Git and GitHub

GIT - VCS - version control system - to track changes in files / folders - to collaborate in teams - free and open source

Centralised VCS | Distributed VCS

GIT = DVCS

GIT HUB - website to upload your repositories online - provides backup - provides visual interface to your repo - makes collaboration easier.

GIT **!=** GIT HUB

Step 1 : Check if git is already installed.

terminal **- git --version**

Step 2 : Download and install git https://git-scm.com/download/mac

Step 3 : Signup and create a account on GitHub https://github.com/

Step 4 : Add your github email and username to git

git config --global user.email "yourGitHub@email.com"

git config --global user.name "yourGitHubusername"

Step 5 : Add file/folders to git - tracking

Step 6 : Commands - terminal - goto the location of the folder/project –

- git init.
- git status.
- git add.
- git commit -m “Any_message”.
- git remote add origin “location of remote repo”.
- git push -u origin master.
- git log - git –help.

Git Branches

Branch is a pointer to a specific commit in a repository's history. When a new branch is created, it points to the current commit at that time. Any new commits made while on that branch will move the pointer forward, indicating the new tip of the branch.

Git branches are a powerful tool for managing changes in a repository, enabling parallel development, experimentation, and easy merging of changes.

Step 1 : Create branch

git branch “branch name”

Step 2 : Checkout branch

git checkout “branch name”

Step 3 : Merge new branch in master branch

git merge “branch name”

Step 4 : Delete branch

git branch -d “branch name” — delete from local.

git push origin —delete “branch name” — delete from remote.

How to trigger notification email from github

Whenever there is any change/commit in the project

Step 1: Github - Repository - Settings - integration & services - add email.

Step 2: Test and validate by making some change in the project.

Git Tags

Step 1: Checkout the branch where you want to create the tag

git checkout "branch name" example : git checkout master

Step 2: Create tag with some name

git tag "tag name" example : git tag v1.0

git tag -a v1.0 -m "ver 1 of .." (to create annotated tags)

Step 3: Display or Show tags:

git tag

git show v1.0 git tag -l “v1.*”

Step 4: Push tags to remote

git push origin v1.0 git push origin –tags

git push --tags (to push all tags at once)

Step 5: Delete tags (if required only) to delete tags **from local** : git tag -d v1.0 git tag --delete v1.0

to delete tags **from remote** : git push origin -d v1.0 git push origin --delete v1.0

git push origin :v1.0 to delete multiple tags at once:

git tag -d v1.0 v1.1 (local)

git push origin -d v1.0 v1.1 (remote

Checking out TAGS

We `cannot checkout tags in git We` can create a branch from a tag and checkout the branch

git checkout -b "branch name" "tag name" example : git checkout -b ReleaseVer1 v1.0

Creating TAGS from past commits

git tag "tag name" "reference of commit"

example : git tag v1.2 5fcdb03

Git Merge and Git Rebase

In Git there are 2 ways to integrate changes from one branch to another

git merge git rebase

git merge - Is a non-destructive operation - Existing branches are not changed in any way - Creates a new merge commit in the feature branch

git rebase - Moves the entire feature branch to begin on the tip of the master branch - Re-writes the project history - We get much cleaner and linear project history

Merging preserves the entire history of a branch and creates a new merge commit, while rebasing creates a linear history and replays the changes of the rebased branch onto the target branch. The choice between merging and rebasing depends on the specific needs of the project and the desired outcome of the integration.

git checkout master

git merge feature

```
A --- B --- C --- D  (master)
          \
           E --- F --- G  (feature)
```

git checkout feature

git rebase master

```
A --- B --- C --- D  (master)
              \
               E' --- F' --- G'  (feature)
```

Git Stashing

In Git, stashing is a way to save changes that you have made to a working directory when you're not yet ready to commit them. Stashing allows you to temporarily remove the changes from your working directory, so that you can switch to another branch or work on a different feature without worrying about losing the changes you've made.

`git stash save`

`git checkout original-branch`

`git stash apply`

`git stash list`

`git stash drop` to remove a stash.