GIT:GIT is distributed version control system.

To create the configuration:

git config **-->**will give list of variables

git config --global --list **-->**gives the list of existing login credentials

git config --global user.name "veena" **-->**set the user name.if the same command is given,overwrites the existing credentials

git config --global user.email "veenanjalitammina999@gmail.com" **-->**to set the email

git config --global --get user.name **->**give the user name

git config --global --get user.email **->**give the email

git init **-->**Transform the current directory into a Git repository. This adds a .git subdirectory to the current directory .By default it creates in master branch.

git init <directory> **-->**create a new subdirectory called <directory>containing nothing but the .git subdirectory.

git init <directory> --template=<template_directory> **-->** --template option allows you to specify a directory to use as a template when initializing a new Git repository.

git status **-->**will give the status of the repository

git init -b main **-->**creates local repository in main branch.(creates a hidden .git folder where staging area and commits will be present there)(b->branch)

#To commit the changes, the file should be in staging area,for staging "**add"** is used

git add file1.txt **-->**will add the file to staging environment but not committed

git log **-->**will show all the commits

git commit -m "message that specify what features are added"

#for each commit, checksum(14 digit hexadecimal value,only first seven digits will be displayed) will be given**.**

#Everytime the changes are made,the file should commit.To commit the file directly by skipping the staging,use -a

git commit -a -m "file moves from working directory to commit without moving to staging area"
-a option doesn't require a separate git add step before committing**.**

git diff **#**give the changes that are made before adding the file to staging area.If the file is added using 'add' then diff won't show wany changes.

git diff -staged #give the changes that are made in staging area

git add . #add all the files to the staging area

git rm --cached file.txt #remove the file from git

git commit -m "removed the file"

git rev-list --count HEAD #to get the count of commits

git rev-list --count <branch-name> #for each branch

git branch -M main #creating a main branch

#settingup ssh

ssh-keygen -o #generates the key and asks the filename to save the key.default stores in id_rsa.pub(stores public key),id_rsa(stores privaqte key) files in .ssh folder.

git remote add origin sshurlofrepo #here instead of ssh url https url can also be given,but for https url it asks for the authentication.It is used to add a remote repository to a local Git repository. In this case, origin is typically used as the name of the remote, and sshurlofrepo is the SSH URL of the remote Git repository.

git push -u origin main #push the repository from local machine to remote repository

#after pushing into remote repository,if any changes made to local macine repository,then the repository need to push again to remote repository to commit .'-u' option maintains the link between local and remote repositories.

git remote -v **#checking existing remotes**

git remote remove origin #remove the origin remote

git tag v1.0 -m "release1" #adds the tag to local repository

git tag #shows the list of tags

git push origin v1.0 #adds the tag to remote repository

git show v1.0 #shows info about version v1.0

git log --pretty=oneline #give the short logs of commits

git switch -c feature1 **#creating new branch named feature1,no limit in creating branches.**

git switch main **#switch to main branch**

git branch #gives the list of branches and highlight the active branch

git branch -all #gives the list of branches in local machine along with remote repository

git switch - #changes to previous branch

#assume that you are in the main branch,and repoitory has feature branch,now to merge feature branch into main ,you should be in main branch and execute the following command.Before pushing, pull the main branch from main repository and do merging.After merging push it back to the remote repository

git merge feature #merging feature branch in main

git show:file1.txt #shows the contents of previous version

git show <commit_hash>:file1.txt #shows the content of specified commit hash

git clone <repo url> #create a copy of remote repositories

counting commits:

to count all commits in the current branch (assuming you are on the master branch),

git rev-list --count HEAD #to count all the commits in the current branch

git rev-list --count HEAD -- <file_path> #Count commits for a specific file:

git rev-list --count <branch_name> #Count commits for a specific branch:

git config levels :

--local(Repository/Project level)

By default, git config will write to a local level if no configuration option is passed.Local configuration values are stored in a file that can be found in the repo's .git directory: .git/config

--global(User level)

Global level configuration is user-specific, meaning it is applied to an operating system user. Global configuration values are stored in a file that is located in a user's home directory. ~ /.gitconfig on unix systems and C:\Users\\.gitconfig on windows

--system(Git installation)

System-level configuration is applied across an entire machine. This covers all users on an operating system and all repos. The system level configuration file lives in a gitconfig file off the system root path.

git config --list --show-origin #check complete git config

git config --global --unset user.name #removes the user name only

git config --local --remove-section user #removes the complete section

How git stores the data?

Git stores data as objects . There are three main types of objects in Git: blobs, trees, and commit

A **blob** represents the content of a file at a specific state. Blobs store the actual file data, and they are uniquely identified by a SHA-1 hash of their contents.

A **tree object** represents a directory and its contents. It points to blobs (file contents) and other tree objects (subdirectories). The tree object is also identified by a SHA-1 hash

A **commit object** represents a snapshot of the entire project at a specific point in time.Commits are also identified by a SHA-1 hash.(SHA-1 is 20 bytes in hexadecimal format)

To decode the hash,use

git cat-file sha-1 -p

Renaming file in git:

git mv file1.txt file.txt   #renames file1.txt to file.txt

git restore

Removing the files in staging area:

git restore --staged file.txt   #moves the file from staging area to unstaged area

This command restores the specified file from the last commit to the staging area. It undoes changes that were previously added to the staging area but not committed.

git restore   --worktree file.txt

This command restores the specified file from the last commit to your working directory. It discards changes in both the staging area and the working directory.

git restore file.txt   **#to discard** the changes that are made in unstaging area

git restore --source=<commit> <file>   #restore the file to   state it was in the given commit

git checkout

To switch to an existing branch:

git checkout <branch_name>

To create and switch to a new branch:

git checkout -b <new_branch>

To discard changes in a specific file:

git checkout -- <file>

To discard all changes in the working directory (use with caution, as it's not reversible). used to discard changes in tracked files, and it won't remove untracked files in your working directory.

git clean -fdX

To remove untracked files, you can use the git clean command. The -d option tells Git to also remove untracked directories, and the -X option ensures that only untracked files are removed, not untracked directories

git clean -fdXn

This will show you a preview of the files that would be deleted without actually deleting them.

git checkout -- .

Checking Out a Specific Commit (Detached HEAD state):

To view a specific commit without creating a branch:

git checkout <commit_hash>

Checking Out a Specific File from a Different Commit:

To replace a file in the working directory with the version from a different commit

git checkout <commit_hash> -- <file>

Switching Back to the Previous Branch:

If you switch to a commit directly and want to return to the previous branch:

git checkout -

Switching to a Tag:

To switch to a specific tag:

git checkout tags/<tag_name>

Creating a branch:

creating a branch and switch to new branch in one command

git checkout -b <branchname>

git switch -c <branch_name>

These two command creates a new branch (<branch_name>) and switches your working directory to that branch. If the branch already exists, it will switch to that branch. If it doesn't exist, Git will create it for you.

creating a branch

git branch <branch_name>

This command creates a new branch named <branch_name> but doesn't switch to it.

Using git clone with git checkout -b (from a remote repository):

git clone <repository_url> -b <branch_name>

If you are cloning a repository and want to create and switch to a new branch right away

creating a branch upto a specific commit hash

git checkout -b <branch_name> <commit_hash>

this command create a   new branch upto the specified commit hash

create a new local branch that tracks a remote branch.

git branch --track <local_branch_name> origin/<remote_branch_name>

creates a new local branch that tracks a remote branch

creating a branch which doesn't have any commit history

git checkout --orphan <branch_name>

This creates a new branch named <branch_name> without any commit history.But it will have contents

git rm --cached <file>

used to remove a file from the Git index (staging area) without removing it from the working directory

git rm -f <file>

This command forcibly removes the specified file from the Git index and the working directory, even if it has been modified.

git rm -rf .

This command removes only tracked files and directories from the working directory.

git commit --allow-empty -m "Initial commit"

The --allow-empty option in Git is used when you want to create a commit even if there are no changes to the files in the working directory.

Renaming a branch:

Move to the branch that to you want to rename through git checkout command git checkout branch1

git branch -m new_branch   #renames to new_branch

Deleting a branch

git branch -d delete_branch   #will generate warning if you commit changes to the branch

git branch -D delete_branch   #will delete the branch forcefully

HEAD pointer:

HEAD points to the latest commit on the currently active branch.

When you switch branches in Git, the HEAD pointer moves to point to the latest commit of the branch you just switched to. If you were on branch1 and made a commit, and then you switched to the main branch, HEAD will now point to the latest commit on the main branch.

configuring editor for git:

git config --global core.editor <editor-name>

git config --global core.editor vim   #setting vim as editor

git config --get core.editor   #to get the current configured editor

git config --global --unset core.editor   #unset the currently configured editor

Merging up branches:

In Git, there are two common types of merges: fast-forward merges and recursive merges.

Fast-Forward Merges:

git merge <source-branch>   #to execute this you should be in the taget branch

If there are no conflicts, Git will automatically perform the merge. If there are conflicts, Git will mark the conflicted files, and you'll need to resolve the conflicts manually.

git merge --abort   #aborts the merge process

When you perform a merge in Git, the branch being merged is typically not automatically deleted. Git does not automatically delete branches after a merge

# Delete the merged branch locally

git branch -d <branch-name>

If you have already pushed the merged branch to a remote repository and you want to delete it there:

 # Delete the merged branch on the remote repository

git push origin --delete <branch-name>

#FAST FORWARD MERGING-Here HEAD pointer will move to the latest commit.no new commit will be created.

```
vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git branch
* master

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git switch -c branch1
Switched to a new branch 'branch1'

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ ls
file1.txt  file2.txt  file3.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ vi file4.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git add .
warning: in the working copy of 'file4.txt', LF will be replaced by CRLF
the next time Git touches it

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git commit -m "commit done"
[branch1 3fba440] commit done
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
```

```
vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git status
On branch branch1
nothing to commit, working tree clean

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git switch master
Switched to branch 'master'

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git merge branch1
Updating b1f8b4d..3fba440
Fast-forward
 file4.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
```

Recursive merge-new commit will be created after merging 2 branches.

```
vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git branch
  branch1
* master

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ vi file7.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git add .
```

```
vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git commit -m "commit1"
[master 9ee9db8] commit1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file7.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git switch branch1
Switched to branch 'branch1'

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ vi file11.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git status
On branch branch1
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file11.txt

nothing added to commit but untracked files present (use "git add" to track)

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git add file11.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git commit -m "commit2"
[branch1 97a4420] commit2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file11.txt
```

```
vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git status
On branch branch1
nothing to commit, working tree clean

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (branch1)
$ git switch master
Switched to branch 'master'

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git merge branch1
hint: Waiting for your editor to close the file... unix2dos: converting
file C:/Users/VLAB/Desktop/git_project/.git/MERGE_MSG to DOS format...
dos2unix: converting file
C:/Users/VLAB/Desktop/git_project/.git/MERGE_MSG to Unix format...
Merge made by the 'ort' strategy.
 file11.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file11.txt

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git status
On branch master
nothing to commit, working tree clean

vlab@VEDA-2F101 MINGW64 ~/Desktop/git_project (master)
$ git log
commit ce90a783063e2ec6c20e3578f9b05d6564e36923 (HEAD -> master)
Merge: 9ee9db8 97a4420
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 15:07:25 2024 +0530
```

```
    recursive merge

commit 97a4420bdb3206a8a726f2e22218c2d075840824 (branch1)
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 15:06:20 2024 +0530

    commit2

commit 9ee9db8f718f2ff60bfa2b3266d9ba8d4adb09bc
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 15:00:20 2024 +0530

    commit1

commit 533e29b1e630540765623b8d0b016742913a5438
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 14:58:23 2024 +0530

    file 6 added

commit b21f8c7ecfff63826ad09d514b0ddb7392c7a3f9
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 14:54:04 2024 +0530

    file5 added

commit 3fba440059fa5595936136dbf7d49d78332f8ebb
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 14:48:25 2024 +0530

    commit done
commit b1f8b4d99f62c7e09c330f96047713b26c171f08
Author: veena <veenanjalitammina999@gmail.com>
Date:   Fri Mar 1 12:03:08 2024 +0530
```

Colored outputs

git config command is used to set these color values.

color.ui **-->**mastervariable to set git colors.setting it to false disable all Git's colored terminal output.

By default, color.ui is set to auto which will apply colors to the immediate terminal output stream. The auto setting will omit color code output if the output stream is redirected to a file or piped to another process.

You can set the color.ui value to always which will also apply color code output when redirecting the output stream to files or pipes

git reset HEAD --

This command unstages all changes that have been added to the index (staging area). The double hyphen (--) is used to indicate the end of options and is typically not necessary in this context, but it's a good practice to include it to avoid ambiguity.

git cherrypick:

This will open the editor without modifying the commit message.

Cherrypick is used for the bug fixes where you want to place that bugfix commit in all the version branches.

Also used when we accidentally made a commit in wrong branch

git reset:

used to move the files from staging area to working directory moves the head to specific commit

git reset 57c6fe3 #MOVES THE HEAD TO SPECIFIED COMMIT WHATEVER THE CHANGES AFTER THE COMMIT WILL MOVE TO THE WORKING area

git reset 57c6fe3 --hard #moves the head to specified commit and whatever the changes after specificed commit will not even seen in working area

git reset 57c6fe3 --soft #moves the head to specified commit and whatever the changes after specificed commit will move to staging area

git reset HEAD --mixed #moves the stages changes to unstaging area --mixed is defaullt option

git reset ==== git reset --mixed HEAD

git reset --soft HEAD^

git stash:

if you switch branch without commiting then:

-> switches to the branch carrying changes

-> git won't allow to switch without commit or stash the changes

git stash

git stash show

display the changes made in the most recent stash

git stash show -v

git stash show -p

git stash show stash@{2} #Show Changes for a Specific Stash

git stash list

git stash pop #recentstash will be applied and removed from stash list

git stash save "stash name"

git stash drop stash@{1}

Creating alias

Aliases can be created through two primary methods:

1)Directly editing Git config files:The global or local config files can be manually edited and saved to create aliases. The global config file lives at $HOME/.gitconfig file path. The local path lives within an active git repository at /.git/config

2)Using the git config to create aliases

git config --global alias.<alias-name> '<git-command>'

Ex:git config --global alias.co checkout #alias created for checkout command

git config --global alias.unstage 'reset HEAD --' #alias createdfor unstaging

git checkout

If you have modifications in your working directory that are not yet staged (unstaged changes) and you want to discard them, you can use the git checkout command to revert the changes.

git checkout to remove changes from a modified file in your working directory. However, git checkout is generally used to switch branches, and using it to remove changes requires a specific syntax**.**

git checkout -- file.txt

Replace file.txt with the name of the file for which you want to discard the unstaged changes.

#To remove all unstaged changes in your working directory (for all files), you can use:

git checkout -- .

The . indicates the current directory, and this command will revert all unstaged changes.

# Unstaged changes

git reset HEAD -- file.txt

# Discard unstaged changes in working directory

git checkout -- file.txt

git add . #add all the files in current dirctory to the staging area

git commit -ammend

The git commit --amend command is used to modify the most recent commit in Git. When you run this command, it allows you to make changes to the previous commit message.

git commit --amend

This opens your default text editor (such as Vim, Nano, or Notepad) with the commit message of the last commit.

git commit --amend --no-edit

This will open the editor without modifying the commit message.

git diff

The git diff HEAD ./path/to/file command is used to view the changes made to a specific file in the working directory compared to the most recent commit (HEAD)

git diff: Initiates the diff command in Git.

HEAD: Refers to the most recent commit on the current branch. It's the commit that HEAD is currently pointing to.

./path/to/file: Specifies the path to the specific file you want to examine.

git diff --staged == git diff --cached

This command is an alternative way of expressing the same concept as git diff --cached. It also shows the changes that are staged but not yet committed.

git diff without a file path will compare changes across the entire repository.By default git diff will show you any uncommitted changes since the last commit.

#Comparing 2 branches

git diff branch1 branch2

git stash:

The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy. For example:

git stash apply

If conflicts occur, Git will mark the conflicted areas in the affected files.Resolving conflicts involves manually editing the files to merge the changes. Here's a step-by-step guide on how to resolve conflicts after applying a stash:

git add <conflicted_file>   #add the conflicted file

git stash drop   #complete the process by removing the stash

If you created a new file after stashing changes and then dropped the stash, the changes from the stash will not be applied, but the new file you created will remain in your working directory.

git stash apply

touch new_file.txt

git stash drop

After these steps, the changes from the stash, including modifications to existing files, will not be reapplied to your working directory. However, the new file (new_file.txt) that you created after stashing will remain in your working directory.

what is ssh key?

In Git, SSH (Secure Shell) is a widely used protocol for securely connecting and authenticating between a Git client (such as your local machine) and a Git server (usually a remote repository host).

Generate SSH Key Pair: First, you need to generate an SSH key pair on your local machine. each ssh key includes a public key and a private key. The private key should be kept secure on your machine, and the public key is added to the Git server for authentication.

ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

ssh-keygen: This is the command-line utility for generating SSH key pairs.

-t rsa: Specifies the type of key to create. In this case, it's RSA (Rivest–Shamir–Adleman), which is a widely used asymmetric encryption algorithm.

-b 4096: Specifies the number of bits in the key. A higher number of bits generally increases the security of the key. In this example, a 4096-bit key is generated.

-C "your_email@example.com": This option adds a comment or label to the key. The comment is often used to identify the key, and it's typically set to the user's email address. While the comment is optional, adding it can help you identify the key later.

Add Public Key to Git Server:

Copy the content of your public key (usually ~/.ssh/id_rsa.pub) and add it to your Git server's SSH keys

Pushing changes to remote repository:

git push <remote> <branch>

<remote> is the name of the remote repository.

<branch> is the branch you want to push.

to push changes to a specific branch in a remote repository

git push <remote> <local-branch>:<remote-branch>

<remote> is the name of the remote repository.

<local-branch> is the name of the local branch you want to push.

<remote-branch> is the name of the branch on the remote repository where you want to push the changes.

to push your local branch to the remote repository, specifying a new branch name on the remote repository.

git push origin <local-branch>:refs/heads/<new-remote-branch>

refs: This stands for references. In Git, references are pointers or labels associated with a specific commit.

heads: This indicates that the references are pointing to the latest commit on branches.

To see the list of branches in remote repository

git ls-remote --heads <remote-name>

when a repository is cloned from git,only the default branch(main) will be cloned to local machine.

to see the branches in remote repository,use

git branch -r

to add other branches from remote repo to local machine repo,

git switch branch1

now branch1 from remote repo will be added to the local machine repo.git checks whether there is a branch named branch1 in local repo,if it is not present,it checks in the remote repo.If the branch1 is present in remote repo,it creates a new branch named branch1 in local repo and then sets up track with the branch1 in remote repo

git fetch:fetches latest info from github ,but don't add it into working directory

```
git fetch <remote>              #fetch changes from specified remote repository

git fetch <remote><branch>      #fetch changes from specified branch in remote repository
```

git fetch doesn't automatically modify your working directory; it updates the remote tracking branches in your local repository.

to fetch changes and automatically merge them into your current branch, you can use **git pull**. The git pull command is essentially a combination of **git fetch followed by git merge**: