L I N U X C O M M A N D S

FILE AND DIRECTORY COMMANDS

ls -al

 # 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

lsof

 # List of all open files

pwd

 # 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

mkdir *<directory>*

 # Create a directory

rm *<file>*

 # Remove (delete) file permanently

cp *<file1> <file2>*

 # Copy file1 to file2

mv *<file1> <file2>*

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

ln -s *</path/to/file> <linkname>*

 # 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

touch *<file>*

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

cat *<file>*

 # View the contents of file

less *<file>*

 # Browse through a text file

head *<file>*

 # Display the first 10 lines of file

Use case -
Quickly look into the file what is inside without actually opening file

tail *<file>*

 # Display the last 10 lines of file

tar -cf *<archive.tar> <directory>*

 # Create tar named archive.tar containing directory

cd -

 # Switch back to previous working directory

cd ~

 # Go back to home directory

df -h

 # Show free and used space on mounted filesystems

du -ah

 # 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.

chmod *<perm> <file/dir>*

 # Change file / directory permissions

chown *<user:group> <file / dir>*

 # Change ownership of file or directory

PROCESS MANAGEMENT

ps -elf

 # Display all the currently running processes on the system

top

 # Display and manage the top processes in real time

kill pid

 # Kill process with process ID of pid

program &

 # Start program in the background

bg

 # Display stopped or background jobs

fg

 # Brings the most recent background job to foreground

command1 && command2
command1; command2

 # Running multiple command in single command

pstree

 # Print process hierarchy

free -h

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

watch *"<commands>"*

 # Execute commands, showing periodic updates

date

 # Show the current date and time

FIND / SEARCH / REPLACE

grep *<pattern> <file>*

 # Search for pattern in file

grep -r *<pattern> <directory>*

 # Search recursively for pattern in directory

find *</home/john>* **-name** *'prefix*'*

 # Find files in /home/john that start with "prefix"

find *</home>* **-size +100M**

 # Find files larger than 100MB in /home

sed -i 's/*<string1>***/***<string2>***/g'** *<file>*

 # Find string1 and replace with string2 from file

USER INFORMATION AND MANAGEMENT

id

 # Display the user and group ids of your current user

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

last

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

L I N U X C O M M A N D S

USER INFORMATION AND MANAGEMENT

who

 # Show who is logged into the system.

w

 # Show who is logged in and what they are doing

NETWORKING

ifconfig -a

 # Display all network interfaces and ip address

ping *<host>*

 # Send ICMP echo request to host

Use case-
If you want to check remote server reachability while troubleshooting

dig *<domainname>*

 # Display DNS information for domain

wget *<http://domain.com/file>*

 # Download http://domain.com/file

netstat -taupne

 # Display listening tcp and udp ports and corresponding programs

Use case-
Shows on which port your application is listening / connected

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.

```
# ps -elf
 <col1 col2 col3  col4   col5   col6 col7 col8 col9    col10  col11    col12    col13 col14    col15>
 F    S   UID PID  PPID C   PRI NI ADDR SZ WCHAN  STIME TTY TIME CMD
4     S   root  1    0    0   80  0    -    31 ep_pol   Jul22   ?    00:12:18 /usr/lib/systemd/systemd --switched-root --syst
1     S   root  2    0    0   80  0    -     0 kthrea   Jul22   ?    00:00:00 [kthreadd]
1     S   root  3    2    0   80  0    -     0 smpboo   Jul22   ?    00:00:18 [ksoftirqd/0]
1     S   root  5    2    0   60 -20   -     0 worker   Jul22   ?    00:00:00 [kworker/0:0H]
1     S   root  8    2    0   80  0    -     0 rcu_gp   Jul22   ?    00:00:00 [rcu_bh]
1     R   root  9    2    0   80  0    -     0 -        Jul22   ?    00:01:59 [rcu_sched]
```

># ps -elf | awk '{print $15}'
Prints 15th column data
Output =>

```
CMD
/usr/lib/systemd/systemd
[kthreadd]
[ksoftirqd/0]
[kworker/0:0H]
[migration/0]
[rcu_bh]
[rcu_sched]
```

># ps -elf | awk '{print $4}'
Prints PID column
Output =>

```
PID
1
2
```

3
5
7

Sometimes the separator in some files is not space nor tab but something else. You can specify it using -F option:
># cat /etc/passwd | 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.

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 "PROCESS LIST"}'
Let's apply it to something we can see the result:

># ps -elf | awk 'BEGIN{print " PROCESS LIST";print " ============"}{ print $15}'
Output =>
PROCESS LIST
============
CMD
/usr/lib/systemd/systemd
[kthreadd]
[ksoftirqd/0]
[kworker/0:0H]
[migration/0]
[rcu_bh]
[rcu_sched]

Awk Postprocessing

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

ps -elf | awk 'BEGIN
{print " PROCESS LIST";print " ============"}{ print $15}
END
{print " ### PROCESS LIST END ###" }'

Output =>
PROCESS LIST
============
CMD
/usr/lib/systemd/systemd
[kthreadd]
[ksoftirqd/0]
[kworker/0:0H]
[migration/0]
[rcu_bh]
[rcu_sched]

PROCESS LIST END

L I N U X C O M M A N D S

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) 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 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 E.g  if [ -e sample.txt ] then; echo "file
already exists" fi

> &> >& >> < <> redirection (STDIN - 0,  STDOUT - 1, STDERR - 2)
        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 e.g [ -e sample.txt ] && rm sample.txt
||     Logical OR
```

FILE Test Operators

Returns true if...

```
-e       file exists  e.g   if [ -e sample.txt ] then; echo "file already exists" fi
-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
```