Git and Github

Page No.:
Date: / /

- Git is a "distributed version control system."
- "Distributed" → When we pull the files from the server to our device, we will get all the files with history.

(along)

Backup is available at device also if server lost data and if device lost data then also server have the data/files.

- Brings a local copy of the complete repo to every team member, so they can commit, branch and merge locally.
  Server does not have to store a physical file for each branch - it just needs "differences b/w each commit."

- Advantages: (a) easily recover files
  (b) who introduced issue and when
  (c) roll back to previously working state.

[Note]:-

"Linus Torwalds" made linux kernel as well as Git.
The need comes when the development of linux goes large: 1991-2002: dev.
2002-2005: Bitkeeper VC S
2005 → Bitkeeper removed free access.
then came git.

Page No.:
Date: / /

Git and Github :-

- Git is a VCS system that allows developers to track changes in their code. Github is a web based hosting service for git repositories.

- You can use git without github but not reverse.

- Git → maintained by linux
  └ installed locally

  Github → maintained by Microsoft
  └ hosted in the cloud.

- Other hosting web service : Bitbucket, Gitlab, Github ⇒ all host git repo.

Features :-

(1) Git save snapshot and not differences.
Ex: Whenever we create a git repo, then ".git" folder will store store all the history of commits.
When some file will be changed, then it will take its snapshot and not the differnces to store.

(2) Almost all of. it local.
(3) Git has integrity.
(4) Git generally adds data only.

⇒ We can use powershell or bash, after installing git.

↑ [Application for a windows environment which provides an emulation layer for git command line experience.]

checksum:-

- is a code that git maintains with every file.
- If a file is downloaded, then checksum will go along with that.
  If there is some change done by any hacker, then checksum will change, and user will become known that some thing wrong happen.

- "pwd" → will show present-working-directory.
- cd → means change directory.
- If we do: cd /c ⇒ then we go to "C" drive of our computer device (directory)

- git config --global user.name "Ankit"

- git config --global user.email "ankit.."

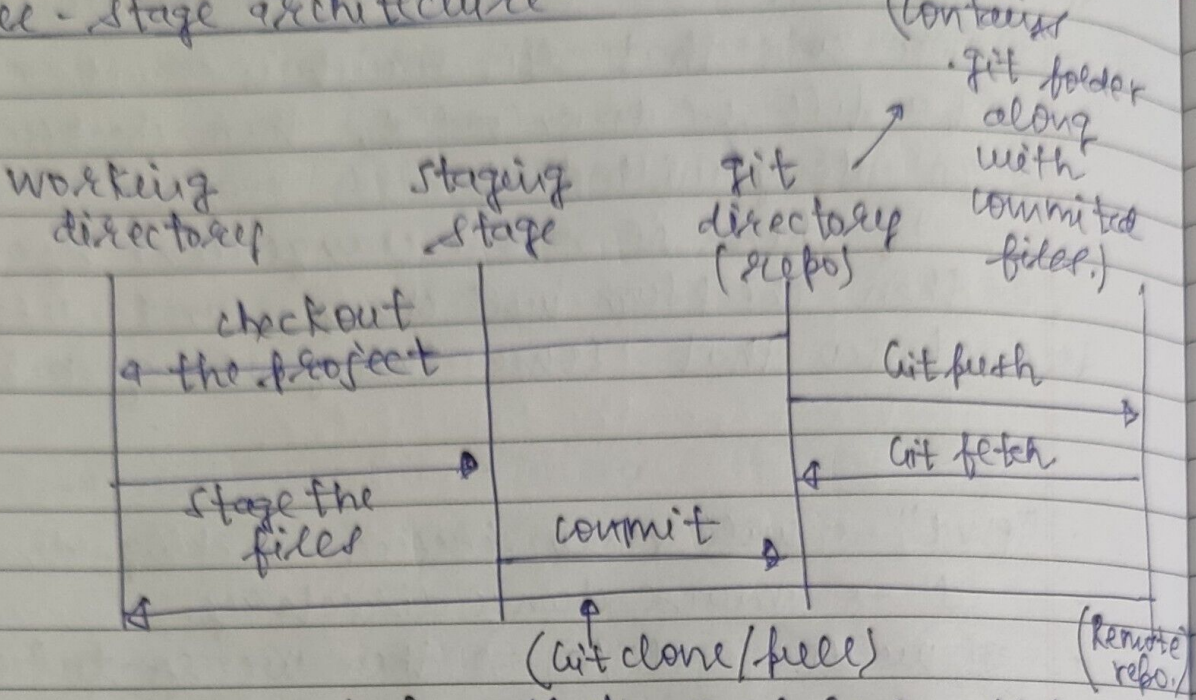
- git config --list ⇒ will give the whole list.

(a) But it will print it three times.
(b) As it "global" will store the data in three levels.

- git config user.name
- git config user.email

Page No.:
Date: / /

Three-stage architecture:-



- staging area contains all those files which will be commited in the next commit.
- Three-stage architecture comes, so that some files in the already commited folder need not require commit while some require.

Example:-

- go to any folder and open both these.
- git status ⇒ (will show that no repo exist)
- git init ⇒ (will make that folder, a repo)
  (.git folder will be made)
- git status ⇒ (will show that ~~nnnnnnnnnn~~ files are present in their folder) "untracked"

{ Note that branch will be "master" }

#clear → (will clear the bash terminal)

Page No.:
Date: / /

- git add --a ⇒ will add all the files to the staging area
- git status ⇒ will show : changes to be committed.
- git commit -m "Anvit Initial Commit"
  { snapshot will be taken } ⇐ commit all the files in the staging area with the explicit message
- git status ⇒ will show : nothing to commit
- git log ⇒ will give history of all the commits.
- If we change two files : { first.txt, fec.xlsx }
  then: git status ⇒ changes not staged : 2 files
- git add first.txt ⇒ will only add one file to staging
- git status
- git commit -m "changed first.txt"
- git status ⇒ will show one unadded file : "fec.xlsx".
- rm -rf .git ⇒ will delete the .git folder and also the repo will n't be a repo. (folder).
  ↑ permanently delete the repo tag of directory

(ctrl + insert ⇒ copy
shift + insert ⇒ paste)

Page No.:
Date: / /

☆ Cloning a remote repositorey from GitHub :-

- Go to that repo and copy it's link to clone.
- git clone https://github.com/tensorflow/tensorflow.git
  - (paste that link)
  - [will take some time]
- ls ⇒ list ⇒ will give all the files and folders in that directorey.

Any changes done to the folder, cloned ⇒ will not be seen in the original repo on github.

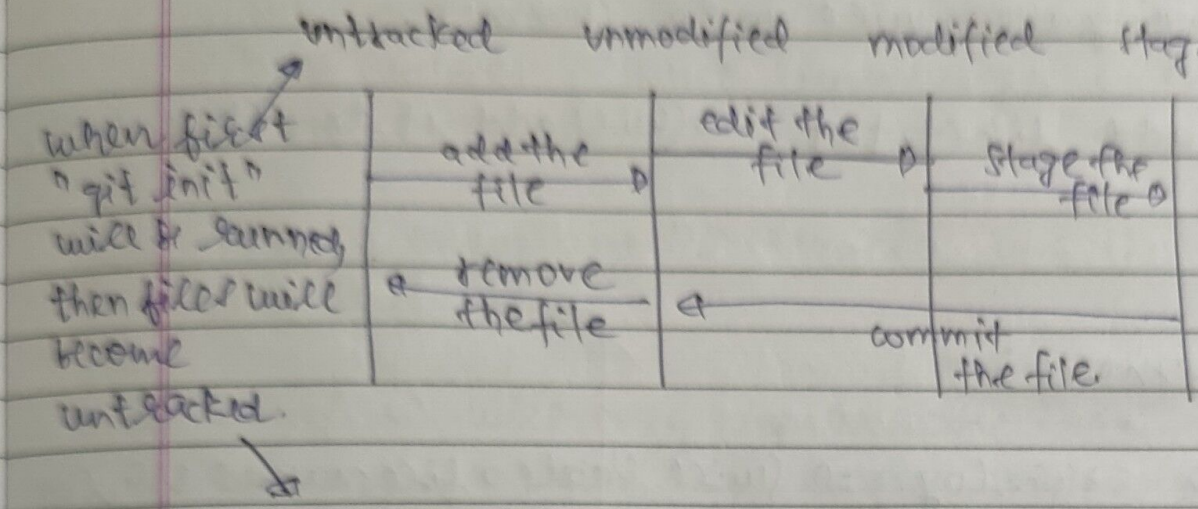
- git log ⇒ will give the history of all changes done.
  - [For exit, type 'q']

Note: git clone <link> mytensor ⇒ folder formed will be named as "mytensor".

Repositorey : A folder which is begin tracked by the git.

Page No.:
Date: / /

☆ File status life cycle :-



- When "git add --a" or "git add" will be executed, then they come to unmodified state. (staging area)
  If any change made, then modified.
  Again git add --a → staging area

- Finally, "commit" will say that - "we have taken snapshot and now the files are unmodified".

Example :- git init
git status → (will show untracked files)
git add --a
git status → will show : changed to be comitted → means in staging area (we call it unmodified)
— change —
git status. → (will show that first.txt is in staging area (unmodify) as well as changes need to be staged)
(in modify).

Hence, changed file will be now in staging as well as modify

If we do commit at this stage, then changes will be ignored and the files that are in the staging area, will go to commit. ..

- git add first.txt → (will add it to stage area).

☆ Git ignore :-

- touch err.log ⇒ (will create a file named err.log in repo)
- touch .gitignore ⇒ (will create a file).
- git status ⇒ will show two untracked file (as they are made now)

Write "err.log" in .gitignore file.

- git status ⇒ will only show that .gitignore is untracked.

- git add --a
- git commit -m "."

If I made any changes to err.log, it will then also ignored.

Note : (a) (*.log in .gitignore ⇒ ignore all log files)

(b) For any directory named anv :
anv/ ⇒ ignore the "anv" directory.

⇒ "git add ." will work same as "git add --a"

Page No.:
Date: / /

(c) Sub:- Git will ignore the blank folder. If i made a "static" folder and within it "anv" folder ⇒ Git is already ignoring the "anv" folder, hence ignoring the empty "static" folder.

- Now if we add ".txt" file and then commit all the changes.

(d) /anv/ ⇒ will only ignore the "anv" folder (inner) in that directory. only, not in any folder directory

Changing anv/ to /anv/ ⇒ will give static/anv/ to be untracked.

- git diff ⇒ will compare the staging area with working directory.

If files added to stage, and then some edit, then above command will show changes made.

git diff will not show anything when there is no editing and staging area and working directory are same of all.

- git diff --staged ⇒ compares staging area with last commit.

Page No.:
Date: / /

The message in the commit will be such that it shows the actual changes in the file. and not that "this file is being changed" → as commit means that only. (commit already contains date already)

Direct commit:-

- git commit -a -m "..." ⇒ will move all the "tracked" files to the stage and commit also.

If untracked present, then it will not go anywhere.
git status ⇒ will show it untracked.

(a)
- git rm third.txt ⇒ will remove the file and also add changes to staging area.
- git status
  ↳ (will show "changes to commit")
- git commit -m "..."

(b) The above method is easier to delete any file, as:
When we delete manually, then changes will be manually pushed to the staging area.

(c) git mv first.txt first1.txt ⇒ will rename the first file and also change added to staging area.
(mv = move)

Page No.:
Date: / /

Important :-

- Let's say we have a file = "db.accdb" which is tracked and comitted.
  Now we add it to .gitignore, modify it:

  - git status ⇒ will still show modified "db.accdb"

  Reason :- It is already tracked.
  For placing it to .gitignore or ignoring its changes, we must bring it to "untracked":

  git rm --cached db.accdb ⇒ will bring tracked to untrack.

  Now ⇒ it will ignore.

☆ Viewing the commits :-

- git log ⇒ all the commits will showed
- git log -p ⇒ will also show what is changed (diffs) along with commits.
- git log -p -3 ⇒ will only show previous three commits along with diffs.
- git log --stat ⇒ will give intro of all the commits.
- git log --pretty=oneline ⇒ will show all the commits in one line
- git log --pretty=short → (only author
  git log --pretty=full → both author commit)

Page No.:
Date: / /

Author and commit:

Author => who made the file firstly
Commit => who changed it.

- git log --since=2.days
  git log --since=2.month
  git log --since=2.years
  git log --since=2.hours.

Filter:

- git log --pretty=format:"%h--%an"
  (%h = abbreviated hash, %an = author name)

  (show all the commits or the string mentioned)

  %ae = author email (https://www.git-scm.com/docs/pretty-formats).

Appending our change into previous commit:-

- do some change, get to staging area:
  "git commit --amend" → and enter.

=> Another window opens : Type 'i' to start text type.

=> Change the message of the commit and then: press escape ; ":wq" → and then enter.

Unstaging and unmodifying files:-

- git restore --staged anvit.txt => will unstage file => [move it "modified" phase]

(also git restore ankit.txt) ⇒ (will match it with the last commit)

Page No.:
Date: / /

- Now, let's say we have done some changes that we don't want to do, hence:

"git ~~restore~~ checkout -- ankit.txt" ⇒ will bring that file to last commit stage.

(Note that this command will not work when ankit.txt is in staging area ⇒ it remains there)

(also working tree will get clean)

- We have modified many files and want that working directory will match with the previously commited stage.

→ (No commit req.)

"git checkout -f" ~~And~~

☆ Working with remote repositories on GitHub :-

- Push ⇒ the thing the that i have push, will be pushed.
- Pull ⇒ all code & history will be moved from remote repo to device.

(i) git remote ⇒ initialise the remote

(ii) git remote add origin (git@github.com...)

↓

we are giving name to the whole repo link as "origin"

[Good practice to give this name].

Page No.:
Date: / /

- git remote ⇒ will now show a remote repo as "origin".

- git remote -v ⇒ will tell you in which repo you will be pushing and pulling / fetching.

- git push -u origin master ⇒ will say you don't have access.

⇒ For generating SSH key:
go to google ⇒ github ssh key ⇒ open github page.

- ssh-keygen -t rsa -b 4096 -C "<email>"
- eval $(ssh-agent -s)
- ssh-add ~/.ssh/id-rsa
- tail ~/.ssh/id-rsa.pub ⇒ copy the key generated.

⇒ Go to github setting → SSH and GPG keys and add a new SSH.

Now: "git push -u origin master" will work.

If any changes made in working directory then,

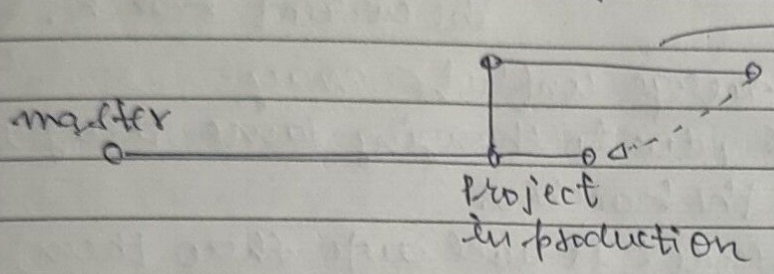
{ git add.
  git commit -m ".."
  git push -u origin master. } ⇒ will push the file change. or commit.

Page No.:
Date: / /

☆ Creating alias of shortcuts :-

- git config --global alias.st status
- git st ≡ will behave like git status.
- git config --global alias.unstage 'restore --staged --'
  ⇓
  now: "git unstage anvit.txt" only.

☆ Creating and switching branches :-



- Working directory will change according to the branch switching

Example:- git checkout -b dev ⇒ (will create and switch to "dev")

git checkout master ⇒ (will go to master)

Note that : the files that are ignored by the git will remain as it is when dev to master switch occurs:

Ex: We combined all the seperate "log" files into 1 folder (dev.)
switched to master
Remains as it is.

(After branch "dev" to master ⇒ "dev" will also exist)

Page No.:
Date: / /

- git branch ⇒ will give all the branches.
- git branch -D dev ⇒ will delete the "dev" branch.
  (make sure you go out of that branch)

Whenever we are checking out the branch, make sure all the changes are committed.

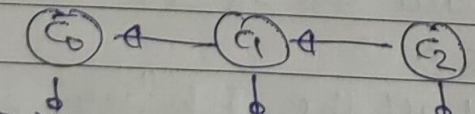
- git merge dev ⇒ will merge the dev with the current branch. (changes.)

Note: Here merge conflicts may arise, due to changing same thing in both the branches.
(Use vscode: it will auto show them)
("stage and")
Finally commit the merged state.

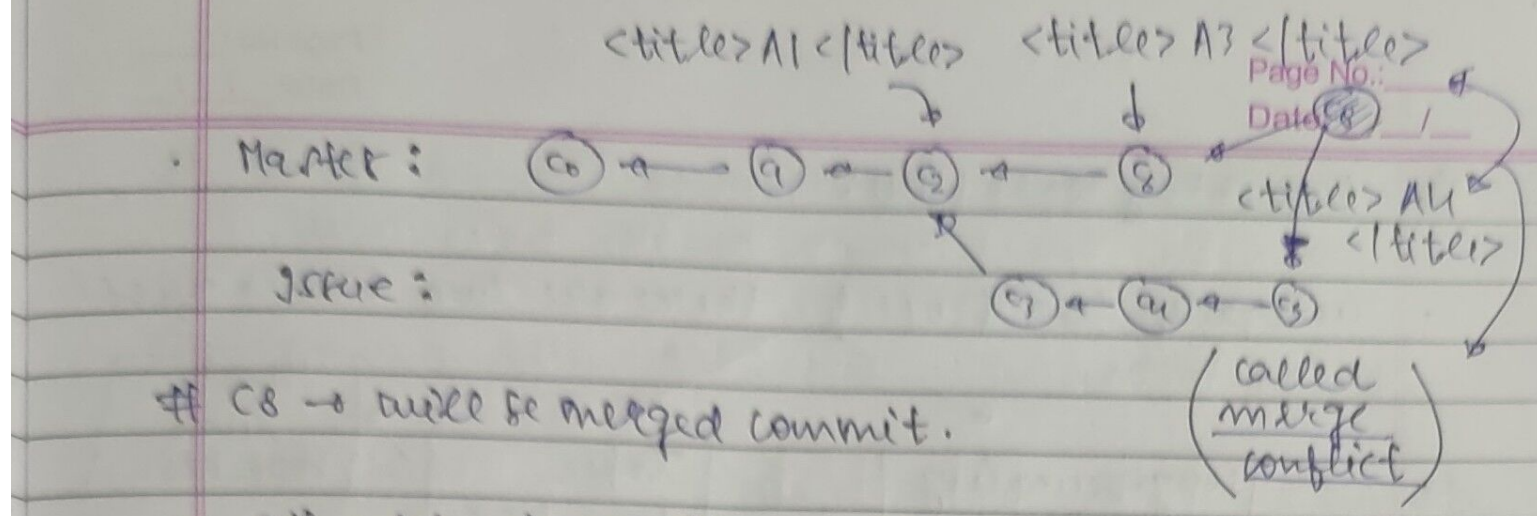
Git log will always show all the commits in all the branches.

☆ Merge conflicts :-

- Commits are actually objects.



⇒ Any commit will have a pointer to its previous commit.



C8 → will be merged commit.

- git add index.html ⇒ this command also resolve merge conflict
  (will do after accepting one from vscode)
  (will mark the merge conflict as resolved)

☆ Branch management:-

- git branch ⇒ will show all the branches (along)
- git branch -v ⇒ will show all the branches with last commit.
- git branch --merged ⇒ already merged branches
- git branch --no-merged ⇒ not merged branches.

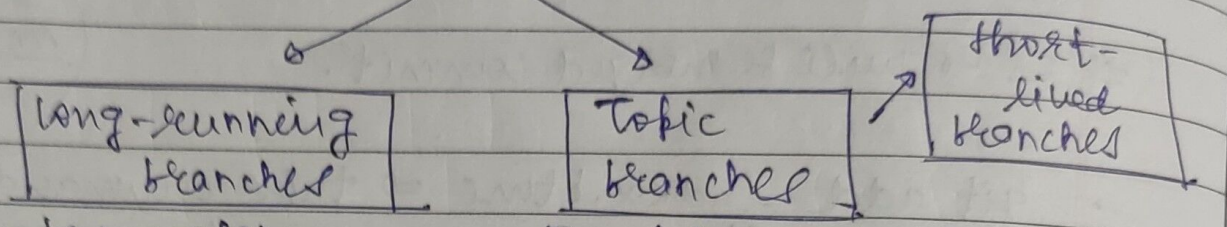
Deleting:-

- git branch -d develop ⇒ will give warning when "develop" branch is not merged. deletes the merged branch
- git branch -D develop ⇒ automatically delete branch without warning.

Page No.:
Date: / /

☆ Branching workflow :-

- Generally in a project, there are two branch types:



- long-running branches
  - master
  - develop
  - Proposed Update
    - ⇓ will live until the project is in production
- Topic branches
  - Fx: typedir ⇒ integrate type.js and remove react.js
    - ⇓ will remove or deleted after the merging of changes.

☆ Pushing branches to git repositories :-

- Branches have to be pushed explicitely.

(git push -u origin master
git push origin develop.) (on remote, it will be mydev)

⇒ git push origin develop : mydev

Note : Recommendations :-

(a) Before switching to other branches, firstly commit all the changes in existing branch.

(b) If pushing the branched named "dev", you must be in that branch.

Page No.:
Date: / /

(c) Don't give different names to branches on remote and local system.

☆ Deleting remote repositories and branches :-

- git push -d origin bugfix ⇒ (will delete the bugfix branch).
- No command to delete the actual repo on github. But you can delete the local copy of it : rm -rf .git.

☆ Pulling changes from the remote repository :-
git pull origin master ⇒ will pull the changes made on github repo.

If you have a merge conflict, you can also abort the merge :
git merge --abort ⇒ (undo a merge).