Linux Commands

FILE AND DIRECTORY COMMANDS

List all files in a long listing (detailed) format
Use case - Displays content of folder in details format providing file/dir permission, ownership information, timestamp, hidden files
ls -al

Display the present working directory
Use case - you are on command prompt and before doing any operation like rm or mkdir in that case to identify your current working directory pwd display present working directory
pwd

Create a directory
mkdir directory

Remove (delete) file permanently.
rm file

Copy file1 to file2
cp file1 file2

Rename or move file1 to file2. If file2 is an existing directory, move file1 into directory file2
mv file1 file2

Create symbolic link to linkname
Use case - You install two different version of same application on the system and you are expecting to use latest version of application to be active

ln -s /path/to/file linkname

Create an empty file or update the access and modification times of file.
touch file

View the contents of file

cat file

Browse through a text file
less file

Display the first 10 lines of file
Use case - Quickly look into the file what is inside without actually opening file
head file

Display the last 10 lines of file
tail file

Create tar named archive.tar containing directory.
tar cf archive.tar directory

Switch back to previous working directory
cd -

Go back to home directory
cd ~

List of all open files
lsof

Show free and used space on mounted filesystems
df -h

Display disk usage for all files and directories in human readable format
Use case - Identify which directory / file consuming maximum disk space during disk cleanup activity.
du -ah

PROCESS MANAGEMENT

Display all the currently running processes on the system.
ps -elf

Display and manage the top processes in real time
top

Kill process with process ID of pid
kill pid

Start program in the background
program &

Display stopped or background jobs
bg

Brings the most recent background job to foreground
fg

Running multiple command in single command
command1 && command2
command1; command2

Print process hierarchy
pstree

Display virtual memory statistics
vmstat 1

Display free and used memory (-h for human readable, -m for MB, -g for GB.)
free -h

Execute commands, showing periodic updates
watch “commands”

Show the current date and time
date

FIND / SEARCH / REPLACE

Search for pattern in file
grep pattern file

Search recursively for pattern in directory
grep -r pattern directory

Find files and directories by name
locate name

Find files in /home/john that start with "prefix".
find /home/john -name 'prefix*'

Find files larger than 100MB in /home
find /home -size +100M

Find string1 and replace with string2 from file
sed s/string1/string2/g file

USER INFORMATION AND MANAGEMENT

Use cases - a) Implement least privilege across your Unix and Linux server infrastructure

Display the user and group ids of your current user.
id

Display the last users who have logged onto the system.
last

Show who is logged into the system.
who

Show who is logged in and what they are doing.
w

Create a group named "test".
groupadd test

Create an account named john, with a comment of "John Smith" and create the user's home directory.
useradd -c "John Smith" -m john

Delete the john account.
userdel john

Add the john account to the sales group
usermod -aG sales john

NETWORKING

Display all network interfaces and ip address
ifconfig -a

Send ICMP echo request to host
Use case - If you want to check remote server reachability while troubleshooting
ping host

Display DNS information for domain
dig domainname

Download http://domain.com/file
wget http://domain.com/file

Display listening tcp and udp ports and corresponding programs
Use case - Shows on which port your application is listening / connected,
netstat -taupne

INSTALLING PACKAGES

Search for a package by keyword from package repository
yum search keyword

Install package.
yum install package

Display description and summary information about package.
yum info package

Install package from local file named package.rpm
rpm -i package.rpm

Remove/uninstall package.
yum remove package

AWK (TABULAR DATA MANIPULATION)

Awk Options

The awk command is used like this:
># **awk options program file**
Awk can take the following options:
-F fs To specify a file separator.
-f file To specify a file that contains awk script.
-v var=value To declare a variable.
We will see how to process files and print results using awk.

Using Variables

With awk, you can process text files. Awk assigns some variables for each data field found:

- $0 for the whole line.
- $1 for the first field.
- $2 for the second field.
- $n for the nth field.

The whitespace character like space or tab is the default separator between fields in awk.

Myfile >>
<col1 col2 col3 col4 col5>
DateTime CpuCount CpuWorkingTime CpuidleTime cpu_percent
2019-09-04 18:35:02 4 215.86 16171.22
2019-09-04 18:40:01 4 252.44 17325.68
2019-09-05 16:10:02 4 157.11 1014.07

>\# awk '{print $1}' myfile

Output =>
DateTime
2019-09-04
2019-09-04
2019-09-05

>\# awk '{print $3}' myfile - print 3rd column from file <myfile>

Output =>
CpuWorkingTime
4
4
4

Sometimes the separator in some files is not space nor tab but something else. You can specify it using –F option:
$ awk -F: '{print $1}' /etc/passwd

root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp

Using Multiple Commands

To run multiple commands, separate them with a semicolon like this:
$ echo "Hello Tom" | awk '{$2="Adam"; print $0}'

Output =>

Hello Adam

The first command makes the $2 field equals Adam. The second command prints the entire line.

Reading The Script From a File

You can type your awk script in a file and specify that file using the -f option.
Testfile >>
{print $1 " home at " $6}

$ awk -F: -f testfile /etc/passwd

root home at /root
bin home at /bin
daemon home at /sbin
adm home at /var/adm
lp home at /var/spool/lpd
sync home at /sbin
shutdown home at /sbin
halt home at /sbin
mail home at /var/spool/mail
operator home at /root
games home at /usr/games
ftp home at /var/ftp
nobody home at /
systemd-network home at /
dbus home at /

Here we print the username and his home path from /etc/passwd, and surely the separator is specified with capital -F which is the colon.

Awk Preprocessing

If you need to create a title or a header for your result or so. You can use the BEGIN keyword to achieve this. It runs before processing the data:
$ awk 'BEGIN {print "Report Title"}'
Let’s apply it to something we can see the result:

># awk 'BEGIN {print "The File Contents:"} {print $0}' myfile

Output =>
The File Contents:
DateTime CpuCount CpuWorkingTime CpuidleTime cpu_percent
2019-09-04 18:35:02 4 215.86 16171.22
2019-09-04 18:40:01 4 252.44 17325.68
2019-09-05 16:10:02 4 157.11 1014.07

Awk Postprocessing

To run a script after processing the data, use the END keyword:

awk 'BEGIN {print "The File Contents:"}

{print $0}

END {print "File footer"}' myfile

Output =>

The File Contents:
DateTime CpuCount CpuWorkingTime CpuidleTime cpu_percent
2019-09-04 18:35:02 4 215.86 16171.22
2019-09-04 18:40:01 4 252.44 17325.68
2019-09-05 16:10:02 4 157.11 1014.07

File footer

This is useful, you can use it to add a footer for example.
Let’s combine them together in a script file:
BEGIN {

print "Users and their corresponding home"

print " UserName \t HomePath"

print "_____ \t _____"

FS=":"

}

{

```
print $1 "  \t  " $6

}

END {

print "The end"

}
```

First, the top section is created using BEGIN keyword. Then we define the FS and print the footer at the end.

```
$ awk -f myscript  /etc/passwd
```

```
 Output =>
Users and thier corresponding home
 UserName       HomePath
_____________    ___________
root              /root
bin               /bin
daemon            /sbin
adm               /var/adm
```

Loops and Branches

1) for loops

Use cases -

a) Performing various operations on number of files from different folders
b) Number of empty files creation
c) Move files from one location to another location

for *arg* in *[list]*

This is the basic looping construct. It differs significantly from its *C* counterpart.

for *arg* in [*list*]

do

command(s)...

done

2) Nested for loop

for *arg* in [*list*]

do

for *arg* in [*list*]

do

command(s)...

done

command(s)...

done

3) Conditional Loop

Use case - Continuously running process - daemon process

while [*condition*]
do
command(s)...
done

4) **Testing and branching**

case (in) / esac

Use Case - User interactive script which prompts user to select option

The **case** construct permits branching to one of a number of code blocks, depending on condition tests. It serves as a kind of shorthand for multiple if/then/else statements and is an appropriate tool for creating menus.

case "$*variable*" in

"$*condition1*")

command...

;;

"$*condition2*")

command...

;;

esac

5) **Conditional if .. then .. else**

if [*condition*]; then

Command ;

else

Command;

fi

Bash Variables

Shown environment variables

env

Output value of $NAME variable

echo $NAME

Executable search path

$PATH

Home directory

$HOME

Current Shell

$SHELL

set environment variable

export PATH="/bin/bash"

BASH Special Characters

comments

; command seperator

$ variable substitution

${ } parameter substitution e.g ${parameter=default} if parameter not set , set it to default

` command substitution (back tick) The `**command**` construct makes available the output of **command** for assignment to a variable e.g MYNAME=`echo "Sunil" `

$# positional parameter (number of command line arguments)

$* all of the positional parameters

$$ process id of current process

[ ] , [[ ]] Test expression in conditional statements

> &> >& >> < <> redirection

scriptname >filename redirects the output of scriptname to file filename.

command &>filename redirects both the stdout and the stderr of command to filename.

script >&2 redirects stdout of command to stderr.

script >>filename appends the output of scriptname to file filename

| pipe. Passes the output (stdout) of a previous command to the input (stdin) of the next one

&& Logical AND

|| Logical OR

FILE Test Operators

Returns true if...

-e file exists

-f file is a *regular* file (not a directory or device file)

-s file is not zero size

-d file is a directory

-b file is a block device

-c file is a character device

-L file is a symbolic link

-S file is a socket