GIT :-

Git is a version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As developers make changes to the projects. any earlier version of the project can be recovered at any time.

Developers can review project history & find out.

* which changes were made
* who made the changes
* which were the changes made
* why were changes needed

Repositories :-

A repository, or git project encompasses the entire collection of files & folders associated with a project along with each files revision history.

Basic git commands:

git init :- Initializes a brand new git repository and begins tracking an existing directory. It adds a hidden sub folder within

the existing directory that needs the internal data structures required for version control.

git clone :- Creates a local copy of a project that already exists remotely. The clone includes all the project's files, history and branches.

git add :- Stages a change. Git tracks changes to a developer's codebase. This command performs staging, the first part of that two step process.

git commit :- Saves the snapshot to the project history & completes the change-tracking process. In short, a commit functions like taking a photo. Anything that is been staged with git add will become a part of the snapshot with git commit.

git status :- Shows the status of changes as untracked, modified, or staged.

git branch :- Shows the branches being worked on locally.

Crit push :- updates the remote repository with any commits made locally to a branch.

Features of crit:

* Tracks history
* Free & open source
* Supports non-linear development
* Creates backup
* Scalabel
* Supports collaboration
* Branching is easier
* Distributed development

Crit Installation on linux:

1) $ sudo apt-get install git
2) To check if git is present
   $ where is git
   ↓
   git by default is installed under /usr/bin/git directory on most linux systems.
3) To get the version of the git
   $ git --version
   O/P → git version 1.9.1

4) To specify a user and password information to git repository, then use the following command.

$ git config --global user.email "mail"

for verifying git configuration

git config --list

O/P - user.email = Th Junaid007pasha@gmail.com.

Functions:

Function is a command in linux which is used to create functions or methods.

i) Using function keyword:

A function in linux can be declared by using keyword function before the name of the function

```
function name{
    commands;
}
```

Eg: function hello{
    echo hello, welcome to LSS
    }

>>> hello

Hello, welcome to LSS

① using parenthesis :

A function can also be declared by using parenthesis after the name of the function. Different statements can be separated by a semicolon or a new line.

name () { COMMANDS; }

Eg: Say_hello ()
{
echo Hello, welcome to CSS }

② Parameterised function :-

function add
{
a = $1
b = $2
add = $((a+b))
echo $ add
{
add 3 4
7

$1 will displays the first argument that will be sent and $2 will display the second argument so on.

Programming Constructs.

The if .. else .. fi

The if-else-fi statement is the next form of control statement that allows shell to execute statement in a controlled way and make the right choice.

Syntax:

```
if [expression]
 then
   Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
fi
```

The shell expression is evaluated in the above syntax. If the resulting value is true, given statement(s) are executed. If the expression is false, then no statement will be executed.

Eg:

```
a=10
b=20
if [$a == $b]
then
  echo "a is equal to b"
else
  echo "a is not equal to b"
fi
```

Nested if:

If specified condition is not true in if part then elseif part will be execute or else part will be executed. If none of the condition is true then it processes else part.

Syntax:

```
if [expression]
then
   Statement1
elsif [expression 2]
then
   Statement 2
   :
  else
   Statement n
  fi
```

if conditional constructor:

If condition checks for the expression. if the expression or condition is true it executes the Statement or else it will come out of loop.

```
if [expression]
then
   statement
  fi.
```

while condition :-

The while loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

Syntax:

```
while command
do
  Statement(s) to be executed if command is
    true
done.
```

Eg:

```
a=0
while [$a -lt 10]
do
  echo $a
  a='expr $a+1'
done
```

Continue condition:

Continue is a command which is used to skip the current iteration in for, while and until loop. It takes one more parameter [N]. if N is mentioned then it continues from the n^{th} enclosing loop.

```
Continue
or
continue [N]
```

Eg: for i in `seq 1 10`
>do
>if ((i == 5))
> then
> continue
> fi
> echo $i
> done
1
2
3
4
5
6
7
8
9
10

Variable Types:-

① Local variables:- A local variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.

② Environment variables:- An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually, a shell script defines only those environment variables that are needed by the programs that it runs.

③ Shell variables: A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

Process Management in linux:

A process means program in execution

1) Foreground process: Such kind of process are also known as interactive process are also known as. There are the processes which are to be executed or initiated by the user or the programmer They can not be executed by the system services.

2) Background process: Such kind of processes are known as non-interaction processes. There are the processes which are to be executed by users or the system itself. These processes have a unique PID.

Example for foreground processes:

Sleep 100

① This command will be executed in the terminal & we would be able to execute another command after the above command.

① To stop a process in between: To stop a process in b/w the execution, the user has to press ctrl+z to force stop it.

Sleep 100
^Z
[1] + stopped Sleep 100

② To get the list of jobs which are running or stopped

Syntax: jobs

[1] + stopped Sleep 100

③ To run all the pending & force stopped jobs in the background.

Syntax: bg

jobs
④ [1] + stopped Sleep 100

bg
⑤ [1] + Sleep 100 &

jobs
⑥ [1] + running Sleep 100 &

User Management

A user is an entity, that can manipulate files & perform several other operation. Each user is assigned a ID which is unique for each user in the operation system.

To list out all the users, use the awk command with -F :

awk -F: '{ print $1 }' /etc/passwd.

Here we are accessing a file and printing only first column with the help of $1 & awk

Using ID command

ID username

using "id username" we can identify any username

Every user has an id assigned to it.

output { Id test
uid = 1003 (test) gid = 1003 (test)
groups = 1004 (test)

The command to add an user

Sudo "useradd "Name"

output

Sudo useradd rohan
password for rohan : ****

using password to assign a password to the user

Sudo passwd Junaid
Password for Junaid
Enter new password
Re-enter new password

Group Creation

Syntax:

groupadd group_name

Output:

groupadd group1

Setting password to group

gpasswd (-group)

New password:

Re-enter password:

To add an user to a existing group

Syntax

usermod -G group_name username

Output:

usermod -G group1 Junaid

To delete a user from a group

Syntax:

gpasswd -d username groupname

Output:

gpasswd -d junaid group1

To delete the whole group

Syntax:

groupdel group_name

output:

groupdel group1

File Permission

Type ls to list out the files

-rw-r--r-- 1 Junaid junaid ...

r = read

w = write

x = execute

-rw- r- -r-

user group other

user = u

group = g

other = other

To give permission to a specific file

Junaid.txt

Syntax

chmod ugo+rwx Junaid.txt

The above command will give the user to give the permission to read write & execute the file

This is known as Symbolic method

Absolute Method

Octal	binary	File mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx

4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Example:

chmod 777 junaid.txt

The above command will give permission to read, write & execute.

chmod 435 junaid.txt.

4 = read only (user)
3 = write & execute (group)
5 = read & execute (others)

So the code gives read permission to the user, write & execute permission to group & read & execute to others.

PATTERN MATCHING, WILD CARD, META & CHARACTER CLASSES

Pattern Matching:

Pattern matching in the shell against filenames has metacharacters defined differently from the rest of the unix pattern matching program

* is match any character except white space, ? is match one character except white space

to *.c is match any filename ending with the two characters .c.

wildcard patterns

Given a text & a wildcard pattern implement wildcard pattern matching algorithm that finds if wildcard pattern is matched with text. The matching should cover the entire text The wildcard pattern can include the characters '?' & '*'

'?' - matches any single character

'*' - matches any sequence of characters

eg: Text = "baaabab"

Pattern = "***ba***ab", Output - True

pattern = "baaa ?ab", output - True

Pattern = "ba*a?", output - True

Pattern = "a*ab", output - False

Meta characters

The command option, option arguments & command arguments are separated by the space character. However, we can also use special character called metacharacter in linux command that the shell interprets rather than passing to the command

eg > - output redirection

< - Input redirection

* - file substitution wildcard, zero or more characters

? - file substitution wildcard, one character

| - pipe (|)

>> - output redirection to append

<< - input redirection

> Syntax

echo "Hello"

test.txt

>> Syntax

echo "filename" >> test.txt

This will append the line to the existing file

* Syntax

is mx

→ mod1 mod2

? Syntax

ls ? txt

eg 1.txt 11.txt

output 1.txt

FILTERS

Filters are program that take plain text [either sorted in a file or produced by another program] as standard input, transform it into a meaningful format, and then returns it as standard output.

1. cat : Displays the text of the file line by line

Syntax : cat [path]

2. Head : Displays the first n lines of the specified text files If the numbers of lines is not specified then by default prints first 10 lines.

Syntax : head [-number_of_lines_to_print] [path]

3. Tail : It works the same way as head, just in reverse order. The only difference in tail is, it returns the lines from bottom to up.

Syntax : tail [-number_of_lines_to_print] [path]

4. wc : wc command gives the number of lines, words and character in the data.

Syntax : wc [-options] [path]

This gives 4 outputs as

→ Number of lines
→ Number of words
→ Number of characters
→ path

5. Grep : Grep is used to search a particular information from a text file.

Syntax : grep [option] pattern [path]

6. nl : nl is used to number the lines of our text data

Syntax : nl [-option] [path]

7. Sed : Sed stands for stream editor. It allows us to apply search & replace operations on our data effectively sed is quite an advanced filter & all its options can be seen on its man page

Syntax : sed [path]

Regular Expression:

In linux regular expression are special characters which helps to search data & matching complex pattern. Regular expressions are called as 'regrep & regx'. Reg exps are most commonly used within the linux commands.

grep, sed, br, vi

Types of regular Expressions

Basic Regular functions

(.) Replaces any character
(^) Matches start of string
(+) Matches end of string
(*) Matches 0 or zero or more times of preceding character
(\) Represents special character
() Groups regular expression
(?) Matches up exactly one character

These are used with commands like br, sed, vi & grep commonly

Internal regular expression

This expression tells us about the number of occurences of a character or string.

Expression :{n} Matches the preceding character appearing n times

{n,m} matches the preceding character appearing n times but not more than 'm'

{n} matches the preceding characters only when it appears 'n' times or more

External regular expression

This expression contains continuous of more than one expression

Expression:

\+ Matches one or more occurance of the previous character

\? Matches zero or more occurance of the previous character

Eg. Give a file 'Sample' containing texts

apple
ball
bat
cat
people

i) Search for content containing letter 'a'

ii) Search for content start with 'a'

iii) Search for content that end with 'b'

iv) Search for content when character 'a' precedes 'b'

i) $ cat sample | grep a

apple
ball
bat
Cat
people

ii) $ cat sample | grep ^a

apple

iii) $ cat sample | grep t $

bat
cat

iv) cat sample | grep -"a \+t"

bat
cat