Harshpreet Singh
2210990393 ST-2.
9056299166 → Helpline :)

classmate
Date
Page

Editors

Gedit → simple yet powerful text editor (like NOTEPAD) that comes pre-installed with the GNOME desktop environment.

Features : • syntax highlighting • plugins • customizable interface
• tabs & split windows • search & replace • auto indentation

Open the file with $ gedit file1.txt this will open a notepad like window where you can edit the file. Then click on save & close it.

Vi → a very powerful text editor that has been a standard on Unix & Linux Systems. It is a modal text editor, i.e it has different modes for editing text, navigating and executing commands.

$ vi file.txt → if not available, then it creates a new file & opens it for editing.
To save this empty file use → :wq

3 types of modes in File editing

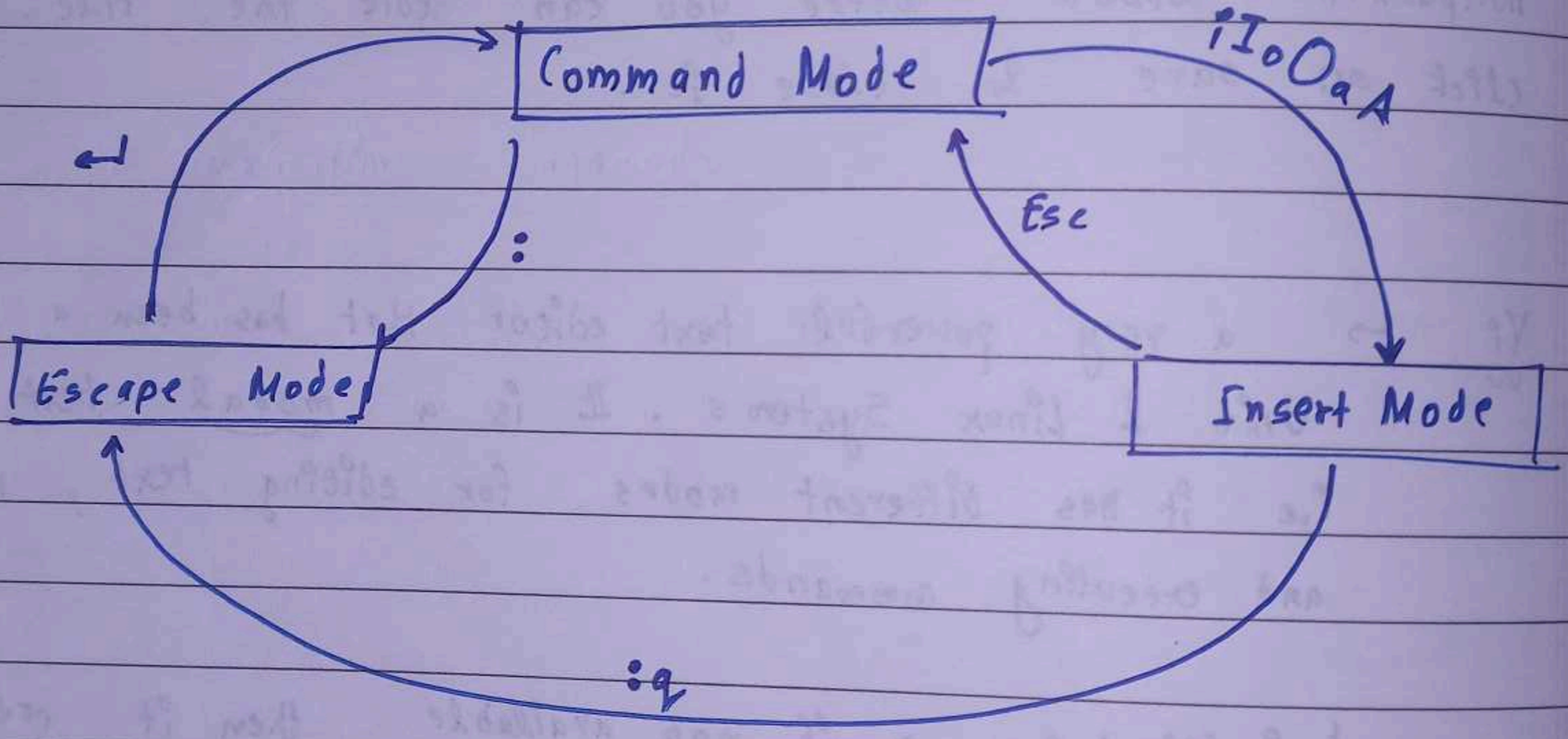
Command Mode : Default mode. Here we can use any vi command. From command mode, we can enter into insert mode through characters that convey a special meaning. (Discussed later)

Insert / Input Mode :

In this mode we can modify file data. We can insert/ append new data. From insert mode, we enter into command mode by using <esc> key.

Exit :

To quit from the editor. From the command mode, we have to press : the we can enter into exit mode.



These are used to take you from Command to Insert mode.

Insert Mode :

How to Insert and Append Data :

A → To append data at the end of file.
I → To insert data at the beginning of the line.
a → To append data to the right side of cursor position (Just after)
i → To insert data to the left side of cursor position (Just before)
O → To open a line above cursor position
o → To open a line below cursor position.

/cc

S → Replaces single character under the cursor with any number of characters and switches to insert mode.

R → Overwrites text from the cursor to the right, without switching to insert mode.

r → replace a current character.

Commands within Insert mode

Commands for Deleting

Character-wise

x → To delete a single character (del key)

X → To delete previous character (backspace key)

nx → To delete ~~prev~~ n characters. (Eg → 5x or 10x)

nX → To delete last n previous characters. (Eg → 5X or 20X)

Word-wise

dw → To delete current word.

ndw → To delete n words.

Line-wise

dd /Dd → To delete current ~~word~~ line.

ndd → To delete n lines

d$ → Deletes from current position to end of ~~line~~ line.

d^ → Deletes from current position to beginning of line.

dgg → Deletes from beginning of the file to current cursor position

Dg → Deletes from current position to end of file.

Copy and Paste Data

yy → To copy a Line (yanking)
Nyy → To copy N lines
yw → To copy a word
Nyw → To copy N words
y$ → To copy from current cursor position to end of line
y^ → To copy from the beginning of the line to current cursor position
p → Paste above the cursor position
P → Paste below the cursor position.

Cursor Navigation Commands

k → move cursor up one line ('↑')
j → cursor down one line ('↓')
h → left one-character ('←')
l → right one-character ('→')
3k → 3 lines up
3j → 3 lines down
3l → 3 characters right
3h → 3 characters left
'$' → End of current line
'^' / '0' → Beginning of current line
H → Beginning of current page
M → Middle of current page
L → End of current page
G → Move to last line (i.e End of file)
u → Undo last operation
w → next word B → previous word
/... → search word after /

:n ⟵ any number ⟶ cursor on that line.

Commands for scrolling.

ctrl + d → ~~spcr~~ screen down by half a page
ctrl + f → screen down by full page
ctrl + u → screen up by half page
ctrl + b → screen up by full page
ctrl + e → screen up by 1 line
ctrl + y → screen ~~up by~~ down by 1 line
ctrl + I → redraw the screen

Exit Mode

:n → To go to nth line.
:w → Save file data
:wq /22 → Save & quit from editor
:q → quit
:q! → force quit
:set nu → To set line numbers in editor
:$set nonu → remove line numbers
:n → Place cursor to nth line
:$ → place cursor to last line
:! <command> → To execute any command
:r [file_name] → read data from a file.
:!cmd → to run shell commands

Search & Replace

/search_term → To search_term in a file. & 'n' to go to next occurance.
:%s/old_text/new_text/g → to globally replace strings/texts.

Difference b/w Vi & Vim

↑ Vi Improved

	Vi	Vim
Description	Original Unix text editor.	Enhanced version built upon Vi
Modes	Normal, insert, command-line.	Normal, insert, visual, command-line etc.
Features	Basic Editing	Extends Vi with additional features.
Customization	Limited customization.	Extensive customization wrt plugins, themes, mappings, etc.
Syntax Highlighting.	Not there.	Supports.
Plugins	Not there.	Supports.
Scripting	x	✓

Nano → a user-friendly, simple and WYSIWYG (What You See Is What You Get) text editor which is an improved version of pico text editor. It has gui. but is a command line editor.

ctrl + g	(F1)	Display this help text.
ctrl + x	(F2)	Close current file buffer / Exit from Nano.
ctrl + o	(F3)	Write current file to disk.
ctrl + r	(F5)	Insert another file into current one.
ctrl + w	(F6)	Search forward for a string or regex.
ctrl + \	(M-R)	Replace a string or regex.

ctrl + k (F9) Cut the current line and store it in cutbuffer.
ctrl + u (F10) Uncut from cutbuffer into current line.

But main important options :

ctrl + O → to save content
ctrl + X → To quit from the editor

Ex.

ctrl + alt + → for syntax highlighting

Vim (Vi improved)

highly configurable & powerful text editor that is evolved from original Vi editor. It retains the modal editing concept just adds upon it.

Modes :

Normal mode → default mode for navigating & executing commands.
Insert mode → Allows insertion & editing of text.
Visual mode → Used for selecting & manipulating blocks of text visually
Command-Line Mode → Used for entering commands like saving changes, searching, and replacing text.

Eg → vim filename.

Pico → simple text editor that comes as part of the Pine email client but is also available as a standalone editor.

To open a file using pico → pico filename

Ctrl + G → Help
Ctrl + X → Exit
Ctrl + O → Save file
ctrl + R → Read in a file into current buffer
ctrl + W → search for a specific text string.
ctrl + K → cut the current line
ctrl + U → paste the cut text
ctrl + C → show cursor position
ctrl + _ → move to a specific line number

Options

-t → Enable tab emulation using spaces.
-w → Disable word wrap
-r → Restore the last editing session for the specified file

Customizing pico → ~/.pico {tabsize, justify, enable_meta-key}

File → command to determine the type of the file.

Syntax : file [file name]

Sed (Stream EDitor) → text processing command. Does operations like search, replace, delete, insert, etc.

Syntax → sed OPTIONS 'command' filename

Options.

-i → in place (makes actual changes in file instead of printing it)

-e → for multiple expressions

-n → suppresses output

-r → allows regex in expression.

Symbols

$ → represents the end of line.

^ → represents start of line

p → prints specific pattern lines

d → delete " " "

a → Append at

< > → start & end of a word

g → for every occurrence

/ → parts in expression

\ → escape sequence. Represent some regex as < >.

s → means substitute

+ → adding some lines after matching the pattern

; → seperate multiple expression.

Displaying some lines

sed "np" file → Display n^{th} line twice.
sed -n "3p" file → only 3rd line
sed -n '$p' file → last line
sed -n '2,4p' file → 2nd to 4th lines
sed -n '2!p' file → all except 2nd line
sed -n '2,4!p' file → display except 2nd to 4th lies
sed -n '/pattern/p' file → display lines containing specific pattern.
sed -n '2,+2p' file → prints 2nd & +2 lines.
sed -n '/pattern/', +2p' → after matching pattern print +2 lines.

Delete some lines.

sed '3d' file → Delete 3rd line but will not delete in file
sed -i '5d' file → Permanently deletes 5th line.
sed -i '1,$d' file → Deletes all lines.
sed '/pattern/d' → all lins where pattern occurs.
sed '/pattern/', +2p' → after matching pattern print +2 lins.

Replace some woods

sed 's/old/new/' file → change old to new word for first occurrence in every line.
sed 's/old/new/i' file → ignores case
sed 's/old/new/n' file → for n^{th} occurance in every line.
sed 's/old/new/g' file → for every occurance.
sed '/r-x/s/Harshpreet/Harsh' file → replace Harshpreet with Harsh Add lines where r-x substring exists.
sed -i 's/^$/this is a blank line/' file → replace every blank like.

Inserting

sed '

sed 'n

sed '

sed '

Cut

Syntax

-c ,

specif

charac

-f

-d

-- c

Inserting Some lines

sed '/pattern/a line' → insert line after the pattern.
sed 'na line' → insert line after nth line.
sed '/pattern/i line' → insert line before that pattern.
sed 'ni line' → insert line before nth line.

Cut → used to extract sections from each line of input by specifying columns or fields. Useful for parsing text or extracting specific parts of a file.

Syntax → cut OPTION... [FILE]

-c, --characters = LIST → to extract only characters specified in the list. This list can be a comma-seperated list of character positions or ranges (e.g., 1-5, 9-12).

cut -c 1,2,4-7 file.txt

-f → extract only fields. "Columns"

cut -f 1,3 -d ',' file.csv
cut -f 1,3 -d ':' file.txt

-d → delimiter

cut -d ':' -f 1,3 file.txt

--complement → to output complement.

Paste → used to merge files.

Syntax → paste [OPTION].. [FILE]...

-d → delimiter. → paste -d ',' file1.txt file2.txt.

-s → past on file at a time instead of merging.

paste -s file1.txt file2.txt.

tr. (translate) used to translate (transform or substitute) or delete characters in a text stream.

cat >demo.txt | tr 'aeiou' 'AEIOU' < demo.txt → replace lower case vowels with upper case vowels.

[0-9] → for digits.

cat > demo.txt | tr '[a-z]' '[A-Z]' < demo.txt → lower to upper case.

cat > demo.txt | tr '[a-zA-Z]' '[A-Za-z]' < demo.txt → replaces lower with upper case & upper case with lower.

-d → Deletes a character

-s → Squeezes repeated occurances of character

Awk. → a scripting language used for manipulating data and generating reports. This requires ~~program~~ no compiling and allows the use of variables, numeric functions, string function, and logical operators.

Abbreviated as → Aho, Weinberger and Kernighan.

What

1. AWK Op

2. Useful fo

3. Programmi

Syntax

awk [optio

awk [optio

Begin Blo

Action Bloc

End Block

Date
Page

What can we do with awk ?

1. **Awk Operations**
   - scans a file line by line.
   - splits each input line into fields.
   - compares input line / fields to pattern
   - performs actions) on matched lines

2. **Useful for**
   - transform data files
   - produce formatted reports

3. **Programming Constructs**
   - format output lines
   - arithmetic & string operations
   - conditionals & loops.

Syntax

awk [option] { Begin Block } { Action Block } { End Block }

awk [options] 'selection_criteria {action }' input_file > output_file.

Begin Block ⟶ only executed before processing of the file

Action Block → executed for every line / record present in file

End Block → executed only after completing the processing of the file.

Options. F → input field separator
f → takes blocks from a file

Predefined Variables of awk :

FS → Field Separator. Default FS is ' ' (space).
NR → Row number / number of rows
NF → No. of fields
RS → Record Separator
$N → Represents n^{th} column
OFS → stores Output Field Separator, which seperates the fields when Awk prints them. The default is blank space. (Taylor's version). Whenever print has several parameters seperated with commas, it will print the value of OFS in b/w each parameter.

ORS → stores Output Record Separator, which separates the output lines when Awk prints them. The default is a newline character. Prints automatically outputs the contents of ORS at the end of whatever it is given to print.

Examples

awk '{print $1}' Afilename → To print column 1
awk '{print $1 " -> " $2}' file → To print col 1 -> col 2
awk '{print $0}' file → To print all columns.
awk -F '|' '{print $1 " -> " $2}' file → changes field seperator to '|'. Same
awk '{print $1 " -> " $2}' FS = '|' file

awk 'NR != 1 {print $0}' file → All rows except first one
awk 'NR == 1 {print $0}' file → Only 1st row
awk 'NR == 2 || NR == 4 {print $0}' file → 2nd & 4th row
awk 'NR > 2 {print $0}' file → All rows except 1 & 2.
awk 'NR >= 2 && NR <= 4 {print $0}' file → 2,3,4.
awk 'NR == 2, NR == 4 {print $0}' file → 2,3,4
awk '$3 > 20 {print $0}' file → Rows in col. 3 with values > 20
awk '{print}' file → print all file
awk '/s/ {print $0}' file → Search pattern s and print that line
awk '{if ($3 > 30) {$3 = $3 + 900; print $0}}' → if using awk
awk '{print NR " " $0}' → Print row numbers
awk 'END {print NR}' → Print total row number / last row numbers
awk 'BEGIN {for (i=1; i<=10; i++) print "square of", i, "is", i*i}'

awk 'BEGIN {OFS = "-->"} {print $1, $2, $3}' file → OFS eg.
awk 'BEGIN {ORS = ":::"} {print $1}' file → ORS eg.
awk -f script file → Take script from file.
awk 'BEGIN {c=0} NR != 1 {c = c + $2} END {print c}' file
↳ ~~max length of a row~~ prints total age
awk '{if (length($0) > max) max = length($0)} END {print max}'
↳ max length of a row
awk 'length($0) > 10' file → Printing all lines with more that 10 characters.

grep → global regular expression print/parser

Searches for a expression & prints it. We can used it to search and print in a single or multiple files.

Syntax : grep [options] pattern [files]

Prints all matched lines.

Options

-c → Prints count of lines that match the pattern.
-h → Display the matched lines but not filenames.
-i → ignores, case for matching (uppercase lowercase)
-l → Displays list of matched filenames only
-L → Display list of non-matched filenames only.
-n → Display the matched lines & their line numbers.
-v → This prints out all the lines that do not maetch the pat
-e expression
↳ Specifies expression with this option
-f file
↳ Takes patterns from file, one per line

-w → Match whole word
-o → Print only matched parts of line
-q → Do not print anything to standard input
-r → Recursive
-A n → Prints searched line & n lines after result.
-B n → " " " before " "
-C n → " " after & before

Q1. Grep command for line having apple or orange but not banana.

~~grep -v -e~~

cat > file.txt | grep -E 'apple|oranges' | grep -v -E 'banana'

Q2. Grep command for mobile number.

grep -E '\b[0-9]{10}\b' file.txt

Q3. Grep command for a ~~sing~~ number in a line.

grep -E '[0-9]+' file

Egrep. → grep with -E option by default. It means that we have to write extended regex.

sort → used to sort a file. By default is sorts with ASCII values. (Lexicographically order). It sorts line by line. Blank space is the default field seperator.

Lines starting with number will appear before lines starting with a letter

Lines starting with an uppercase letter will appear after lines starting with the same letter in lowercase.

Syntax : Sort [options] filename

Options

sort a.txt → Sort in an increasing order.

sort -r a.txt → reverse order.

sort -o filename.txt inputfile.txt → Write output to a new file

sort -n a.txt → Sort on the basis of numbers (Counting).

sort -u a.txt → Remove duplicate lines. (Unique)

sort -k2 a.txt → Sort on the basis of mentioned column number.
Field separator is

-t → Change field separator

-c → Checks whether the file is sorted or not.

-M → Sort by month

-h → Sort to human readable numbers like 1G, 10G, 5t, 50M

-m → merge multiple sorted files

-C → to check file is sorted or not

-k <keydef> → to sort on a key

-k <start>, <end> → in a range

-f → to ignore case

uniq → used to filter out adjacent, duplicate lines in a file. i.e to display unique content in a file. Much like grep & sort it's used through pipelines to process and filter text output.

Syntax : uniq [options] file

-d → To display only duplicate lines
-c → no. of occurances of each line
-i → ignores case
-u → only unique lines (lines which are not duplicated)
-f N / -skip-fields(N) → to skip N fields
-s N → to skip N chars in each line
-w N → compares only N characters in a line
-z → if we don't want a newline after output.

wc command already done behind in ST1*

cmp → to compare 2 files byte by byte & helps to find out whether the files are identical or not.

only first difference shew.

when called it reports the location of first mismatch to the screen if difference is found and if no difference is found, cmp displays no message.

Syntax : cmp [option] file1 file2

Eg. $ cmp a.txt c.txt
Output : a.txt c.txt differ : byte 7, line 2

Options

-b → displays differing bytes in the output
-i [skip / skip1 : skip2] → if you wanna skip starting bytes of both files

-I → verbose
-s → supresses normal output. 0 → identical 1 → different 2 → error.
-n N → to limit no. of byts to compare to first N.

diff (difference) → compares line by line & tells us which lines in one file have is to be changed to make the two files identical. unlike cmp.
↳ uses special symbols.

Special Symbols

c = change
d = delete
a = add
< file 1
> file 2

Options

-c → context mode
-o → unified mode
-q → shows message while files different
-s → " " " same
-y → shows comparison line by line
-i → ignore case
-r → recursive

sdiff → used to compare 2 files & the writes the results in a side-by-side format.

Syntax: sdiff [-l | -s] [-o OutputFile] [-w Number] File1 File2

Options.

-l → displays only left side when they are identical.
-s → does not display identical lines
-w N → sets width of output. Default → 130
Max → 2048 Min → 20
-i → Ignore case
-o File → to store output in a file

Comm → compare 2 sorted files.

Syntax: comm filea.txt fileb.txt

Display results in 3 columns,
col-1 → Data present only in file a but not in fileb
col-2 → " file b but not in file a
col-3 → common in both files

Options
-1 → to not display col 1
-2 → to not display col 2
-3 → to not display col 3
-12 → for not 1 & 2 display.

Aaaa god my english- I am sleep.

File compression

Technique used to reduce the size of files or folders by encoding information efficiently.

- Improves memory utilization
- Transportation will become very easy.
- It reduces download times
- Sometimes enhances security

Common formats → ZIP, GZIP, TAR, RAR, 7z

This process has 2 steps :

Creating Archive file → Apply compression algorithms.

What is Archiving ?

→ Process of consolidating multiple files or directories into a single file called an "Archive".

tar. → We can group multiple files & directories into a single archive file by using tar command.

Syntax : tar [options] [Archive file] [file to be archived]

Options
- -c → creates archive
- -x → Extracts archive
- -f → creating archive with a given file name
- -t → displays or lists files in archive file

-v → Displays verbose info

-z → zip (gzip)

-r → update or add file or directly in already existed .tar file.

To create a tar file → tar -cvf demo.tar filea fileb filec

To display contents of tar file → tar -tvf demo.tar

To extract contents of tar file → tar -xvf demo.tar

gzip → To compress files & folders. Very Fast but less compression power.

To compress a file → gzip demo.tar

↓

this creates

↓

gzip demo.tar.gz

To uncompress a gz file → gzip -d demo.tar.gz

or

gunzip demo.tar.gz

System Utility Commands.

date → to display date & time. also used to set date & time. (admin)

Syntax : date [OPTION]... [+FORMAT]
date [-u | --utc | --universal] [MMDDhhmm[CC]YY[.ss]]

Various Date formats. written like * date +%D *
→ the letter

D → mm/dd/yy {only date}
T → hh:mm:ss {only time}
d → only day
m → month
y → year in yy
Y → year in yyyy
H → hrs in 24 scale formate
M → minutes
S → seconds
A → full day name
a → abbreaviated day name { tue, wed, sun, etc.. }
B → full month name
b → abbreaviated month name { jan, feb, ... }
w → display day of week _ 0 is Sunday 6 is Saturday
W → week number of year
q → quarter of year
j → day of year {1 → 366}
I → to display hr in 12 hr format
r → 12-hr clock format.

Eg. $ date "+%Y / %m / %d"

" 04/02/2024 " or " Feb 4 2024 "

Date Strings

Date flags

date -u → to display UTC time
date -s " string " → set date as string
date --date="string" / date -d " string " → past & future date
-r → last modified time
TZ = xxxxxx date → change timezone

" 2 years ago " → today's date 2 yrs ago.
" 5 sec ago " → time 5 sec ago
" next sun " → next sunday
" last month " → last month.

" + 2 day " or " 2 day " → time after 2 days
" - 2 weeks " → time 2 weeks ago

cal → a calander viewer. Command has to be installed by sudo apt get cal .

cal → to display as current month calander
Cal 2024 → to show calander for complete year.
cal 1 → 1st year in calander range {1 → 9999}

cal 08 2019 → To display august 2019 th calander.
cal -j → julian calender
cal -3 → past current & next month calender
cal -m 5 → 5th month calendr
cal -y 2024 → year 2024 calander
cal -1 → current month calendr (this is default)
cal -d string

uptime → to find out how long is system is active.

includes:

→ current time
→ amt of time the system is in running state
→ no. of users currently logged in/o
→ load time for past 1, 5 & 15 minutes

eg → 0.00 1, 0.05 0.10
Should be less that no. of process in the OS.

Syntax: uptime [options]

-p → Output pretty. Just like you!
-s → since when the system is running.

Hostname → used to obtain DNS (Domain name system) name and set the system's hostname or NIS (Network Information System) domain name. A hostname is a name given to a computer and attached to the network.

Syntax: hostname [options] [file]

-i → ip address of hostname
-I → list all IP addresses associated with system.
-s → short hostname
-f → fully qualified domain name (FQDN)
-d → to display domain name of system
-a → alias of hostname
-A → get all FQDNs
-b → set a hostname
-F → set the hostname.

Uname . → display information about system .

-a → all info of system
-v → current kernel version
-n → hostname
-s → kernel name
-r → release version
-o → os name
-i → hardware platform
-m → machine hardware name
-p → processor type .

which → allows users to search list of paths in $PATH environment variable & outputs the full path of the executable file of the command specified as an argument.

Syntax : which [options] filenames...

-a → display all occurrences of the file.

P.T.O.

bc . → basic calculator .

Syntax. bc [options] → shows interactive option
echo "expression" | bc [options] → take input from echo
bc [options] < file → takes input from file.

Options

-h → Help
-i → interactive mode
-l → math library
-w → give warnings for extensions to POSIX bc
-s → Process exactly POSIX bc language
-q → print normal GNU bc welcome.
-v → version no. & copyright

bc supports → Arithmetic operators, ++ --, assignment op, comparison & relational op, logical or boolean, math functions, conditional statements, iterative.

Mathematical functions

s(x) → sine of (x in radians)
c(x) → cos of x
a(x) → arctangent → returns radians.
l(x) → natural logarithm of x
e(x) → exponential function
j(n,x) : bessel function
sqrt(x) → square root of X
length(x) → no. of digits in X
read() → reads number from stdin.
scale(expression) → value of scale function
ibase & obase to change base of numbers.

/bin/bash ⟶ location of user's shell

uid → 0 → root uid → 1 to 99 → reserved for predefined accounts
uid → 100 - 999 → reserved for system accounts & groups.

useradd used to add user accounts to your system.

It makes the changes to the following files :-
/etc/passwd
/etc/shadow
/etc/group
/etc/gshadow

Creates a new directory for a new user in /home.

Syntax : useradd [options] [user_name]

Options

-c "SampleText" → add a comment to user.
-d "Home Dir" → To give a home directory path for new users, we use the -d option
-e "yyyy-mm-dd" → to give expiry date
-g gid → to assign a group to user
-G gids → multiple groups
-m → creates user's home directory, if it dosen't exist
-M → to create a user without a home directory
-p "password" → To specify a password
-s "path to shell" → users login shell
-u UID → custom UID
-K → create a User's Home directory using a custom skeleton
-N → create a user without a group.

normal user uid → 1000 to 60,000
root uid → 0
999 system users from 1 to 999

adduser → high level interface to 'useradd'. It is more user-friendly & interactively prompts for information such as password & user details.

→ automatically creates a home directory and sets up the user environment.

groupadd.

In Linux, there can be many users of system. In scenario, where there are many users, there might be some privileges that some users have and some don't, it becomes difficult to manage all the permissions at individual user level. So using groups, we can group together a number of users and set privileges an permissions for entire group.

Syntax : groupadd [option] group_name

For every new user it comes in /etc/group.

To verify → sudo tail /etc/group.

group_name : password : group_id : list of members.

-f, --force → silently aborts if group already exists with given name. or assigns a unique gID.

-g GID, -gid GID → assigns a specific group id.

-h → to help

-K KEY=VALUE → overrides default values set in /etc/login.defs

-O → permits adding a group with non-unique gID.

-p PASSWORD → sets an encrypted password to group.

-r → creates system group (changes, CHROOT_DIR)

usermod. → to change attributes of a user.

Information is stored in /etc/ folders name, passwd, group, shadow, login.defs, shadow, login.defs.

Syntax : usermod [options] username

OPTIONS

-c → comment field addition
-d → modify directory for existing user account
-e → account expire
-g → add primary group.
-G → add supplementary group
-a → add anyone from one group to another
-l → change login name
-L → lock
-m → move from one dir to another
-p → unencrypted password (idk why you would use this)
-s → Specified shell
-u → used to assign UID
-U → unlock the account.

userdel. → delete user account & related files.

Syntax : userdel [options] User Name

-f → to force removal

-r → recursive

-h → help

-R → applies changes in CHROOT_DIR & uses it.

groupdel. → delete existing group.

Syntax : groupdel [OPTIONS] GROUP.

The info is stored in /etc/group & /etc/shadow.

-f → force
-h → help
-R → root same as userdel.

P.T.O.

su Switch User. → Allows switching to another user account.

- → switches to root user
-l → simulates full login

Eg → su - Hardik

su	su -
Retain current environment variables.	Resets to target
keeps the current working dir.	Changes to target
keeps the shell settings.	Changes to target
PATH is not updated.	Updates.

sudo SuperUser DO. Executes command with elevated privileges.

-u → Run the command as a specific user.
-s → Run shell as another user.

sudo command
sudo -u username command
sudo -s -u username.

Monitor User Activity.

who To find → • Time of last system boot
• Current run level of systems
• List of logged in users.
↓
Name, Numbers, Time, Remote Hostname.

Syntax : who [options] [filename]

-a → all info
-H → prints header
-b → last boot time
-l → system login process
-m → hostname & user associated with stdin.
-r → current runlevel.

last.

list of all users logged in and out since the file '/var/log/wtmp' was created.

Syntax : last [options] [username ...] [tty...]

-n → last n entries
-R → to hide hostfield name
-F → the '-F' option can be used to display login & logout times.
-s -t → -s (since) -t (until) login entries within a specific time period.

nice kdrama ^^

W information about who is currently using the computer.

Syntax : w [options] user [...]

Columns	Description
USER	username of logged in user
TTY	terminal device
FROM	indicates remote host or IP address
LOGIN@	time of login
IDLE	duration of inactivity
JCPU	cpu time used by all processes of user
PCPU	percentage of cpu ↑
WHAT	info. about command or process running.

id. used to find out user & group names & numeric ids.

Syntax : id [option]... [user]

-g → effective GID
-G → all GIDs
-n → name instead of number
-r → real ID instead of numbers.
-u → only effective user id.
--help → help messages.
--version → version information.

Questions.

Q1. How to add a new group? → sudo groupadd group_name

Q2. How to add a new user? → sudo useradd username

Q3. switching from one user to another → su username

Q4. How to get information about particular user? → id username / finger username

Q5. How to delete users? → sudo userdel username

Q6. How to delete a group → sudo groupdel group_name

Q7. How to modify user ~~mode~~ info ? sudo usermod

eg → sudo usermod -l newname oldname

Q8. How to modify group info? → sudo groupmod

Q9. How to change ownership of a file? → sudo chown newowner file

Q10. How to change group ownership of file? → sudo chgrp newgpname file

Q11. Change group of user? → sudo usermod -g new old username.

Q12. Add user to multiple groups? → sudo usermod -G g1, g2, g3 username

Q13. Check available groups → cat /etc/group

Q14. Difference b/t adduser & useradd ? → adduser → interactive, high level
useradd → low level complex.

Q15. How to check the allowed commands by sudo for a particular reason ?

sudo -l -U username

Q16. Sudo command → allows a permitted user to execute a command as superuser or another user.

Q17. Sudoers file → contains rules & configurations that define which users can run specific commands with sudo.

---

ps. for viewing information related with the processes on a system which stands as "Process Status"

Syntax : ps [OPTIONS]

Result of PS →

PID → unique process ID

TTY → terminal type of user

Time → amount of time process has been running in CPU

CMD → name of command that launched the process

Options.

-A / -e → all processes

-a → view all processes except both session leaders and processes not associated with a terminal.

-aux → detailed list of all processes in a user friendly format

-f → full format listing

-d → all processes except session leader.

- N → negates the selection
Na → print both session leaders & processes not associated with a terminal
- T → all processes associated with terminal
-r → view all running processes
-x → runner of ps command
- F → extra full format
-l → long format

Process Selection →

-C → select process by command name
-G → select by group name
-g → view by groupid
-p → view by process id
-sid → by session id
-s → by session id
-t/--tty → select by tty
-u → by username
-o → by user defined format

Sleep. tell's computer to do nothing for a while.

Syntax : sleep [time]

sleep 5 → 5 seconds
sleep 5m → 5 months
sleep 5d → 5 days.

Jobs

+ →

-p →

-s

ctal

bg.

Syn

%

%

%

%

%

Same fo

fg

S

Jobs. give information about jobs which are in background or foreground.

+ → last job - → second last job & → indicates job that is running in background

-p → to give Process IDs -r → only running jobs

-s → only stopped jobs -l → long form ctrl+c → kill foreground job

ctrl+z → suspend foreground job.

bg. used to place foreground jobs in background

Syntax : bg [job_spec..]

%n → refer to Job number n

%str → job which was started with a command starting with str

%?str → " containing str

%% or %+ → refer to current job

%- → refer to previous job

Same for **fg**.

fg. put a background job in foreground

Syntax : fg [job_spec..]

nice. Each process has a priority assigned to it. nice & renice are used to change the priority of a process

nice applies priority before the process has to run & renice ~~after~~ during.

Syntax. nice -nice_value arguments

Lower the nice value, higher the priority. Ranges from -20 to 20 (most ↓ least)

Eg. nice -5 command

sudo nice --5 command

↳ * negative values can only be assigned by sudo. *

Networking Commands.

Traceroute. used to trace the pathway that data packets take from local machine to destination host or server.

Syntax: traceroute [options] destination

Eg. traceroute google.com

→ no. of number.

-n → numeric IP address

-q # → sets no. of queries per hop (3 is default)

-w # → sets timeout value of each probe ($\frac{5}{sec}$ is default)

-m # → sets maxm. no of hops

-I → uses ICMP echo request

-T → uses TcP SYN packets

-U → uses UDP.

Nslookup. (Name Server Lookup)

used to query DNS servers to obtain info like, IP, hostname and other records.

Eg. nslookup google.com

or nslookup 157.240.16.35 → for reverse lookup

Options.

-domain = [domainname] → to change default dns name

-debug → debugging information

-port = [portno.] → use port no. for queries

-timeout = [seconds] → to specify timeout time

-type = a → Lookup for a record

-type = any → any record

-type = hinfo → for hardware related info.

ipconfig. → command used to display detailed information about the network we are connected to.

ifconfig. → used to configure and display information about network interface.

Syntax ifconfig [options] [interface]

- a → Display information for all interfaces
- s → short list of full details.

up → activate an interface

Down → deactivate an interface

to change ip / netmask / broadcast address → ifconfig interface addresstype address.

netstat. → displays various network related information like connections, routing table, interface statistics, masquerade connections & multicast memberships.

Options. **Syntax:** netstat [options]

- a → shows both listening & non listening sockets
- at → all tcp connection
- au → all udp
- l → all listening states
- lt → only tcp listening
- s → ~~statfic~~ statistics protocol wise
- st → tcp
- su → udp
- p → PID
- n → numeric port
- c → continuous statistics
- I → interface
- ie → extended interface
- r → routing table

ctrl + c to stop Ping

dig. (Domain Information Grouper)

Provides detailed information about DNS records and can help diagnose DNS-related issues.

For eg. dig google.com.

ctrl+c to stop

Ping. (Packet Internet Groper)

- Test connectivity
- Check server availability
- Troubleshoot network issues.

Ping used to check Network connectivity.

Syntax: ping [options] hostname/ip

Headers.

From → tells target & its IP

Important → different geographical addresses

ttl = n → n from 1-255 (Time to live)

icmp_seq = n → sequence no.

time = n ms → tells the time it took for packet to reach and come back. (in ms)

Local Network Check.

- ping 0
- ping localhost
- ping 127.0.0.1

Options

-4 / -6 → to specify IPv4 / IPv6

-i → lower ping time interval

sudo ping -f → ping flood for network testing
-s 1000 → increase packet size
-c 2 → Stop after 2 packets
-w 25 → Stop after certain time
-c 10 -q → quite mode with statistics.
-D → Timestamp before each line.

Additional Options.

-a → sound on successful ping
-b → broadcast IP ping
-B → prevent probe source address change
-c → limited transmitted requests
-d → set SO-DEBUG option
-f → Network flood
-i → interval b/w packet transmission
-I → set source of IP
-l → Specify no. of packets without delay
-q → show IP addresses
-t → set TTL
-v → verbose
-V → show ping version & exit.

Web & Database Services :

Configuring web servers

Apache Web Server Config.

installation → sudo apt update
sudo apt install apache2

Basic Config → • Main Configuration File : '/etc/apache2/apache2.conf'
• Modify settings like server name, ports, user/group and global configurations.

Virtual Hosts → • Sites Configuration Direction : '/etc/apache2/sites-available/
• Create seperate configurations for different websites
• Enable sites 'a2ensite <site-name>

Security & Permissions : • Utilize ".htaccess" files for directory-specific configurations
• Set file permissions properly to restrict access.

Modules & Extensions. : "a2enmod <module_name>"
↓
mod_rewrite, mod-ssl, mod_proxy.

Logging & Monitoring : Utilize tools like "tail" & "grep" to monitor logs on "/var/log/apache2"

After making Changes Reset

sudo systemctl restart apache2
sudo systemctl restart httpd

Nginx

Install → sudo apt install nginx.

Basic Config → /etc/nginx/nginx.conf

Sever Blocks → /etc/nginx/sites-available.
rest same to Virtual Hosts in Apache.

Security → enable → SSL/TLS in nginx.conf

Logging & Monitoring → /var/log/nginx

Restart → sudo systemctl restart nginx.

Setting up databases.

MySQL.

installing → sudo apt install mysql-server

configuring → sudo mysql-secure_installation
sudo mysql -u root -p
Now you can creat databases & all

Eg. Create database dbname
create user 'username'@'localhost' identified by 'passwor'
grant all privilages on dbname.* to 'username'@'localhost';
flush privileges.

PostgreSQL.

install → sudo apt install postgresql postgresql-contrib

configuring → sudo -i -u postgres
createuser --interactive
createdb dbname.

Basic Usage.

MySQL.
- Start service → sudo systemctl start mysql
- Access MySQL shell → mysql -u username -p
- Run SQL commands for managing dbs, tables & all.

PostgreSQL.
- start service → sudo systemctl start postgresql
- Access shell → psql -U username -d dbname
- Use SQL commands again.

Automation & Scripting.

- Shell Scripting
- Python Scripting
- Automation Tools (Ansible, Puppet, Chef, SaltStack)
- Cron Jobs
- VCS
- Monitoring & Alerting

Using Configuration Management Tools.

Ansible

Open Source Automation Tool for configuration management, application deployment & task automation. Uses declarative language to describe system configurations known as playbooks written in YAML format. connects to nodes via SSH by default.

install → sudo apt install ansible

Study more about this!

Shell Scripting.

A shell script is a type of computer program developed to be executed by a Unix-shell, which is also known as command-line interpreter. Several shell script dialects are treated as scripting languages.

Classic operations implemented by shell scripts containing printing text, program execution & file manipulation.

Capabilities of SHELL.

Comments → start with '#'

Scripting language configurable choice → with '#!' she-bang

Shell also does → Batch jobs, Generalization & Programming

How to determine Shell ?

echo $SHELL → by default 'bash'

* She-Bang *

The sign #! is called she-bang is written at top of script. It passes instruction to program /bin/~~bash~~ sh

To start your script in a certain shell use :-

```
#!  /bin/bash
echo Hello World
#!  /bin/ksh
echo Hello world
```

Shell Scripting Variables. → Environment variables ↗ kinda system variables
→ User defined variables

How to assign variables → var1 = Hello $var2 = 10 ↗ read only
echo "$var1"

Sourcing A file. → ./<filename> or source [filename]

Rules to define a variable
- should not start with digits
- no special symbol except _
- no spaces.

Steps to write & execute a script.

- Open terminal. Go to directory where you want to create the script
- Create .sh extension
- Write a script using file editor like gedit
- Make the .sh file executable by, chmod +x <filename>
- Run script using ./<filename>

Types of Shell.

Bourne Shell
- Developed by Stephen Bourne
- First ever shell for Unix
- we can use this with sh command

Bash shell
- Bourne Again Shell
- Advanced version of Bourne Shell
- Default shell for most linux distros

Korn Shell
- By david korn
- Used by IBM AIX OS
- Used by Ksh command

Cshell.
- by Bill Joy
- C is for California University
- also default available with Unix
- Using csh command

if you wanna switch shells → just write the shell name

Tshell
- Teaminal for T
- advanced version of CShell
- used in HP Unix
- tcsh command

Zshell
- Developed by Paul
- By using zsh command

How to check all available shells in our system?

→ cat /etc/shells

Example Script.

```
#! /bin/bash
a = 10
b = 20
COURSE = "linux"
ACTION = "SLEEP"
echo "values of a & b are : $a and $b"
echo " Course is : ${COURSE}"
echo " Action is : $ACTIONING"
```

Command Substitution

```
echo " Today date is : $(date +%D)"
echo " Present working directory is : $(pwd)"
```

Command Line Arguments.

The arguments which are passing from the command prompt at the time of executing our script, are called command line arguments.

```
                  1        2      3     4     5
$ ./test.sh  learning  linux  is  very  easy
```

$# → result : 5
$1 → learning
$2 → linux
$3 → is
$4 → very
$5 → easy

→ Command Line Arguments

$? → Represents exit code.

Difference b/w $@ & $*

$@ → All command line arguments with space separator

$* → All command line arguments as single string

Eg Script

```
#! /bin/bash
IFS = "-"
echo " no. of arguments : $# "
echo " Arguments seperated by IFS : $* "
echo " Argument seperated by space : $@ "
```

./test.sh My Name is Harshpreet Singh

Output

```
no. of arguments : 5
Arguments by IFS : My - Name - is - Harshpreet - Singh
Arguments by space : My Name is Harshpreet Singh
```

* for a prompt message * * -s → to hide * input

↳ read -p "Enter A Value: " A

Q: Shell script to check length of given command line argument.

Script.

```
#!/bin/bash
len = $(echo -n "$1" | wc -c)
echo "The length of given string is $1 : $len"
```

Q: How to read Dynamic data for use.

```
read a b
100 200
echo "a = $a, b = $b"
a = 100, b = 200
```

what if there are more than 2 inputs here

eg → 100 200 300 400

Output → a = 100, b = 200 300 400

Q. Write a shell script which takes input data of an employee and add that to a file.

Shell.

```
#! /bin/bash
read -p "Id: " eid
redd -p "Name: " ename
read -p "Salary: " esal
read -p "Role: " erole

echo "$eid : $ename : $esal : $erole" >> empdata.txt
```

Note :— It's always preferred to have variable names in capital letters. eg→ EID, ENAME.

Q. Write a script to read a file name from user and then display its contents & the delete it's blank lines.

```
#! /bin/bash
read -p "Filename : " fname
cat $fname

sed -i '/^$/d' "$fname"
cat $fname
```

To remove duplicate line → `sort -u $fname > temp.txt`
`mv temp.txt $fname`

Operators woahhh what a O.

Arithmetic → +, -, *, /, %, ++, --

Relational → `==` `!=` `<` `<=` `>` `>=`
`-eq` `-ne` `-lt` `-le` `-gt` `-ge` → for [ ] brackets

Logical → `&&` `||` `!`
`-a` `-o` `!` → for [ ] brackets

Bitwise → & | ^ ~ << >>

Assignment → =

Types to perform mathematical operations.

1) By using expr keyword

```
sum = $(expr $a + $b)
```

2) By

let

3) By

sum

4) By

Sum

Q.

If we want decimal output of calculations
↳ sum = $(echo $a + $b | bc)

2) By using let keyword

let sum = a + b or let sum = $a + $b

3) By using (())

sum = $((a+b))

4) By using []

sum = $[a+b] or sum = $[$a + $b]

Q. Script to read a 4 digit no. & print sum of those digits.

```
#! /bin/bash
read -p "Enter no.: " n
a = $(echo $n | cut -c 1 )
b = $(echo $n | cut -c 2 )
c = $(echo $n | cut -c 3 )
d = $(echo $n | cut -c 4 )

echo " sum : $[a+b+c+d] "
```

P.T.O.

Control Statements.

if

simple if → if [condition]
then
action
fi

if - else → if [condition]
then action
else action
fi

Nested if → if [condition]
then
if [condition]
:

ladder if elif else → if [condition] then action
elif [condition] then action
else action
fi

Q. Find greater of 2 numbers.

```
#! /bin/bash
read a
read b
if [ $a -gt $b ] then
echo " $a is greater "
else
echo " $b is greater "
fi
```

Q. Script to read marks in 3 subjects if any has less than 33 then fail.

```
#! /bin/bash
read a b c
if [ $a -gt 33 -a $b -gt 33 -a $c -gt 33 ]
then echo "pass"
else echo "fail"
fi
```

Some Practice Questions → Check no. is +ve or -ve?
Even or Not?
3 Digit or Not?

File test options

-e → file exists or not
-s → true if file is non empty
-f → true if regular file
-d → true if a directory
-l → true if link file
-b → true if block special file
-c → true if character special file
-r → if user has read perm.
-w → " write "
-x → " executable "
-o → " owner "
file1 -ot file → f1 older than f2
f1 -nt f2 → f1 newer than f2

Q. Check whether file is regular or directory?

```
#! /bin/bash
read fname
if [-e $fname ] then
    if [-f $fname ] then
        echo " regular file "
    elif- [-d $fname ] then
        echo " Directory file "
    else echo " special file "
    fi
else
   echo " $fname dosen't exist "
fi
```

String Test options.

str1 = str 2 → if they are same or not

str1 != str 2 → if they are not equal

-z str1 → if str is empty

-n str1 → not empty

str1 > str 2 → str1 is alphabetically greater

str1 < str 2 → str 2 is alphabetically greater.

Case Statement :

```
case $variable in option 1) action - 1
;;
option 2)
action - 2
;;
option 3)
action - 3
;;
4)
default action
;;
esac
```

Q. take digit from 0 - 3 & print digit.

```
#! /bin/bash
read n
case $n in
0) echo "zero" ;;
1) echo "one" ;;
2) echo "two" ;;
3) echo "three" ;;
*) echo "other" ;;
esac
```

Q. Check if input character is digit, alphabet or special symbol.

```
#! /bin/bash
read n
case $n in
[A-Z a-z]) echo "alphabet" ;;
[0-9]) echo "digit" ;;
*) echo "special symbol" ;;
esac
```

Iterative Statements.

while loop.

```
while [condition]
do body
done
```

Q. Print 1 - 10.

```
#!/bin/bash
i=1
while [$i -ne 10 ]
do
echo $i
i=$[i+1]
done
```

* echo -n if you want them in same line.

Q. sum of first n integers.

```
#! /bin/bash
read n
i=1
sum=0
while [ $i -le $n ]
do
  let sum=sum+i
  let i++
done
echo " Sum of first $n numbers : $sum "
```

Use Break to break out of while based on a condition. Like in our normal programming.

Q. Script to run timer till given hr & min.

```
#! /bin/bash
while [true]
do
  clear
  printf "\n\n\a\n\n\t\t\t $(data +%H:%M:%S )"
  sleep 1
  h= $(date +%H)
  m= $(date +%M)
  if [ $h -eq 8 -a $m -eq 54 ]; then break
  fi
done
```

Similarly we have continue.

Q. Take a string as input & print it's reverse.

```
#! /bin/bash
read str
len = $(echo -n $str | wc -c)
ans = " "
while [ $len -gt 0 ]
do
ch = $(echo -n $str | cut -c $len)
ans = $ans $ch
len = $[len - 1]
done

echo "Reversed $ans"

if [ $str = $ans ] then echo "palindrome" fi
```

Until loop.

```
until [condition] do
body
done
```

Q. 1 to 5 using until.

```
#! /bin/bash
i = 1
until [ $i -gt 5 ]
do
echo $i
let i++
done
```

For loop.

```
for variable in item 1 item 2 ... item N
do action
done
```

Q. 1 to 5 using for loop.

```
#! /bin/bash
for i in 1 2 3 4 5
do
echo $i
done
```

Q. 1 - 100 only numbers which are divisible by 10

```
#! /bin/bash

for i in {1...100}
do
if [ $[i%.10] -eq 0 ]
then
echo $i
fi
done
```

P.T.O.

Q. Print all CLA.

```
#! /bin/bash
if [ $# -ne 0 ]
then
for arg in $@
do
echo "$arg"
done
else
echo "no CLA passed"
fi
```

Alternative syntax

```
#! /bin/bash
read n
for ((i=0; i<n; i++))
do
 echo $i
done
```

Q. Prime or Not.

```
#! /bin/bash
read n
prime=true
for (( i= 2; i<= $n /2 ; i++ )) : do
    if [ $((n% i)) -eq 0 ] ; then prime = false break
    fi
done
echo $prime
```

Exit Codes / Status Codes

Every command and script return some value after execution, which indicates that whether it is successfully executed or not. This return value is called exit code or status code.

We can find the exit code by using "$?"

0 → successfully
non-zero → not successful.

Jff, example of arrays too.

```
read n
for ((i=0;i<n;i++)) do read nums[$i] done
oddsum=0
evensum=0

for ((i=0;i<n;i++)) do
    if [ $((nums[$i]%2)) -ne 0 ] then
        oddsum=$((oddsum+nums[$i]))
    else
        evensum=$((evensum+nums[$i]))
    fi
done

echo $oddsum
echo $evensum
```

Jff, example of function too.

```
calc() {
    if [ $# -ne 2 ] then
        echo " pass exactly 2 arguments "
else
    a=$1
    b=$2
    echo " $a + $b = $((a+b)) "
    echo " $a - $b = $((a-b)) "
    echo " $a * $b = $((a*b)) "
    echo " $a / $b = $((a/b)) "
```

```
calc  10  20
calc  50  2
calc  4   2
calc  19  12
```

```
factorial() {
    original=$n
    fact=1
    while [ $n -gt 1 ] do
        let fact = fact * n
        let n--
    done
    echo $fact
}
read n
factorial $n
```

GCC (GNU Compiler Collection)

GCC is an open-source compiler system developed by the GNU project. It supports programming languages such as C, C++, Objective-C, Fortran, Ada and more. Default compiler for many Linux distros.

Syntax : gcc [options] filename.c -o executable_name

Options :

-c → Compiles source files without linking to produce object file.

-O → Enables code optimization

-g → generates debugging information in output file, aiding debugging tools to trace program execution & find bugs.

-Wall → Enable all warning messages

-std=<standard> → specifies version of compiler eg -std=c11 or -std=c++17

Flow :