from $1..$9,

You can also refer all of them by using $\* (which expand to `$1,$2...$9`).

Note that $1..$9 i.e command line arguments to shell script is know as "positional parameters".

Variables & Description:

$0 ---The filename of the current script.

$n ----These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position

of an argument (the first argument is $1, the second argument is $2, and so on).

$#----The number of arguments supplied to a script.

$\*-----All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2.

$?-----The exit status of the last command executed.

$$ ----The process number of the current shell. For shell scripts, this is the process ID under which they are executing.

$!-----The process number of the last background command.

ex:

mtc@typhoon-base-unit0:/$ ./demo dev singh

Total number of command line argument are 2

./demo is script name

dev is first argument

singh is second argument

All of them are :- dev singh or dev singh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Input-output redirection:

Mostly all command gives output on screen or take input from keyboard, but in Linux (and in other OSs also) it's possible to send output to file or to read input from file.

For e.g.

$ ls command gives output to screen; to send output to file of ls command give command

$ ls > filename

It means put output of ls command to filename.

There are three main redirection symbols >,>>,<

(1) > Redirector Symbol

Syntax:

Linux-command > filename

To output Linux-commands result (output of command or shell script) to file. Note that if file already exist, it will be overwritten else new file is created. For e.g. To send output of ls command give

$ ls > myfiles

Now if 'myfiles' file exist in your current directory it will be overwritten without any type of warning.

(2) >> Redirector Symbol

Syntax:

Linux-command >> filename

To output Linux-commands result (output of command or shell script) to END of file. Note that if file exist , it will be opened and new information/data will be written to END of file, without losing previous information/data, And if file is not exist, then new file is created. For e.g. To send output of date command to already exist file give command

$ date >> myfiles

(3) < Redirector Symbol

Syntax:

Linux-command < filename

To take input to Linux-command from file instead of key-board. For e.g. To take input for cat command give

$ cat < myfiles

eg:

mtc@typhoon-base-unit0:/$ cat name

rohan

akshay

krishna

anima

devendra

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ cat name>myfile

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ date >>myfile

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ cat myfile

rohan

akshay

krishna

anima

devendra

Thu May 17 16:44:12 IST 2018

mtc@typhoon-base-unit0:/$ sort<myfile>sortednames

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ cat sortednames

Thu May 17 16:44:12 IST 2018

akshay

anima

devendra

krishna

rohan

filters:from using head and tail cmds we can filter the content

eg:

mtc@typhoon-base-unit0:/$ cat myfile

rohan

akshay

krishna

anima

devendra

mtc@typhoon-base-unit0:/$ tail +2<myfile | head -5>tailfile

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ cat tailfile

akshay

krishna

anima

devendra

## we can see here tail will print the value from start 2nd line of myfile and head will print till 5th line.

Will print the total value in tailfile.

Structured language constructs(Decision making and loops):

IF condition:

if condition which is used for decision making in shell script, If given condition is true then command1 is executed.

Syntax:

if condition

then

command1 if condition is true or if exit status

of condition is 0 (zero)

...

...

fi

ex:

#! /bin/sh

## if statement

echo "please provide two numbers"

read a b

if [ $a -lt $b ]

then

echo "a is less than b"

else

echo "a is greter than b"

fi

Test command or [expr]:

est command or [ expr ] is used to see if an expression is true, and if it is true it return zero(0),

otherwise returns nonzero for false.

Syntax:

test expression OR [ expression ]

ex:

#! /bin/sh

## if statement

echo "please provide two numbers"

read a b

if test $a -lt $b

then

echo "a is less than b"

else

echo "a is greter than b"

fi

Mathematics operator:

Operator meaninig statement

-eq eqvals to 5==6

-ne not eqvals to 5!=6

-lt less than 5<6

-le less than or eqvals to 5<=6

-gt greater than 5>6

-ge greater than or eqvals to 5>=6

\*we dont use square bracket in test cmd.

String comparisions:

Operator Description Example

= Checks if the value of two operands are equal or not; if yes, then the condition becomes true.

!= Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.

-z Checks if the given string operand size is zero; if it is zero length, then it returns true.

-n Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.

str Checks if str is not the empty string; if it is empty, then it returns false.

Test for file and directories:

Test Meaning

-s file Non empty file

-f file Is File exist or normal file and not a directory

-d dir Is Directory exist and not a file

-w file Is writeable file

-r file Is read-only file

-x file Is file is executable

Logical operators:

Logical operators are used to combine two or more condition at a time

Operator Meaning

! expression Logical NOT

expression1 -a expression2 Logical AND

expression1 -o expression2 Logical OR

Logical loops:

if...else...fi :

syntax:

f given condition is true then command1 is executed otherwise command2 is executed.

Syntax:

if condition

then

condition is zero (true - 0)

execute all commands up to else statement

else

if condition is not true then

execute all commands up to fi

fi

ex:

!/bin/sh

# Netsed if else script

echo "enter two values"

read a b

if [ $a -eq $b ]

then

echo "value a is eqval to b "

else

if [$a -lt $b ]

then

echo "a is less than b "

else

echo "a is greater than b"

fi

fi

Mutltilevel if-then-else:(if -elif -else)

Syntax:

if condition

then

condition is zero (true - 0)

execute all commands up to elif statement

elif condition1

then

condition1 is zero (true - 0)

execute all commands up to elif statement

elif condition2

then

condition2 is zero (true - 0)

execute all commands up to elif statement

else

None of the above condtion,condtion1,condtion2 are true (i.e.

all of the above nonzero or false)

execute all commands up to fi

fi

ex:

!/bin/sh

# Netsed if-elif-else script

echo "enter two values"

read a b

if [ $a -eq $b ];then

echo "a is eqval to b"

elif [ $a -lt $b ];then

echo "a is not eqval to b"

elif [ $a -gt $b ];then

echo "a is greater than b"

else

echo "Values are not integers ,please provide numbers"

fi

Output:

mtc@typhoon-base-unit0:/$ ./ifst.sh

./ifst.sh: line 2: !/bin/sh: No such file or directory

enter two values

10 20

a is not eqval to b

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ ./ifst.sh

./ifst.sh: line 2: !/bin/sh: No such file or directory

enter two values

20 10

a is greater than b

mtc@typhoon-base-unit0:/$

mtc@typhoon-base-unit0:/$ ./ifst.sh

./ifst.sh: line 2: !/bin/sh: No such file or directory

enter two values

10 10

a is eqval to b

mtc@typhoon-base-unit0:/$ ./ifst.sh

./ifst.sh: line 2: !/bin/sh: No such file or directory

enter two values

dev s=

./ifst.sh: line 6: [: dev: integer expression expected

./ifst.sh: line 8: [: dev: integer expression expected

./ifst.sh: line 10: [: dev: integer expression expected

Values are not integers ,please provide numbers

Loops :

bash supports:

for loop

while loop

Note that in each and every loop,

(a) First, the variable used in loop condition must be initialized, then execution of the loop begins.

(b) A test (condition) is made at the beginning of each iteration.

(c) The body of loop ends with a statement that modifies the value of the test (condition) variable.

FOR LOOP:

syntax:

Syntax:

for { variable name } in { list }

do

execute one for each item in the list until the list is

not finished (And repeat all statement between do and done)

done

ex:

#!/bin/sh

for i in 1 2 3 4 5

do

echo "Looping ... number $i"

done

output:

mtc@typhoon-base-unit0:/$ ./for.sh

Looping ... number 1

Looping ... number 2

Looping ... number 3

Looping ... number 4

Looping ... number 5

eg:script for printing tables upto 10,if value will be gt than 10 then error will be print.

#!/bin/sh

echo " please provide one number for tables "

read n1

if [ $n1 -gt 10 ];then

echo "Number is greter than 10"

else

for number in 1 2 3 4 5 6 7 8 9 10

do

echo "table of $n1 is `expr $n1 \\* $number`"

done

fi

eg: if and for loop for tables of any given number

#! /bin/sh

##script for table of argument

if [ $# -eq 0 ];then

echo -e "$0: please provide argument"

else

for number in 1 2 3 4 5 6 7 8 9 10

do

echo -e "table for the given arguemnt is `expr $1 \\* $number`"

done

fi

for loop can also be writen as below:

Syntax:

for (( expr1; expr2; expr3 ))

do

.....

...

repeat all statements between do and

done until expr2 is TRUE

Done

\*\*\*\*\*\*\*In above syntax BEFORE the first iteration, expr1 is evaluated. This is usually used to initialize variables for the loop.

All the statements between do and done is executed repeatedly UNTIL the value of expr2 is TRUE.

AFTER each iteration of the loop, expr3 is evaluated. This is usually use to increment a loop counter.\*\*\*\*\*\*\*\*\*\*\*\*\*

eg:

#! /bin/sh

##script for table of argument

if [ $# -eq 0 ];then

echo -e "$0: please provide argument"

else

for (( i=1 ; i<=10 ; i++ ))

do

echo -e "table for the given arguemnt is `expr $1 \\* $i`"

done

fi

\*\*\*here 1st expr is initilization of value and second is condtion and third one is counter increment.

output will be:

mtc@typhoon-base-unit0:/$ ./for.sh 5

table for the given arguemnt is 5

table for the given arguemnt is 10

table for the given arguemnt is 15

table for the given arguemnt is 20

table for the given arguemnt is 25

table for the given arguemnt is 30

table for the given arguemnt is 35

table for the given arguemnt is 40

table for the given arguemnt is 45

table for the given arguemnt is 50

Nested for loop:

#! /bin/sh

for (( i = 1; i <= 5; i++ )) ### Outer for loop ###

do

for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###

do

echo -n "$i "

done

echo -e "\n"

done

output:

mtc@typhoon-base-unit0:/$ ./nestedfor.sh

1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

4 4 4 4 4

5 5 5 5 5

While loop:

Syntax:

while [ condition ]

do

command1

command2

command3

..

....

done

eg:

#! /bin/sh

#while loop script for multiplication

echo "please provide the number"

read n

i=1

while [ $i -le 10 ]

do

echo -e "multiplication of the number is `expr $i \\* $n`"

i=`expr $i + 1`

done

output:

mtc@typhoon-base-unit0:/$ ./while.sh

please provide the number

6

multiplication of the number is 6

multiplication of the number is 12

multiplication of the number is 18

multiplication of the number is 24

multiplication of the number is 30

multiplication of the number is 36

multiplication of the number is 42

multiplication of the number is 48

multiplication of the number is 54

multiplication of the number is 60

Break and continue statement:

Break statement:

#break statement is used to come out of loop.

syn: break n ----where n is nth eclosing loop.

eg:

#! /bin/sh

#break statement

n=0

while [ $n -le 10 ]

do

echo -e "$n"

n=`expr $n + 1`

if [ $n -gt 10 ];

then

break

fi

done

output:

0

1

2

3

4

5

6

7

8

9

10

continue :

The continue statement is similar to the break command, except that it causes the current iteration of the loop to exit,

rather than the entire loop.

This statement is useful when an error has occurred

but you want to try to execute the next iteration of the loop.

eg :

#!/bin/sh

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS

do

Q=`expr $NUM % 2`

if [ $Q -eq 0 ]

then

echo "Number is an even number!!"

continue

fi

echo "Found odd number"

done

output:

Found odd number

Number is an even number!!

Found odd number

Number is an even number!!

Found odd number

Number is an even number!!

Found odd number

Case statement:

Syntax:

case $variable-name in

pattern1) command

...

..

command;;

pattern2) command

...

..

command;;

patternN) command

...

..

command;;

\*) command

...

..

command;;

esac

The $variable-name is compared against the patterns until a match is found. The shell then executes all t

he statements up to the two semicolons that are next to each other. The default is \*) and its executed if no match is found.

eg:

#! /bin/sh

#shell script for wehicle rental price

if [ -z $1 ];then

rental="\*\*\* Unknown vehicle \*\*\*"

elif [ -n $1 ];then

# otherwise make first arg as rental

rental=$1

fi

case $rental in

"car") echo "For $rental Rs.20 per k/m";;

"van") echo "For $rental Rs.10 per k/m";;

"jeep") echo "For $rental Rs.5 per k/m";;

"bicycle") echo "For $rental 20 paisa per k/m";;

\*) echo "Sorry, I can not gat a $rental for you";;

esac

output:

mtc@typhoon-base-unit0:/$ ./case.sh van

For van Rs.10 per k/m

Debugging shell script:

While programming shell sometimes you need to find the errors (bugs) in shell script and

correct the errors (remove errors - debug). For this purpose you can use -v and -x option with sh or bash command to debug the shell script. General syntax is as follows:

Syntax:

sh option { shell-script-name }

OR

bash option { shell-script-name }

Option can be

-v Print shell input lines as they are read.

-x After expanding each simple-command, bash displays the expanded value of PS4 system variable, followed by the command and its expanded arguments.

Shell quoting mechanism:

\* ? [ ] ' " \ $ ; & ( ) | ^ < > new-line space tab

character may be quoted (i.e., made to stand for itself) by preceding it with a \.

eg:

#!/bin/sh

echo Hello; Word

will give you error.cmd not found.But after using backslash (\).

!/bin/sh

echo Hello\; Word

output:

echo Hello world.

Quoting & Description

1

Single quote

All special characters between these quotes lose their special meaning.

2

Double quote

Most special characters between these quotes lose their special meaning with these exceptions -

$

`

\$

\'

\"

\\

3

Backslash

Any character immediately following the backslash loses its special meaning.

4

Back quote

Anything in between back quotes would be treated as a command and would be executed.

Advance shell scripting:

/dev/null - Use to send unwanted output of program

This is special Linux file which is used to send any unwanted output from program/command.

Syntax:

command > /dev/null

Example:

$ ls > /dev/null

Output of above command is not shown on screen its send to this special file.

The /dev directory contains other device files. The files in this directory mostly represent peripheral

devices such disks like floppy disk, sound card, line printers etc. See the file system tutorial for more information on Linux disk, partition and file system.

Future Point:

Run the following two commands

$ ls > /dev/null

$ rm > /dev/null

Local and Global Shell variable (export command)

Normally all our variables are local. Local variable can be used in same shell, if you load another copy of shell (by typing the /bin/bash at the $ prompt) then new shell ignored all old shell's variable. For e.g. Consider following example

$ vech=Bus

$ echo $vech

Bus

$ /bin/bash

$ echo $vech

NOTE:-Empty line printed

$ vech=Car

$ echo $vech

Car

$ exit

$ echo $vech

Bus

Command Meaning

$ vech=Bus Create new local variable 'vech' with Bus as value in first shell

$ echo $vech Print the contains of variable vech

$ /bin/bash Now load second shell in memory (Which ignores all old shell's variable)

$ echo $vech Print the contains of variable vech

$ vech=Car Create new local variable 'vech' with Car as value in second shell

$ echo $vech Print the contains of variable vech

$ exit Exit from second shell return to first shell

$ echo $vech Print the contains of variable vech (Now you can see first shells variable and its value)

Global shell defined as:

"You can copy old shell's variable to new shell (i.e. first shells variable to seconds shell), such variable is know as Global Shell variable."

To set global varible you have to use export command.

Syntax:

export variable1, variable2,.....variableN

Examples:

$ vech=Bus

$ echo $vech

Bus

$ export vech

$ /bin/bash

$ echo $vech

Bus

$ exit

$ echo $vech

Bus

Command Meaning

$ vech=Bus Create new local variable 'vech' with Bus as value in first shell

$ echo $vech Print the contains of variable vech

$ export vech Export first shells variable to second shell i.e. global varible

$ /bin/bash Now load second shell in memory (Old shell's variable is accessed from second shell, if they are exported )

$ echo $vech Print the contains of variable vech

$ exit Exit from second shell return to first shell

$ echo $vech Print the contains of variable vech

Conditional execution i.e. && and ||

The control operators are && (read as AND) and || (read as OR). The syntax for AND list is as follows

Syntax:

command1 && command2

command2 is executed if, and only if, command1 returns an exit status of zero.

The syntax for OR list as follows

Syntax:

command1 || command2

command2 is executed if and only if command1 returns a non-zero exit status.

You can use both as follows

Syntax:

command1 && comamnd2 if exist status is zero || command3 if exit status is non-zero

if command1 is executed successfully then shell will run command2 and if command1 is not successful then command3 is executed.

Example:

$ rm myf && echo "File is removed successfully" || echo "File is not removed"

functions:

Function is series of instruction/commands. Function performs particular activity in shell i.e.

it had specific work to do or simply say task. To define function use following syntax:

Syntax:

function-name ( )

{

command1

command2

.....

...

commandN

return

}

ex:

mtc@typhoon-base-unit0:~$ sayhello()

> { echo "please provide your name"

> read name

> echo "welcome $name to test lab"

> return

> }

output:

mtc@typhoon-base-unit0:~$

mtc@typhoon-base-unit0:~$ sayhello

please provide your name

dev

welcome dev to test lab

mtc@typhoon-base-unit0:~$

Note:

functions expires after current seesion.So to make it alive through all session,add your script to

/etc/bashrc file.it needs su-.logout and run funtion.

User interface and dailog utility:

Script to print menu and choosing option:

#! /bin/sh

#Menu item display

while : ## it will start an infinite loop and will terminate only when case 5"

do

clear

echo "-----------------------"

echo "Main menu"

echo "-----------------------"

echo "[1] Show Todays date/time"

echo "[2] Show files in current directory"

echo "[3] Show calendar"

echo "[4] Start editor to write letters"

echo "[5] Exit/Stop"

echo "-----------------------"

echo -e "Enter your choice [1-5]"

read choice

case $choice in

"1")echo -e " today date is `date`";;

"2")echo -e "files in current dir are `pwd | ls -lart`";;

"3")echo -e "calander for the month is `cal`";;

"4")vi;;

"5")exit 0;;

\*) echo -e "please choose the correct option"

esac

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sample examples\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ex1

How to write shell script that will add two nos, which are supplied as command line argument, and if this two nos are not given show error and its usage

sh :#! /bin/sh

#script for adding two number and if argument is missing then printing error

#here prog begins

if [ $# -ne 2 ];then

echo -e "Argument is missing,usage :$0 arg1 arg2"

else

echo -e "addition of two numbers are :`expr $1 + $2`"

fi

~

ex2:

rite Script to find out biggest number from given three nos. Nos are supplies as command line argument. Print error if sufficient arguments are not supplied.

sh: #! /bin/sh

#script for finding out biggest no in given three no

#here starts

if [ $1 -gt $2 ] && [ $1 -gt $3 ];then

echo -e "Biggest no is:$1"

elif [ $2 -gt $1 ] && [ $2 -gt $3 ];then

echo -e "Biggest no is:$2"

elif [ $3 -gt $1 ] && [ $3 -gt $2 ];then

echo -e "Biggest no is:$3"

elif [ $# -ne 3 ];then

echo "Argument is missing,please provide argument"

fi

ex3:Write script to print nos as 5,4,3,2,1 using while loop

sh:

#! /bin/sh

i=5

while [ $i -gt 0 ]

do

echo -e "numbers are :$i"

i=`expr $i - 1`

done

ex:4 Write Script, using case statement to perform basic math operation as

follows

+ addition

- subtraction

x multiplication

/ division

The name of script must be 'q4' which works as follows

$ ./q4 20 / 3, Also check for sufficient command line arguments

sh:

#! /bin/sh

#script for math operations

if [ $# -eq 3 ];then

case $2 in

+)echo -e "the addition of the two numbers is:`expr $1 + $3`";;

-)echo -e "the substration of the two number is:`expr $1 - $3`";;

\\*) echo -e "the multiplication of two number is:`expr $1 \\* $3`";;

/)echo -e "the devision of the two number is:`expr $1 / $3`";;

\*)echo -e "Operator is missing";;

esac

else

echo -e "Arguments and oprator are missing"

echo -e "Please provide the Argument and operator as:$0 $1 (+,-,\\*,/) $2"

fi

ex5:

Write script to determine whether given file exist or not, file name is supplied

as command line argument, also check for sufficient number of command line argument

sh: #! /bin/sh

#sc to search the file

if [ $# -ne 1 ];then

echo -e "usage $0 file name"

exit 1

fi

if [ -f $1 ];then

echo -e "\n file is present:$1"

else

echo -e "\n file is not present"

fi

ex:6 Q.19. Write shell script to show various system configuration like

1) Currently logged user and his logname

2) Your current shell

3) Your home directory

4) Your operating system type

5) Your current path setting

6) Your current working directory

7) Show Currently logged number of users

8) About your os and version ,release number , kernel version

9) Show all available shells

10) Show mouse settings

11) Show computer cpu information like processor type, speed etc

12) Show memory information

13) Show hard disk information like size of hard-disk, cache memory, model etc

14) File system (Mounted)

sh:#!/bin/sh

#script for checking system configuration

clear

echo -e "\n Here are the system configuration"

echo -e "\n Current logged in users and numbers of users are:`who`and `who | wc -l`"

echo -e "\n Current shell is `$SHELLS`"

echo -e "\n Home directory is \n `$HOME`"

echo -e "\n OS type is \n `$OSTYPE`"

echo -e "\n Current path is \n `$PATH`"

echo -e "\n Current working dir id \n `pwd`"

Linux boot process:

\*BIOS is the 1st piece of software that run.BIOS have may options of boot device.Once it found bootable device,it load boot loaders.

\*initrd(initial ram disk) is the temporary file system,which is loaded into memory.it mounts the filesystem of os.

\*there are two types of boot loaders.

LILO (LINUX LOADER) AND GRUB(GRAND UNIFIED LOADER)

\*Primary purpose of the boot loader is to start os.Necessary files for loading os reside on /boot.It contains below:

-contains the files to boot the os

-initrd

-kernal

-boot loader config

\*Kernal ring buffer contains the messege from kernel.We can check these messeges.Log reside in **/var/log/dmesg .**

**Cmd – dmesg**

System runlevels:

0 – shuts down system

1. Single user mode
2. Multi-user mode with gui(debian/Ubuntu)
3. Multi-user text mode
4. Undefined
5. Multi-user mode with gui(redhat/centos)
6. Reboot

System shutdown:

Cmd :

Shutdown -r 15:30 “REBOOTING”

Shutdown –r +5 “rebooting soon”

Shutdown –r now

* Earliar system runlevel was controlled by inittab,but now it is controlled by systemd.
* We can check this under /lib/systemd/system.
* To check the default config of system -
* **cmd : systemctl get-default**
* T0 set system runlevels: **set-default graphical.target.**
* To change the system run level :systemctl isolate **graphical.target.**

**System logging:**

\*linux uses syslog standard for message logging.A system logger records all messegses created by applications and programs.

\*Allows logging to be centrally controlled and eliminates the need to kept log separately for each application.

\* Uses facilities and severities to categorize message. each message is labeled with facility code and severity lebels.

\*Examples of facilities include Kern,user,mail,daemon ,auth,syslog .Each facility has a number and keyword associated with message.

\*Examples of the severity include Emergency,alert,critical,warning,notice,info,dubug.

**Syslog servers**:

\*Processes syslog messages based on the rules.

\*Earlier syslogd was used but now most of the linux uses rsyslog and syslog-ng.

**Rsyslog:**

\*config files for rsyslog reside in /etc/rsyslog.conf .

**Logging rules:**

\*logging rules composed by two fields selector field (facility.severity)and action field (Determines how a message is processed,)

\*For ex – mail.\* rules tells that write all log messege to /var/log/mail.log

Caching vs non caching:

\*caching is used if the path starts with hyphon sign.eg

**Mail.info -/var/log/mail.info**

**\***Caching mode don’t syc the messges with previous one.So you may loose message when system crashes.

\* using caching mode can improve system i/o performance.

Logger:

\*we can use logger cmd to generate syslog messges from your shell scripts.

Syntax: logger [options] message

Options:

-p FACILITY.SEVERITY

-T TAG

\*logger file generally writes all logiles to syslog.

This easy. Just type **logger <message>** on the command line and your message will be added to the end of the /var/log/syslog file.

$ logger comment to be added to log

$ tail -1 /vvar/log/syslog

May 21 18:02:16 butterfly shs: comment to be added to log

**Command output**

You can also add the output from commands by enclosing the commands in backticks.

$ logger `who`

$ tail -1 /var/log/syslog

May 21 18:02:43 butterfly shs: shs pts/0 2018-05-21 15:57 (192.168.0.15)

### Content from a file

The contents of text files can be added by using the **-f** option. Put the name of the file to be added to the log following the **-f** option as shown below.

$ cat msg

Backups to off-site facility will run this coming weekend.

System availability will not be affected.

$ logger -f msg

$ tail -2 /var/log/syslog

May 21 18:06:01 butterfly shs: Backups to off-site facility will run this coming weekend.

May 21 18:06:01 butterfly shs: System availability will not be affected.

### Using logger in scripts

You can add logger commands to scripts to make it easier to track the completion of important tasks.

$ grep logger /bin/runme

logger "$0 completed at `date`"

$ sudo runme

$ tail -1 /var/log/syslog

May 21 17:57:36 butterfly shs: ./runme completed at Mon May 21 17:57:36 EDT 2

**Disk partitioning :**

**\***Disk can be devided into parts called partition.Partitoins allows us to separate data.

\*partions schemes:

1- os ,applications ,user,swap,user home directory.

**MBR(Master boot record):**

\*It is bootsector at the beginning of the disk.It t contains a partion table which gives the info of logical partitioning of the disk.

\*MBR and partition table can address the disk upto 2TB,that’s why it is been replaced by GPT (GUID partion table)

\*MBR only allows you to partition upto 4 parts.if you want more then you have to use extended partition.

**GPT(Global uniq identifier partition table)**

\*Supports upto 128 partitions.Supports disk upto 128 zettabytes(2^70 bytes)

\*Not supported on old os and need some advance tool.

**Mountpoint:**

**\***A directory used to access the data on the partition./ slash is always an mounting point.

\* for eg if we mount /Jason to /home like this /home/Jason .Then /home/Jason is on the partition mounted on /home.

\*if we will change the mount point like /export/home/Jason ,then all data of /home/Jason will be available under /export dir.

Mounting over existing data:

\*we can mount the partition over dircectories which have already data present.

\*But old data will be only accessible after umounting the new partition.

For eg:

Mkdir /home/sarah is mounted on / dir. then again we have mounted the /dev/sdb2 on /home.

Like mount dev/sd2 /home.

Then we will not be able to see /home/sarah,we will have to unmount /dev/sd2 first then we will be able to see the /home/sarah.

**Fdisk:**

**\***Tool to create and modifiy the parttions.

Syntax:

Fdsik ( path to the device )

To see the current partion cmd: fdsik – l .It will show all disk partitions and to part it

Cmd : fdisk /dev/sda1 and give option.

\*To create new GPT table we will give “g” option and then “n” for new and finally for writing change “w” option.

**File system:**

**\*Ext** is the default file system for linux.ext2,ext3 and ext4 are latter releases.

Other linux file system are :

\*ReiserFS

\*JFS

\*XFS

\*ZFS

\*Btrfs

Mkfs:

Submitted by Sarath Pillai on Sat, 12/01/2012 - 10:19



Let me begin this tutorial by a famous UNIX statement which emphasizes the main underlying working model of Linux operating system. The statement is that "**Everything Is a File**".

If you are one of that person who spends a lot time with Linux or any NIX system, then you might have already understood that statement. Each and everything in Linux is accessed by a file. To make that much more clear, even block devices like Hard disks,and CD/DVD's are nothing other than a file(yeah but a special file..but a file).

Now if we start digging a bit deeper into files, we will come to know that, to the operating system(Linux) files inside a file system(be it ext2,ext3 or ext4) are not really accessed by their name. Names are helpful to humans but, the file system recognizes a file not by its name but by a number. That number through which the operating system reaches the location and other attributes of that file is called as an inode number. Now don't worry smiley, this tutorial is all about Inodes and its structure.

# 

# **What is an INODE in Linux?**

I must say that its a data structure that keeps track of all the information about a file.

*You store your information in a file, and the operating system stores the information about a file in an inode(sometimes called as an inode number).*

Information about files(data) are sometimes called metadata. So you can even say it in another way, "An inode is metadata of the data."

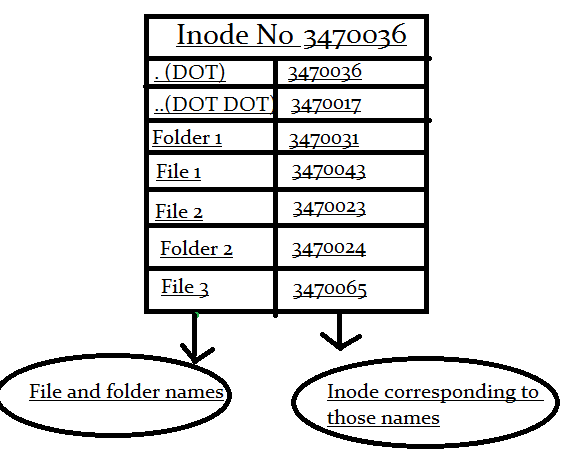
Whenever a user or a program needs access to a file, the operating system first searches for the exact and unique inode (inode number), in a table called as an inode table. In fact the program or the user who needs access to a file, reaches the file with the help of the inode number found from the inode table.

To reach a particular file with its "name" needs an inode number corresponding to that file. But to reach an inode number you dont require the file name. Infact with the inode number you can get the data.

# **How does the structure of an inode look like?**

This is the most important part to understand in terms of an inode. Here we will be discussing the contents of an inode.

## **Inode Structure of a Directory:**



Inode structure of a directory just consists of Name to Inode mapping of files and directories in that directory.

So here in the above shown diagram you can see the first two entries of (.) and (..) dot dot. You might have seen them whenever you list the contents of a directory.(most of the times they are hidden. You will have to use -a option with "ls" command to see them).

And people who are more into Linux or any NIX system, knows that the command "cd ." will change the directory to the current directory itself(which means it does nothing..because you are already in that directory.).

And the command "cd .." will take you to the previous directory or call it the parent directory of the current directory. Now why that happens?

Lets understand why this happens with an example. Imagine am in the directory /var/log on my system.

[root@slashroot1 log]# ls -ia

3633723 . 3633786 faillog 3634889 rpmpkgs.3

3633697 .. 3634727 gdm 3634893 rpmpkgs.4

3634833 acpid 3633883 httpd 3633813 samba

Now lets note down inode numbers of .(dot) and ..(dot dot).

.(dot) = 3633723

..(dot dot) = 3633697

Now lets do the directory listing of /var/ directory and see the inodes there.

[root@slashroot1 var]# ls -ia

3633697 . 3634275 cvs 3633698 lib 3633733 nis 3633737 spool

2 .. 3633724 db 3633729 local 3633734 opt 3633700 tmp

3633844 account 3633725 empty 3633730 lock 3633735 preserve 3634278 tux

3633701 cache 3633726 games 3633723 log 3633838 racoon 3633884 www

3634135 crash 3634624 gdm 3633732 mail 3633736 run 3633740 yp

Now lets note down the inode numbers of log directory and .(dot) from the directory listing of /var

.(dot) = 3633697

log = 3633723

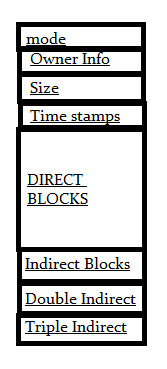
So you can clearly note that inode of .(dot) inside /var/log directory is equal to inode of log directory. And inode of ..(dot dot ) inside /var/log/ is equal to inode of .(dot) inside /var/ directory.

.(dot) always means the current directory just because its inode is same as the directory's inode. And ..(dot dot) means parent directory inode because its inode is same as the previous(parent) directory.

# 

# **Inode Structure of a File**

Now lets see how the structure of an inode of a file look like.



**Mode:**

This keeps information about two things, one is the permission information, the other is the type of inode, for example an inode can be of a file, directory or a block device etc.

**Owner Info:** Access details like owner of the file, group of the file etc.

**Size:**This location store the size of the file in terms of bytes.

**Time Stamps:**it stores the inode creation time, modification time, etc.

Now comes the important thing to understand about how a file is saved in a partition with the help of an inode.

**Block Size:** Whenever a partition is formatted with a file system.It normally gets formatted with a default block size. Now block size is the size of chunks in which data  
will be spread. So if the block size is 4K, then for a file of 15K it will take 4 blocks(because 4K\*4 16), and technically speaking you waste 1 K.