Git

&

GitHub

Git

Git is a free and open source distributed version control system software.

it is used to track changes to your files/source code.

It helps to revert to previous versions of a file/code.

GitHub

GitHub is a web-based hosting service for Git repositories.

it is a an online platform to store and share our code with others.

Before Git there was a version control system called SCCS (Source Code Control System).

It was proprietary software that was used to manage history of code. It was very expensive and not very user-friendly.

Some common version control systems are Subversion (SVN), CVS, Perforce, Mercurial.

Git flow

Git init

Git add

Git commit

Git push

Repository

It is a collection of files and directories that are stored together.

Staging area

It is an intermediate space where the changes are collected and it allows us to group, organize and review the files before they are committed to the repository.

Indexing

it is a process of adding the files to the staging area

Set username and email globally

git config –global user.name “Your Name”

git --version

This command displays the version of the git installed on our system.

git status

This command is used to display the current state of the working directory / repository.

git init

This command initializes the project directory as a git repository.

It converts the current directory into git repository. This directory is now tracked by Git

This command adds a hidden .git folder to our project.

.git folder

This folder contains all the metadata that git uses to manage the repository.

Git directory is the database where metadata about project files' history will be tracked.

git add

This command adds the files to the staging area. Stage is a way to tell git to track a particular file or folder.

-> git add .

To add all files present in the directory to the staging area.

-> git add -all or -A

To add all files present in the directory to the staging area.

-> git add <file1> <file2>

To add specific files to the staging area

Git commit -m "message"

This command is used to commit files to the local git repository with a message.

It records the changes made to the repository.

Each commit represents a snapshot of the project at a specific point in time, allowing you to track progress and revert to previous states if necessary.

The information related to this commit (names of files committed, date and time of commit, author of commit, commit message) is stored in the Git directory.

git log

This command shows the history of our repository.

It will show all the commits that were made to the repository.

-> git log --oneline

This command shows only the commit message

Atomic Commits

It is way to make sure that each commit is a self-contained unit of work.

The commit should be able to stand on its own, without depending on or affecting other commits.

It should also have a clear and descriptive message that summarizes the changes made.

An atomic commit is the smallest, most meaningful change you can make to the software.

Commit message

Present tense, Imperative - Depends

Add Footer to code base or add footer

Default code editor

In Git the default code editor is vim

If we commit without a message, the git will open the vim editor to enter a message for the commit.

We need t add message and exit the vim editor using esc key and :wq

.gitignore file

Gitignore is a file that tells git which files and folders to ignore.

It is a way to prevent git from tracking certain files or folders.

You can create a gitignore file and add list of files and folders to ignore.

.gitkeep file

Git does not track empty folders.

If we want to keep track of empty folders we can create a .gitkeep or any other file in that empty folder.

Branches

Branches allows us to work on different versions of a project at the same time.

They allow you to create a separate line of development that can be worked on independently of the main branch.

This can be useful when you want to make changes to a project without affecting the main branch or when you want to work on a new feature or bug fix.

git branch or git branch --all

To display all branches both local and remote .

git branch <branch name>

To create a new branch.

git switch <branch name>

To switch to a different branch.

git switch -c <branch name>

This command creates a new barnch if it is not already existed and switches to that branch. If the branch with the same name is already present it just switches.

If we create a new file in one branch it will not appear in another branch.

git checkout <branch name>

It is used for switching branches, checking out specific commits, and restoring files. However, be cautious as it can lead to data loss if misused.

git checkout is used for both switching branches and restoring file states, whereas git switch is specifically tailored for switching branches.

Git Graph extension n VS code

git merge <branch name>

To merge ne branch into another one.

Fast-forward merge

This one is easy as branch that you are trying to merge is usually ahead and there are no conflicts.

When you are done working on a branch, you can merge it back into the main branch.

Not fast-forward merge

In this type of merge, the master branch also worked and have some commits that are not in the bug-fix branch. This is a not fast-forward merge.

Managing conflicts between branches

We have to manually resolve the conflicts. We need to decide what to keep and what to discard.

VSCode has a built-in merge tool that can help you resolve the conflicts.

Github also has a merge tool that can help you resolve the conflicts.

First we need to merge the branch and then edit the file and keep the content we need and then add the modified file and commit the changes.

Then the conflict will be resolved.

git branch -m <old-branch-name> <new-branch-name> or git branch -m <new-branch-name>

To rename a branch

git branch -d <branch-name>

To delete a branch

Git diff

The git diff command is a powerful tool in Git that allows you to compare changes between various states of your repository.

The git diff is an informative command that shows the differences between two commits or two branches.

It is used to compare the changes made in one commit with the changes made in another commit.

Git consider the changed versions of same file as two different files. Then it gives names to these two files and shows the differences between them.

How to read the diff

a -> file A and b -> file B

---- indicates the file A

+++ indicates the file B

@@ indicates the line number

Here the file A and file B are the same file but different versions.

Git will show you the changes made in the file A and file B. It will also show you the line number where the change occurred along with little preview of the change.

git diff --staged

This command shows the changes between your last commit and the staging area (i.e., changes that are staged and ready to be committed).

git diff <branch-name-one> <branch-name-two> or git diff branch-name-one..branch-name-two

This command compares the difference between two branches.

git diff <commit-hash-one> <commit-hash-two>

This command compares the difference between two commits.

Git Stash

We cannot switch the branches without commiting the changes.

We can use git stash command to save the changes in a temporary location.

Stash is a way to save your changes in a temporary location.

It is useful when you want to make changes to a file but don’t want to commit them yet.

You can then come back to the file later and apply the changes.

git stash

Naming the stash

git stash save "work in progress on X feature"

View the stash list

git stash list

Apply the stash

git stash apply

apply the specific stash

git stash apply stash@{0}

drop the stash

git stash drop

apply and drop the stash

git stash pop

apply the stash to a specific branch

git stash apply stash@{0} <branch-name>

Clear the stash

git stash clear

Git Tags

git tag -a <tagname> -m "message"

This command creates a new tag with the specified name. The tag will be attached to the current commit.

git tag -a <tag-name> -m "Release 1.0"

This command creates an annotated tag with the specified name and message. The tag will be attached to the current commit.

git tag

This command lists all the tags in your repository.

git tag <tag-name> <commit-hash>

You can tag a specific commit

git tag -d <tag-name>

You can delete a tag

git push origin <tag-name>

push tags to a remote repository

git push origin :<tag-name>

delete a tag on a remote repository

Git rebase

Git rebase is a powerful Git feature used to change the base of a branch.

It effectively allows you to move a branch to a new starting point, usually a different commit, by “replaying” the commits from the original base onto the new base.

This can be useful for keeping a cleaner, linear project history.

Some people like to use rebase over the merge command because it allows you to keep the commit history cleaner and easier to understand.

It also allows you to make changes to the code without affecting the original branch.

work on two branches

checkout to branch other than master

git rebase <branch-name>

To realign the base of a branch

Git reflog is a command that shows you the history of your commits.

It allows you to see the changes that you have made to your repository over time. This can be useful for debugging and understanding the history of your project.

git reflog

This will show you the history of your commits. You can use the number at the end of each line to access the commit that you want to view.

git reflog <commit-hash>

find a specific commit

Recover lost commits or changes

git reset --hard <commit-hash>

give the commit hash on which you want to go back

git reflog <commit-hash>

git reset --hard HEAD@{1}

you can use HEAD@{n} to reset to the nth commit before the one you want to reset to.

GitHub

ssh-keygen -t rsa -b 4096 -C "moghalakrambaig8@gmail.com"

check the remote url setting

git remote -v

add a remote repository

git remote add origin <remote-url>

Push code to remote repository

git push origin main

Setup an upstream remote

Setting up an upstream remote is useful when you want to keep your local repository up to date with the remote repository.

It allows you to fetch and merge changes from the remote repository into your local repository.

git remote add upstream <remote-url> or git remote add -u <remote-url>

This will set up an upstream remote and push your code to the remote repository.

git push -u origin main

Git treats everything as an object and Everything is stored as an object and each object is identified by a unique hash code.

Every commit has its own hash value

The commit objects are stored in .git folder.

Git Snapshots

A git snapshot is a point in time in the history of your code.

It represents a specific version of your code, including all the files and folders that were present at that time.

Each snapshot is identified by a unique hash code, which is a string of characters that represents the contents of the snapshot.

Three pillars that git uses to stores objects

Commit Object

Tree Object

Blob Object

Commit Object

Each commit in the project is stored in .git folder in the form of a commit object. A commit object contains the following information:

Tree Object

Parent Commit Object

Author

Committer

Commit Message

Tree Object

Tree Object is a container for all the files and folders in the project. It contains the following information:

File Mode

File Name

File Hash

Parent Tree Object

Everything is stored as key-value pairs in the tree object. The key is the file name and the value is the file hash.

Blob Object

Blob Object is present in the tree object and contains the actual file content. This is the place where the file content is stored.

git show -s --pretty=raw <commit-hash>

To display the Commit Object info

git ls-tree <tree-id>

To display the Tree Object info

git show <blob-id>

To get the Blob Object.

git ls-tree 06cac5498b07be76623c255af5a0c053883cc2c5

git show 9fa2b4a81494400a6a5007173704806c87a6ca34

git restore

To unstage the staged files and to restore uncommited changes