**Linux Command and Shell scripting**

1. Flavour of UNIX

* Debian.
* Gentoo.
* Ubuntu.
* Linux Mint.
* Red Hat Enterprise Linux.
* CentOS.
* Fedora.
* Kali Linux.
* Etc.

1. Features of UNIX/LINUX

* Freeware
* Open-source software
* Multiuser
* Multitasking
* Supports both GUI and CUI
* More Secure.

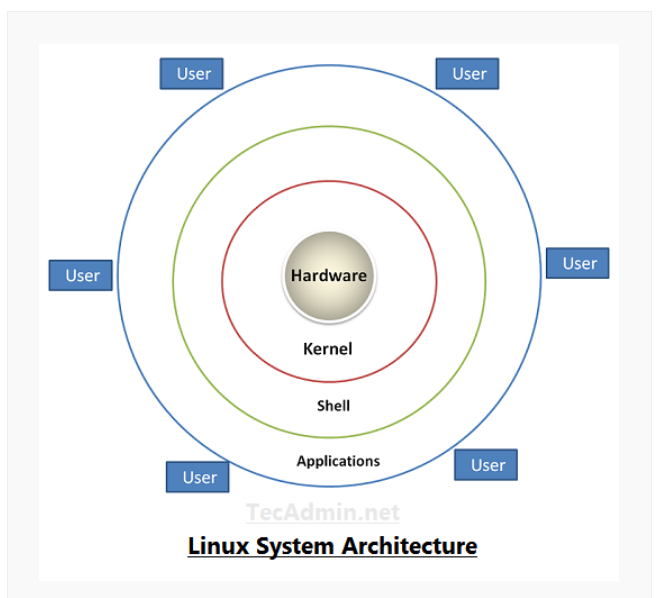
1. Components of UNIX/LINUX
   1. **Shell:**

* Shel it is the outer layer of UNIX operating system.
* Shell reads commands provided by users.
* Shell will check whether is it valid command or not.
* Shell will check whether is it properly used or not.
* If everything is proper, then shell interprets/converts the commands into kernel understandable form and handover that to the kernel.
* Shell acts as interface between user and kernel
  1. **Kernel**
* It is the core component of UNIX operating system.
* It is responsible to execute our commands with the help of hardware components.
* Memory allocation and processor allocation will takers care by kernel.
* It acts as interface between shell and hardware components.

1. Types of Users in Linux

* Super user/root users/admin user **🡪** **#** prompt
* Normal user 🡺 **$** prompt

1. Linux Architecture



1. Types of Files in Linux

In Linux everything is treated as Files

1. Normal Files or Ordinary Files
2. Directory Files
3. Device Files
4. Normal Files or Ordinary Files:These files contain data. It can be normal text file or binary files (images, video files, audio files etc)

Example:

* abc.txt
* test.sh 🡪 Script file
* test.py 🡪 Python file
* test.java 🡪 Java file
* Sunny.jpg 🡪 binary file
* Movie.mp4 🡪 binary file

Note: - In Linux file extension is not important. Based on our content the Linux can identify file type.

1. Directory Files:These files represent directories. In windows, we can use folder terminology but Linux we can use directory terminology.

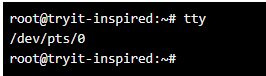
Directory contains files and sub directories.

1. Device Files: In Linux, every device is represented as a file. By using this file we can communicate with that device.

Inside /dev directory all device related files will be there.

Terminal is also treated as a file.

$ tty



Terminal 1 🡪 /dev/pts/0

Terminal 2 🡪 /dev/pts/1

By using “>” directional we can communicate from terminal 1 to terminal 2

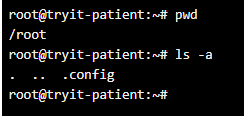
Control + Alt + t 🡪 To open the terminal in ubuntu.

Control + d 🡪 To close the terminal in ubuntu.

1. File System Navigation commands

To display all the hidden files

ls -a



. 🡪 represents current directory.

.. 🡪 represents parent directory.

/home/ubuntu

ubuntu is current directory.

home is the parent directory.

$ cd . 🡪 current directory

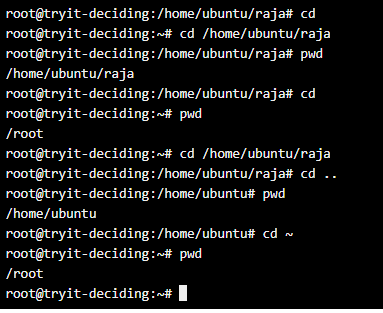
$ cd .. 🡪 changes to parent directory.

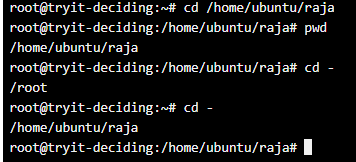
$ cd ../../... 🡪

$ cd ~ 🡪 to go to user home directory.

$ cd - 🡪 if we want to go to previous working directory.

$ cd 🡪 to go to root directory.



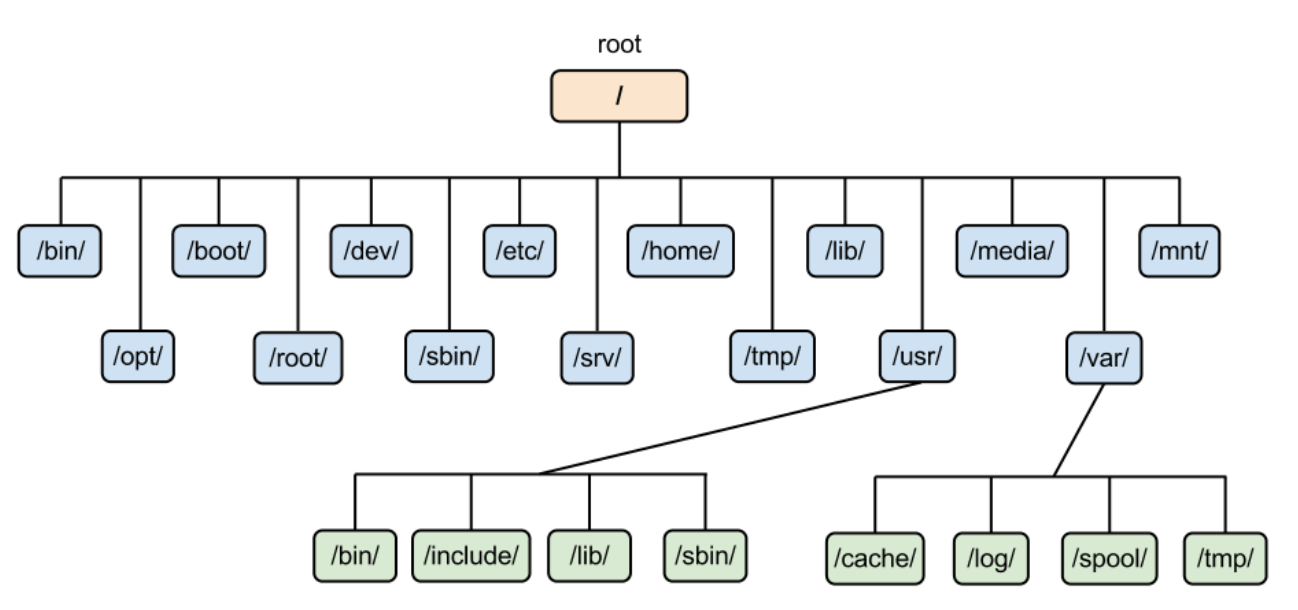


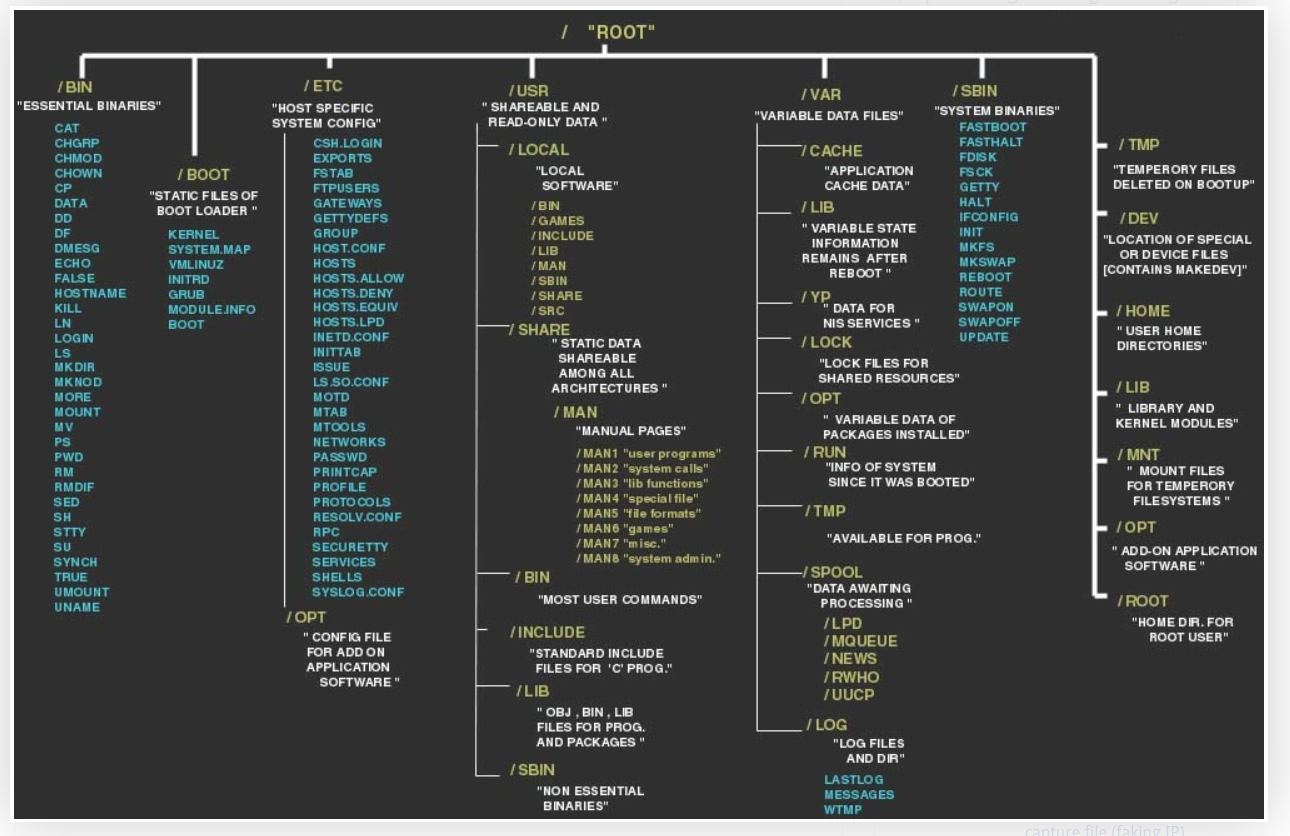
1. Linux File system Hierarchy

* Linux file system has TREE like structure.
* It starts with root (/)
* / is the top most directory

Sub directories:

* bin
* etc
* home
* lib
* dev
* user





**DIRECTORY DESCRIPTION**

|  |  |
| --- | --- |
| **/** | The root directory |
| **/bin** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

/ Primary hierarchy root and root directory of the entire file system hierarchy.

/bin Essential command binaries that need to be available in single user mode, for all users, e.g., cat, ls, cp.

/boot Boot loader files, e.g., kernels, initrd.

/dev Essential devices, e.g., /dev/null.

/etc Host-specific system-wide configuration files. There has been controversy over the meaning of the name itself. In early versions of the UNIX Implementation Document from Bell labs, /etc is referred to as the etcetera directory, as this directory historically held everything that did not belong elsewhere (however, the FHS restricts /etc to static configuration files and may not contain binaries). Since the publication of early documentation, the directory name has been re-designated in various ways. Recent interpretations include backronyms such as “Editable Text Configuration” or “Extended Tool Chest”.

/opt

Configuration files for add-on packages that are stored in /opt/.

/sgml

Configuration files, such as catalogs, for software that processes SGML.

/X11

Configuration files for the X Window System, version 11.

/xml

Configuration files, such as catalogs, for software that processes XML.

/home

Users’ home directories, containing saved files, personal settings, etc.

/lib

Libraries essential for the binaries in /bin/ and /sbin/.

/lib<qual>

Alternate format essential libraries. Such directories are optional, but if they exist, they have some requirements.

/media

Mount points for removable media such as CD-ROMs (appeared in FHS-2.3).

/mnt

Temporarily mounted filesystems.

/opt

Optional application software packages.

/proc

Virtual filesystem providing process and kernel information as files. In Linux, corresponds to a procfs mount.

/root

Home directory for the root user.

/sbin

Essential system binaries, e.g., init, ip, mount.

/srv

Site-specific data which are served by the system.

/tmp

Temporary files (see also /var/tmp). Often not preserved between system reboots.

/usr

Secondary hierarchy for read-only user data; contains the majority of (multi-)user utilities and applications.

/bin

Non-essential command binaries (not needed in single user mode); for all users.

/include

Standard include files.

/lib

Libraries for the binaries in /usr/bin/ and /usr/sbin/.

/lib<qual>

Alternate format libraries (optional).

/local

Tertiary hierarchy for local data, specific to this host. Typically has further subdirectories, e.g., bin/, lib/, share/.

/sbin

Non-essential system binaries, e.g., daemons for various network-services.

/share

Architecture-independent (shared) data.

/src

Source code, e.g., the kernel source code with its header files.

/X11R6

X Window System, Version 11, Release 6.

/var

Variable files—files whose content is expected to continually change during normal operation of the system—such as logs, spool files, and temporary e-mail files.

/cache

Application cache data. Such data are locally generated as a result of time-consuming I/O or calculation. The application must be able to regenerate or restore the data. The cached files can be deleted without loss of data.

/lib

State information. Persistent data modified by programs as they run, e.g., databases, packaging system metadata, etc.

/lock

Lock files. Files keeping track of resources currently in use.

/log

Log files. Various logs.

/mail

Users’ mailboxes.

/opt

Variable data from add-on packages that are stored in /opt/.

/run

Information about the running system since last boot, e.g., currently logged-in users and running daemons.

/spool

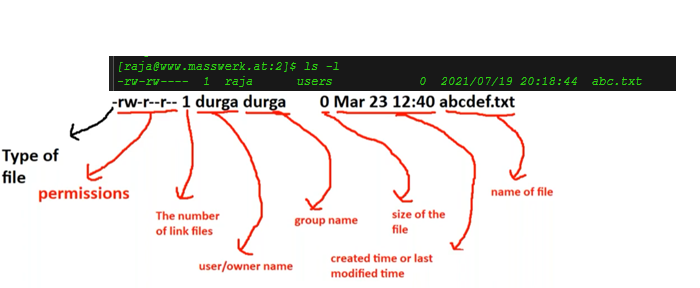
Spool for tasks waiting to be processed, e.g., print queues and outgoing mail queue.

/mail

Deprecated location for users’ mailboxes.

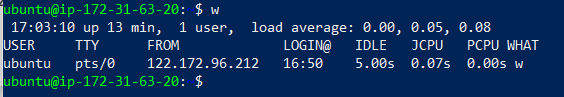
/tmp

Temporary files to be preserved between reboots.

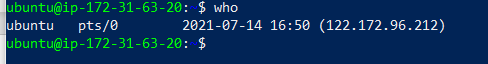


1. To check the user logged into the Linux machine.

$ W



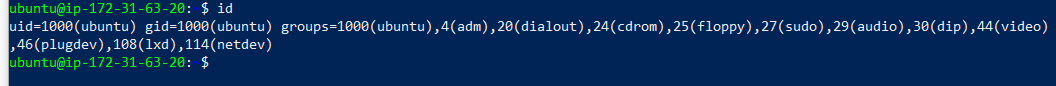
$ who



$ whoami



$ id

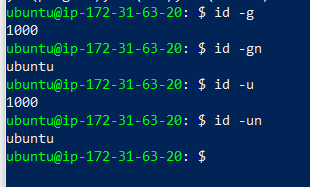


$ id -g

$ id -gn

$ id -un

$ id -u



1. To check which shell you have logged into the Linux machine.

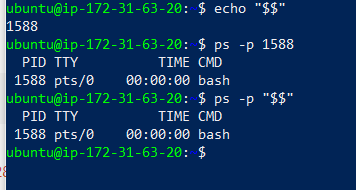
Unix provides different flavors of shells:

* Bourne shell (sh)
* C shell (csh)
* TC shell (tcsh)
* Korn shell (ksh)
* Bourne Again shell (bash)

$ echo “$$”

$ ps -p <PID>

$ ps -p “$$”



**Shell scripting**

**echo.sh**

#!/bin/bash

echo "Scripting is fun!"

$ chmod 755 script.sh

$ ./script.sh

**Output:**

Scripting is fun!



**Shebang**

#!/bin/csh

echo "This script uses csh as the interpreter."

#!/bin/ksh

echo "This script uses ksh as the interpreter."

#!/bin/zsh

echo "This script uses zsh as the interpreter."

**Sleepy.sh**

#!/bin/bash

sleep 90

$ ./sleepy.sh & [1]

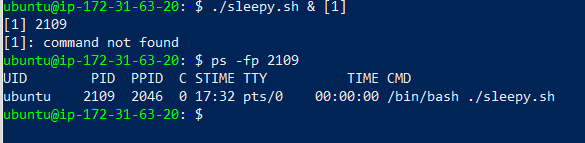
16796

$ ps -fp 16796

UID PID PPID C STIME TTY TIME CMD

jason 16796 16725 0 22:50 pts/0 00:00:00 /bin/bash ./sleepy.sh

**Output:**



**Variables**

* Storage locations that have a name
* Name-value pairs

**Syntax:**

**VARIABLE\_NAME="Value"**

* Variables are case sensitive
* By convention variables are uppercase

**Variable Names**

Valid:

* FIRST3LETTERS="ABC"
* FIRST\_THREE\_LETTERS="ABC"
* firstThreeLetters="ABC"

Invalid:

* 3LETTERS="ABC"
* first-three-letters="ABC"
* first@Three@Letters="ABC"

**variable.sh**

#!/bin/bash

MY\_SHELL="bash"

echo "I am ${MY\_SHELL}ing on my keyboard."

**Output:**

I am bashing on my keyboard.

**hostname.sh**

#!/bin/bash

SERVER\_NAME=$(hostname)

echo "You are running this script

on ${SERVER\_NAME}."

**Output:**



**Making Decisions - The if statement**

if [ condition-is-true ]

then

command 1

command 2

command N

fi

Example: **if.sh**

#!/bin/bash

MY\_SHELL="bash"

if [ "$MY\_SHELL" = "bash" ]

then

echo "You seem to like the bash shell."

fi

**Output:**



**If/else**

if [ condition-is-true ]

then

command N

else

command N

fi

Example: **ifelse.sh**

#!/bin/bash

MY\_SHELL="csh"

if [ "$MY\_SHELL" = "bash" ]

then

echo "You seem to like the bash shell."

else

echo "You don't seem to like the bash

shell."

fi

**if/elif/else**

if [ condition-is-true ]

then

command N

elif [ condition-is-true ]

then

command N

else

command N

fi

Example: **ifelifelse.sh**

#!/bin/bash

MY\_SHELL="csh"

if [ "$MY\_SHELL" = "bash" ]

then

echo "You seem to like the bash shell."

elif [ "$MY\_SHELL" = "csh" ]

then

echo "You seem to like the csh shell."

else

echo "You don't seem to like the bash or csh shells."

fi

**FOR LOOP**

for VARIABLE\_NAME in ITEM\_1 ITEM\_N

do

command 1

command 2

command N

done

**Command Line Arguments**

What are command line arrguments

How to pass commands line arguments

Inside script how we can access these command line arguments

The arguments which are passing from command prompt

$ ./test.sh 10 20 30 40 50

$ ./test,sh learning Linux is very easy

Learning,linux,is,very,easy

$# 🡪 Number of arguments (5)

$0🡪 Script name (./test.sh) test.sh 🡪 /home/user/test.sh

$1 🡪 First arguments (learning)

$2 🡪 Second arguments (Linux)

$3 🡪 Third arguments (is)

$4 🡪 Four arguments (very)

$5 🡪 Fifth arguments (easy)

$\* 🡪 Number of arguments (learning Linux is very easy 5)

$@ 🡪 Number of arguments (learning Linux is very easy)

$?🡪 Represents exit code of previously executed command or script.

Example: **arg.sh**

#! /bin/bash

echo "The Number of arguments: $#"

echo " Script Name: $0"

echo " First Argument: $1"

echo " Second Argument: $2"

echo " Third Argument: $3"

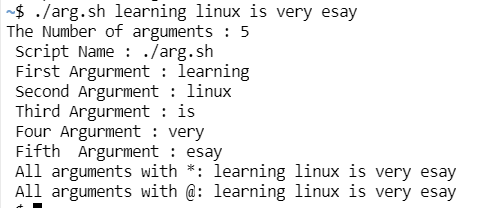
echo " Four Argument: $4"

echo " Fifth Argument : $5"

echo " All Arguments with \*: $\*"

echo " All Arguments with @: $@"

$ ./arg.sh Learning Linux is very easy.



Difference between $@ and $\*:

$@ 🡪 All command line arguments with space separator

“$1” “$2” “$3” “$4” “$5”

Learning Linux is very easy

$\* 🡪 All command line arguments as as single string

“$1c$2c$3c$4c$5c”

Where c is the first character of IFS (Internal Field Separator)

The default is space

Learning Linux is very easy

How to check default IFS:

$ set | grep “IFS”

IFS=$' \t\n'

set or env can be used

Example: **arg1.sh**

#! /bin/bash

IFS="-"

echo " All arguments with \*: $\*"

echo " All arguments with @: $@"

**output:**

All argument with \*: learning-Linux-is-very-easy

All argument with @: learning Linux is very easy

$@ 🡪 Separate entity

$\* 🡪 Single entity

What is the main purpose of command line arguments?

To customize behaviour of the script

echo “APPLE” | wc -c

output = 6 (apple count is 5 and one more character is for \n i.e., next line)

echo -n “APPLE” | wc -c

output = 5

Example: **length.sh**

#! /bin/bash

len=$(echo -n "apple" | wc -c)

echo "The length of given string:$len"

**output**

The length of given string: 6

$(cmd) 🡪 command substitution

len=$(echo -n "apple" | wc -c)

$len 🡪 variable substitution

Example: **length.sh**

#! /bin/bash

len=$(echo -n "$1" | wc -c)

echo "The length of given string $1: $len"

**Run:**

./length.sh apple

**output**

The length of given string: 5

Write a script to create log file with timestamp:

echo "Hello" > date.txt (it will overwrite the data)

echo "Hello" >> date.txt (it will append the data)

Example: **log.sh**

#! /bin/bash

timestamp=$(date +%d\_%m\_%Y\_%H\_%M\_%S)

echo " This is data to log file " >> ${timestamp}.log

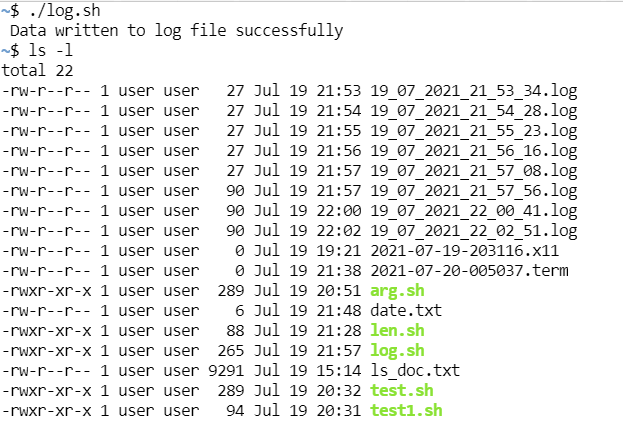
echo " This is extra data to log file " >> ${timestamp}.log

date >> ${timestamp}.log

echo >> ${timestamp}.log

echo " Data written to log file successfully"

**Output**



timestamp=$(date +%d\_%m\_%Y\_%H\_%M\_%S) 🡪 log file created for every second

timestamp=$(date +%d\_%m\_%Y\_%H\_%M) 🡪 log file created for every Minute

timestamp=$(date +%d\_%m\_%Y\_%H) 🡪 log file will be created for every Hour

timestamp=$(date +%d\_%m\_%Y) 🡪 log file will be created for every month

timestamp=$(date +%Y) 🡪 log file will be created for year

**How to view content of the file:**

We can view content of the file by using the following commands:

* cat
* tac
* rev
* head
* tail
* more
* less

**cat command:**

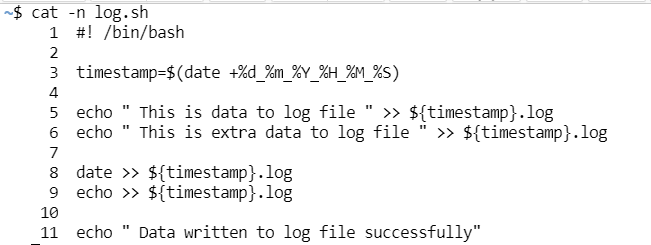
cat < file1.txt

Where < is optional

cat file1.txt

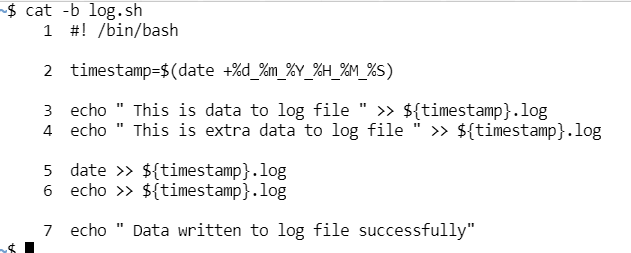
while display data if we want line numbers, we have to use -n option

cat -n file1.txt



Numbering skipped for blank lines, we have to user -b option

cat -b file1.txt



To view multiple files content simultaneously

Cat file1.txt file2.txt file3.txt

Various utilities of cat command:

1. create new file with some content.

cat > file1.txt

ctrl + d 🡪 to save and exit.

1. To Append some extra data to an existing file

cat >> file1.txt

Some more extra data

ctrl + d 🡪 to save and exit.

1. To view content of the file

cat < file1.txt or cat file1.txt

1. To copy content of one file to another file

cp file1.txt file2.txt

cat file1.txt > file2.txt

1. To append one file content to another file

cat file1.txt >> file2.txt

1. To copy content of multiple files to a single file

cat a.txt b.txt c.txt d.txt > total.txt (overwrite)

cat a.txt b.txt c.txt d.txt >> total.txt (appends)

cat is the word derived from from con “cat” enation

**tac command:**

it is the reverse of cat

it is vertical revese order

it will display file content in reverse order of lines

example :

cat file1.txt

sunny

bunny

chinny

vinny

tac file1.txt

vinny

chinny

bunny

sunny

**rev command:**

To show in horizontal reversaly

rev file.tx

Ynnus

ynnuB

cat command will display total file content at a time.

It is best suitable for small files

If the file contains thousands of lines it is not recommended to use cat command

For this requirement we should go for

head

tail

more

less

**head command:**

To view specified number of lines from top of the files

10 is default value for head

head abc.txt 🡪 it will display only 10 lines from top of the file

head -n 15 abc.txt or head -15 abc.txt

head -n 5 abc.txt or head 5 abc.txt 🡪 it will display only top 5 lines

head -n -5 abc.txt 🡪 It will display all lines from top except last 5 lines

head -c 100 abc.txt 🡪 It will display first 100 characters in the file

In Linux every character will take one byte. Hence it will display first 100 bytes of file content

**tail command:**

to view file content from bottom of the file 🡪 tail command

It is opposite to head command

tail abc.txt 🡪 It will display last 10 lines from the bottom

tail -n 5 abc.txt 🡪 last 5 lines

tail -5 abc.txt 🡪 last 5 lines

tail -n -5 abc.txt 🡪 last 5 lines

tail -c -100 abc.txt 🡪 It will display 100 bytes of characters from the bottom of the file

head -7 abc.txt | tail -5 🡪 from 3rd line to 7th line ( 7-3 +1 =5)

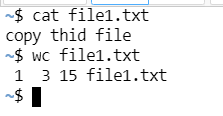
head -23 abc.txt | tail -9 🡪 From 15th line to 23rd line (23-15 +1=9)

**Word count command (wc command):**

We can use wc command to count number of lines words and characters present in the given file

$ wc filename

No\_of\_lines no\_of\_words no\_of\_character filename



1 🡪 no of lines

3 🡪 no of words

15 🡪 no of character

We can use the following options with wc command

-l 🡪 to print only number of lines

-w🡪to print only number of words

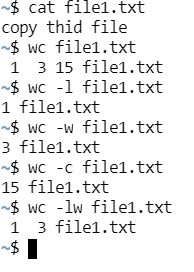
-c🡪 to print only number of characters

-lw🡪 to print only number of lines and words

-lc🡪 to print only number of lines and characters

-wc🡪to print only number of words and characters

-L 🡪 to print number of characters present in longest line



We can use wc for multiple files

$wc file1.txt file2.txt file3.txt

**Command Aliasing:**

Alias mean alternative name or nick name.

We can give our own more convenient nicknames for unix commands. This concept is called command aliasing.

**Syntax:**

$ alias nickname=’original command’

$ alias nickname=”original command”

**Example:**

$ alias cls=’clear’

$ alias cls=”clear”

How to remove alias name

**Syntax:**

$ unalias command name

**Example:**

$ unalias cls

To list all the alias

$ alias

To remove all the alias

$ alias -a

After giving alias name is original command will work yes.

Can we have multiple alias name for a single command: yes

I want to define alias name, but how to check is it available already?

Just use type command

$ type cls

cls is aliased to ‘clear’

where we can use aliasing?

* if any lengthy command repeatedly required then we can create shortcut alias name and we can use that shortcut every time.

mkdir dir1;touch dir1/file{1..20}.txt

$ alias d20f=’ mkdir dir1;touch dir1/file{1..20}.txt

Eg: To list out all files present in current working directory, save this data to output.txt and display the number of lines to the terminal. Define alias name ‘current’ for this total activity

$ alias current=”ls -l | tee output.txt | wc -l”

* To use our home operating system commands in linux directly widows or mac

$ alias cls=’clear’

$ alias rename=’mv’

* To handle typing mistakes

grep

grpe

$ alias grpe=’grep’

* To handle language barriers

In Germany 🡪 datum

$ alias datum=’datum’

$ alias koopy=’cp’

How to persist aliases permanently

We can make our created aliases permanently in our system by using 2 ways

1st way

.bashrc file inside user home directory

Text

Description automatically generated

To reflect these aliases, we have to restart terminal

2nd way

Instead of editing .bashrc file, we can create our own file to maintain our defined aliases.

The name of the file should be .bash\_aliases and should be present in user home directory

.bash\_aliases

### **Text Processing**

* Using GREP command,
* Using SED command,
* Using AWK command,
* Mounting a file to the virtual box,
* Creating a shared folder (mounting a folder),
* Using SORT command and
* Using pipes to combine multiple Commands.