Linux

In linux to switch to postgres use
sudo -i -u postgres
or
-login

To switch user in general
sudo su - harsh

If software already in repository then install it using
sudo apt-get install 'package-name'
To
sudo apt-get purge 'software-name'
or
sudo apt-get purge ~~--autoremove~~ --auto-remove 'name'
for uninstalling dependencies too

Linux Tutorials

1- Present working directory (pwd) → prints current directory

2- ls → list all files & directories of current directory
eg: ls, ls Documents/ → to list files & dir of Documents
ls ~ → home dir, ls / → root dir, ls .. → prev

ls -l → long format (detailed)
total 88
drwxr-xr-x 2 harsh harsh 4096 creation Time Desktop
Rights size

drwx → directory read write execute

d | rwx rwx rwx
- omitted in r-x means write is missing so it cannot be written by others (public)

user rights | group rights | other rights

for viewing hidden files
ls -a

sorting by size
ls -lS

Search using wildcard (*)

ls *.html → list files with .html extension

ls *.* → list all files containing '.'

listing directories only
ls -d */

* → for other all
ls * → when '*' is used it will list all files & files of the directories too with / symbol

Videos → abc.mp4
imp → doc1, doc2
Music → chester, weekend → live → morning.be

In Videos directory: ls *
abc.mp4
imp:
Music:
chester weekend live

ls */ → all files with / meaning files of directory only

imp/
doc1 doc2

Music:
chester weekend live

ls -d : output → . → represents current directory

~~so combining with~~ -d with */ means
ls prints files under current directory so -d flag will print the directories of listed files which is the current file.

Combining ~~it with~~ -d with */ listing directories of files under the directories which are under current directory

Videos
imp/ Music
doc1 doc2

ls -d * & ls produce same output

To get information about ls command
man ls
manual

print sizes with human readable form (KB, MB) with -l
ls -l -h

-r, --reverse : reverse order while sorting

-R --recursive: ls -R
list subdirectories recursively (nesting level: indicated)

-t → sort by modification time

Recursively search files:
We can't just use ls -R *.pdf together because first '*.pdf' is parsed then -R so no advantage

There are 2 ways
1- Enable globstar (**): shopt -s globstar
ls **/*.pdf
Disadvantage: Not work for very large no. of files

2- find . -name "*.pdf"
By default it has attribute -print so it will just print it.

To print in the file
find . -name "*.pdf" -fprint file.txt

cd (Change directory): Fix space between two words
1- cd My\ books or cd "My books" or cd 'My books'

2 cat (concatenate): Related to text files
Syntax: cat options file1 file2

Eg cat ↵
Hello ↵
Hello ← echo input
Display contents of text file.
cat list1.txt ↵
list 1 line 1
list 1 line 2

2- cat list1.txt list2.txt ← display contents of both files

3- cat -b list1.txt : Add line no. to non-empty lines
4- cat -n list1.txt : Add line no. to all lines (including empty lines)

5- cat -s list1.txt: Squeezes multiple blank lines to one line

eg: line 1
line 2
→
line 1
line 2
line 3

line 3

6- cat -E list1.txt: Add '$' to end of line
→ list 1 line 1$
list 2 line 2$
$
list 3 line 3$

I/O Redirection

output > file

cat > test.txt ↵
line 1
line 2
line 3 ↵

cat test.txt
"
"
"

cat < test.txt
"
"
"

'>' truncates it not append it. So to append use '>>'

Concatenating contents of 2 files to one file:
cat list1.txt list2.txt > out.txt

mkdir (make directory): mkdir foldername
It can create a directory at single level only
mkdir foldername/subdirname
→ error: foldernames do not exist
So we need to create directory structure
mkdir -p image/camera
or
" --parent " "

for multiple dir under one

mkdir -p name/~~abcd~~{abc1, abc2, abc3}
Also
mkdir -p name/{ab{1,2}, c/{3,4}}

name:
ab:
1 2
c:
3 4

rm & rmdir
mkdir -p a/b/c/d/e
rmdir a/b/c/d/e → remove only a folder
To delete whole directory structure use -p
rmdir -p a/b/c/d/e

-v, --verbose: prints message for the action

If at dir b, there is a file then using rmdir -p a/b/c/d/e will remove c, d & e but not a & b as 'b' directory contains a file another than c.

So we can use rm -r (recursive) a
This will delete whole a dir along with its subdir & files.

cp (copy) → cp options source destination

Copying a file to specific directory
cp file.txt dir1
ls dir1
→ file.txt

Copying multiple files

cp file.txt file2.txt dir2
↓
sources

Interactive copying (Ask before replacing same name file)

cp -i file.txt dir1
cp: overwrite 'dir1/file1.txt'? y
or n

Copying files to curr dir:

cp ../file.txt . ← curr dir symbol

For copying directory (non-empty)
cp -R dir1 dir3
↓
recursive

If the dir3 doesn't exist then files of dir1 is transferred to the newly created directory 'dir3'.
Content of dir1
abc1.txt
abc2.txt

Content of dir3
abc1.txt
abc2.txt

If dir3 already exist
Content of dir1
"
"

Content of dir3
dir1 → " "
"
"

mv (move): mv options source dest

Renaming
mv file.txt file3.txt
↑ new name

Same renaming can be with directory

But it will happen if dir doesn't exist.
If dir exist then
mv dir1 dir3.
dir1 is copied under dir3
Content of dir3
→ dir1

less: shows contents of the file

For searching keyword up to down type '/book'
From down to up type '?book'

touch: to create empty files
To modify timestamp of file
If file already exist, then file timestamp of file will be updated.

nano: simplest cli text editors in linux

top: To get usage of CPU, memory, etc with default refresh rate of 3s which too can be changed with s key.

Using I key, no idle process is being shown, just only running processes

To kill a process, press K & enter PID for the process.

kill: kill pid
A stronger version
kill -KILL 3294
forcefully close process

Another stronger but softer option
kill -9 3294

To get the list of all running process.

ps -ux
for all the processes
ps -aux
for getting list of particular user
ps -U harsh

To get list of all instances
ps -C brave

echo: print the value
x=10
echo "the value of x is $x"

for using escape sequence characters

echo -e 'some text \t is written'
enable escape sequence
→ some text is written

chmod: Change permission of file.
eg: chmod o+wx file
others adding write & execute

Adding permission for others to write & execute. To remove these permission
chmod o-wx file
remove

For group use g instead of o
For user use u instead of o.
replacing
eg: chmod ug=rw
with user & group

Suppose if user already has permission for write & execute (wx) then wx is replaced by rw (read & write). Due to '=' now user can't execute which was possible before this command
chmod ugo=r or chmod a-rwx file
all

eg: chmod u+rw,g=rw,o+r file

chmod can be used to change permission for directories
Write permission - Eg - Creating file inside dir
Read: Eg: Using ls for dir
Execute: Eg: Using cd for dir

Octal & Numerical permission: Instead of using u+wx, o-w, we can simply use octal representation

Users	Group	others
r wx	r-x	r--
1 1 1	1 0 1	1 0 0
4 2 1	4 0 1	4 0 0
=7	=5	=4

chmod u=rwx, g=r-x, o=r
or
chmod 754

Bash Scripting

nano myscript.sh

#! location of bash
To get location of bash type 'which bash' on terminal to list it
eg: #!/usr/bin/bash
ls -l
Ctrl + O (To save)
Ctrl + X (Exit)

By default script doesn't have execute permission so do
chmod +x myscript.sh

To execute script : ./myscript.sh

which and whatis command:
which command is used to get location of the executable
Eg: which ls
→ /usr/bin/java

whatis: to get short description
eg: whatis ls
ls(1) - List directory contents

useradd: To create user in linux
sudo useradd new_user -m -s /usr/bin/bash
username default home dir (/home) dir for shell

-g users -c 'comment'
group name comment

Change password of user:
sudo passwd test
New password:
Retype new password:
passwd: password updated successfully

Deleting user: sudo userdel test
This won't delete data from /home nor its folder
Use sudo userdel -r test.
or after sudo userdel test
use sudo rm -r /home/test/

Basic Group Management

Listing the groups which are connected to current user
$ groups
harsh adm cdrom sudo dip plugdev lpadmin sambashare

Listing all the groups
cat /etc/group

Adding groups: groupadd
sudo groupadd cpp

Deleting group
sudo groupdel cpp

Adding user to group
sudo gpasswd -a harsh cpp
adding user name group name

Removing user from group
sudo gpasswd -d harsh cpp
deleting

.bashrc file

This file contains script which is loaded each time a terminal opens
for eg: eg in bashrc file
echo "This is test"
alias ls='ls --color=auto -l'

Now when terminal opens
This is test
harsh $ ls
This ls behaves as ls --color=auto -l

Viewing Resources

1- df: Amount of disk space used by file system
eg: list partions & its spaces, mount, etc
Flag: -h is recommend for human readable sizes

2- du: list the disk space usage of every file under the directory and at last total space used by directory
desktop $ du -h
Flag: du -sh
-s: summary. Just gives total space used by dir

3- free: Displays total amount of free & used physical & swap memory in system and also buffer used by kernel.
eg: free -m → in megabytes, -g → GB

	total	used	free	shared	buff/cache	available
Mem	5919	2086	15733	226	2260	3316
Swap	1906	0	1906			

watch command

To run command/script at regular intervals
eg: watch free -m

Modify - frequency:
watch -n 1 free -m
in seconds

Head and Tail Command

Head: first 10 lines of file is printed
Tail: Last 10 lines of file is printed
eg: head var.log
tail var.log

Flags: changing no. of lines
eg: Printing first 3 lines
head -n3 dmesg
tail -n3 dmesg ← last 3 lines

tail -f var.log
→ follow
Any changes at end of file ie last 10 lines will be detected.

Multiple files. head dmesg kern.log
dmesg
- First 10 lines

kern.log
first 10 lines

find

find directory -name regular expression

eg: find /home/harsh -name *.txt
→ list all files with .txt (including files inside nested dir recursively)
Flags: find - -mtime n
n or -n or +n

n → exactly n days ago
-n → less than n days ago
+n → more than n days ago

wc (word count)

eg: wc test.txt
1 6 42 test.txt → name of files
↓ no of lines ↓ no of words ↓ no of characters

Flag: -c: To print only no. of characters
-l: To print " " no. of lines
-w: " " words
-L: Print no. of characters in longest line

cal

To display conventionally formatted calendar

```
Jan 2021
So Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
```

21 → current date

ncal: To display as transpose of calendar

```
Jan 2021
So   1   8
Mo   2   9
Tu   3   .
We   4   .
Th   5   .
Fr   6   .
Sa   7   .
```

cal 2016: display full calendar of 2016 including all months

For specific month of year

cal 2 2016

Feb

default (current month)

Flag: -1 or -3

-3: Print prev, current & next month calendar

date

- Display the current time

→ Saturday 21 August 2021 02:42:56 PM IST

format specifier:

- +%m: Specify month number
- +%h: abbreviate month (Jan-Dec)
- +%B: full month name
- +%d: day of month
- +%y: year
- +%H, +%M and +%S → hour, min & sec
- +%D: Date as mm/dd/yy
- +%T: Time as hh:mm:ss
- +%n: New line character

eg: date "+%d-%h-%y"

Combine multiple commands

By using semicolons

ls ; pwd

By using &&

ls && pwd

Difference b/w ';' && and '&&':

In ls command fails then rest of the commands wont execute in case of '&&'.

But in ';' it doesn't happen & remaing commands will execute

'||': ls || pwd

Here if the command is successful then next command won't execute

apt-get

It is APT package handling utility used to install packages & softwares available in repositories.
APT stands for Advanced Packaging Tool

sudo apt-get update: Resynchronize local package file
Location of local package file: /etc/apt/sources.list

or in /etc/apt/sources.list.d/
There are various files containing the repositories with same file name.

sudo apt-get install php

Uninstall
sudo apt-get --purge remove packagename
delete all configuration files
↓
uninstall

or
" " --purge autoremove packagename
↓
uninstall packagename
& its dependencies too

Sometimes autoremove dont delete dependencies. So make sure to have list of dependencies saved before installing program then
sudo apt-get purge php7.4 php-common - - -

Also if dependencies name not available then they can be found (not all but most) do using

dpkg -l | grep package name | awk '{print $2}' | tr "\n" " ".

ifconfig (interface configuration)
Used to view & change network interface configuration of system

eth0: ethernet
lo: loopback interface - special interface that system uses to communicate with itself

In eth0
HWaddr or ether or ether addr
Mac address is hard coded address assigned to network interface card (NIC). So ethernet, bluetooth, wifi have their own Mac address.

inet addr: IPv4
inet6 addr: IPv6

Enable/disable network interface

sudo ifconfig wlan0 down → disable
up → enable

tar: used to create, maintain, modify & extract files which are archived in tar format.
It stands for tape archive.

eg: tar -cvf → verbose
c → creating archive
f → to allow us to specify filename

If -f isn't used then system dependent filename is created.

Compressing
tar -cvf test.tar test → foldername (It can include too)
↓ zipfile name

tar.gz, gz stands for gzip
↓ for gzip
tar -czvf test.tar.gz test
z must come after just after c flag

Using z actually reduce the size too.
↓ compresses

Extracting: tar -xzvf test.tar.gz
↑ extracting

grep
Stands for global regular expression. It processes text line by line & print any line which matches our specified pattern

grep "options" file.txt
→ Got all line containing the substring "options"
" 123options "
" we options are "

Flags: -i: Case insensitive search as case sensitive search is by default

-n: To print line number along with result
eg: grep -in "options" test.txt
→ 9: [options]
14: options is test

grep -n "Some options" file1.txt file2.txt
↓ multiple files search

Also
" " " "some options" * → all files in current folder

-v: To invert match - All lines which do not contain pattern
grep -v "options" file.txt

-w: whole word search
eg: grep "options" filename
123options → X
this options is → ✓
[options] ✓

To see 4 lines before the match
eg: grep -win -B 4 "options" file
47 " "
48 " "
49 " "
50 " "
51 this is options

Note: to get a list of commands history use command "history".

To see after the 4 lines use -A instead of -B
To see both
eg: grep -C 2 "....." file
→ 2 lines before then asked line then 2 lines after it.

For recursive search: -r
grep -winr "....."

-l: returns list of files containing the matches

grep -wirl "...".

~~history~~

Pipes

A pipe is a form of redirection to send output of one command/program/process to another command
Symbol: '|'
Output of one command can be used as input of another command.
It is unidirectional (left to right)

$ history | grep "ls"
→ list all the commands used which has ls in it.

To narrow down the result -
If we want to search command containing both ls and grep
$ history | grep -w "ls" | grep -w "grep"

grep uses POSIX regular expression

1- Phone no: "...-...-...."
. → character
But for digits only
"\d{3}-\d{3}-\d{4}" → Perl regex
\d{n} → n digits
grep -P "\d{3}" ... "
for perl regex

General Regex

Metacharacters (Needs to be escaped):

. [{ ( ) \ ^ $ | ? * +

These have special meaning so if these characters are to be really included in regex, not it's meaning then use '\'
To search for dots: "\."

In case of grep '\' is used in both regex & grep so to search for '\'

grep "\\\\"

\\ → \ as grep has special meaning
so \\ is required which will be converted to search for '\'

\\\\ → grep → \\ → regex → \

\D → Non-digits (alphabets & special characters)

\w → match all characters except special characters
\W → special characters
\s → whitespace
\S → not whitespace
\b → word boundary (\B → not a word boundary)

\bfoo (abcfoo) — boundary & word

\babc
first 2 are \b?b?c
1st 1 is \babc

In word boundary case, either side of boundary is a word character & other side a non-word (special character)

^ - Beginning of string
$ - End of string

Ha Ha Ha

^Ha → (Ha) Ha Ha
Ha$ → Ha Ha(Ha)

123-456-789
123.456-789
123*456*789

\d{3}[-.]\d\d\d[-.]\d{3}
→ First 2 lines

[]: Valid list of characters inside it
[-.] → either '-' or '.'

Characters in regex
[1-7] → characters from 1-7: 1, 2, 3, 4, 5, 6, 7
[a-z]: lowercase letter
[A-Z]: uppercase ''
[a-zA-Z]: both uppercase & lowercase are valid

[^]: Matches characters not in brackets
|: Either or
(): Group

Quantifiers:

* : 0 or More
+ : 1 or more
? : 0 or 1
{n} : Exact number
{min,max} : Range of numbers

Eg: Mr. Sha | Ms Davis
Mr Laa | Mrs. Robinson
Mr. T

Mr\.?\s[A-Z]\w*

- \.? : one or zero dot
- [A-Z] : Capital letter
- \w* : 0 or more word characters

For other right side, we need groups

M(r|s|rs)\.?\s[A-Z]\w*

In [ ], there are no group like 'rs' possible as only single characters are possible

netstat (Network Statistics)

Displays network connections, routing tables and no. of network interfaces

Recommended way to use netstat
eg: netstat -a | less
(default has output is large)

For TCP only connection use -t
netstat -at
For UDP connections use -u
use -l to get output of listening state

-s: for statistic
→ Ip:
168 total packets received
0 forwarded
165 incoming packets delivered

Icmp:

Icmpmg:

Tcp:
60 active connections openings

Udp:

Get pid of related interfaces
eg: netstat -pt | less

To get numeric ports of interfaces use
netstat -n | less

-c: Used to show states, stats continously (keep refreshing)

-ie: extended interface
(interface → extended)

netstat -ie is same as ifconfig

scp
Used to copy files, directories from local machine to remote server or vice-versa

eg: scp hello.sh codebird@192.168.245.141:/home/codebird

Password of host: ....

→ hello.sh
↑ Transferred

For transfering dir along with its files, use flag -r

Shell Scripting

sh: First shell used was this for Unix & unix like system.
bash (bourne again shell): Improved version of sh which is intutive & flexible. Standard nowadays
(having the ability to know or understand without any proof)

Comment: # this is comment

Variables: Recommended way for defining variable is in lowercase
a variable BASH which stores the directory of bash ie /bin/bash or /usr/bin/bash
More pre-defined variables: → system variables
BASH_VERSION, HOME, PWD
To use variable's value put '$' sign before them

User defined variable: name=Mark
echo "Name is $name" → if instead of double quotes, single quote is used then $name is printed as $name not value of name

Taking Input

echo "Names:"
read name1 name2
echo "Names are $name1 $name2"

→ Names:
Harsh Harsh1
Names are Harsh Harsh1.

To take input in same line as that of msg.

read -p "username: " username
echo username is $username

For entering password (password typing not visible)
-s
read -sp 'password: ' passvar
echo

When using -s once the password is typed & enter is hit the output or next prompt will be from next line & is in same line so using echo at last, it will be in next line

Reading inputs in array

```
echo "Enter names : "
read -a names
echo "Names: ${names[0]}, ${names[1]}"
```

Inbuilt variable → REPLY
When reading if no variable is specified then by default value is stored in REPLY variable

```
echo 'Enter name:'
read
echo "Name: $REPLY"
```

Passing arguments to Bash-Script

By default arguments are passed in variables 0, 1, 2, 3, 4 & so-on
to use them e.g.
$1, $2, $3...

./hello.sh Mark Tom John ↵

```
hello.sh
echo $0 $1 $2 $3
```

→ ./hello.sh Mark Tom John

Another way: args=("$@") ← no spaces between them
args=("$@")

```
# $@ : stores argument as an array
echo ${args[0]} ${args[1]} ${args[2]}
```

→ Mark Tom John
args[0] args[1] args[2]

In case of inbuilt variables numbers for passed argument starts from 1 not 0.
So 0 is for (first argument (non-passed) ie script name

To print all arguments passed use
echo $@
→ Mark Tom John

To print no. of arguments passed use
echo $#
→ 3

Conditional Statements

Syntax:

```
if [condition]
then
    statement
fi  end of if statement
```

eg: count=10

```
if [ $count -eq 9 ]
then
    echo 'cond is true'
fi
```

Relational Operator Usage Guide

Integer comparision

-eq : is equal to. Eg: if [$a -eq $b] (space before and after)
-ne : not equal to
-gt : greater than
-ge : greater than or equal to
-lt : less than
-le : less than or equal to

Symbols:

< : less than. eg: ((a < b))
<= : less than or equal to
> : greater than
>= greater than or equal to

String comparision

= equal to. eg: if [$a = $b] (space before and after)
== : equal to
!= : not equal to
< : less than. eg: if [[$a < $b]]
> : greater than
-z : string is null, ie zero length

In case of integer comparision if angular brackets (<,>) are used then double parenthesis are used → ((...))
In string comparision double square brackets : [[...]]

else: if [condition and]
& elif then statement

elif [condition]
then
    statement
else
    statement
fi ← end of if/elif/else

File test Operators

echo -e "Enter name of file : \c":
-e to enable escape sequence characters
\c : Puts the cursor in same line. Meaning after pressing enter - Next prompt/output will be in not be in new line and will be in same line

echo -e " Enter ... "
read file_name

if [-e $file_name] ← exist or not
then
    echo "$file_name found"
else
    echo "$file_name not found"

if [-d $filename] ← exists or not → if yes checks if file is dir or not

-f :- like -d it checks for regular files (not folder files)

Also there are two types of files on basis of encoding

1- Block special files :- binary files like media → byte 64.
2- Character special files :- text files → ASCII specially

-b : block special
-c :- character special

-s : Checks if file is empty or not

-r :- Check if file has read permission
-w :- " " write "
-x :- " " execute "