**GITHUB**

**GitHub:**

GitHub is a free and open source version control system used for source code management, tracking the changes in the source code. It allows to developers to work together.

Features:

**Build code**: Developers can use GitHub to write a code, track changes.

**Share work**: we can showcase the and let others review it

**Collaborate**: Multiple people can work on a same project simultaneously without worrying about conflicting work.

**Git:**

Git is command line interface toll used to interact or manage the remote repositories from local

**Git Branch:**

Branches used for parallel development for two or group of people work on same piece later we can integrate by merging.

git init 🡪 initialize the git on local on first time

git branch <branch\_name> 🡪 used to create branch

git branch 🡪 list the branches / can show current branch

git checkout <branch\_name> 🡪 used to switch between the branches

git branch -b <branch\_name> 🡪 create new branch and switch to same branch

**Git clone:**

Used to bring the remote repository to the local for the first time, or copying the remote repo to local.

git clone <repo url>

**Git add:**

Adding the file to staging area. It moves the file from workspace to staging.

git add . or git add <file\_name>

**Git Commit:**

Git commit will moves the files from staging area to git repository ( can give message about the changes)

Git commit -m “message”

**Git Push:**

Used to push the changes from local repo to remote repo.

git push origin <source branch>:<destination branch>

**Git pull:**

used to bring the changes from remote repo to local repo and it will automatically merges the changes to your current branch so we need careful when pulling the code.

git pull <repo url>

**Git fetch:**

Used to pull the changes from remote repo to local repo and it won’t merge the changes to your current branch you can create branch then you can merge manually if required.

git fetch <repo url>

**Git status:**

This command is used to check whether the files in workspace or in staging area.

**Git log:**

This command can be used to check the history of the repository.

**Git Merge:**

Git merge is used to merge the changes between branches or integrate the changes from one branch to another branch.

It will create new commit for each merge.

Merge history will be added in chronological order (new one is on top)

**Git rebase:**

This is also used for merging the changes from one branch to anther

It won’t create any commit id after the merge

History will be added in linear order (older one is on top)

**Git revert:**

Git revert is used to undo the committed changes but it doesn’t delete the history we can track easily who has done the revert because it creates new commit id for each revert.

**Merging conflict or git conflict:**

It will occur when same piece of code is changed in two different branches, so when we try to merge the branches conflict will occur.

As a devops person I don’t know which changes should I merge. So I will contact the developer who modified the code on the two branches using git log. They will decide which changes need to keep in then file then we will start merging.

**Git cherry-pick:**

This command is used to merge or update specific commit to your branch.

git cherry-pick <commit id>

**Git squash:**

This command used to combine multiple commits into a single commit. it is useful when your branch has many small commits that make the more history to avoid this we can used git squash.

Git rebase -I HEAD~[number of commits to squash]

Git rebase -squash

**Git stash:**

If I am working on one branch if get critical work on another branch I need to switch to other branch without completing other work. If I switch to other branch all the files are on the workspace or staging area will be visible in another branch. To avoid wrong commit on other branch I will stash it before switch to another branch.

So this will store it on temporary, once you complete that critical work on another branch you can switch to main branch where you working, then you can resume your work by running git stash popcommand.

git stash

**Branching strategy:**

We have created branches for releases.

so Development will be going on in development branch, Once the

code is ready for 1st release, we create the branch release 1.0 from the

development branch and we make 1st release from the release 1.0

branch.

Simultaneously development will be going on development branch,

for the 2nd release, whatever the issue we got on release 1.0 , we fix it

on release 1 branch and it acts as a maintainance branch for release 1

Once the code is ready for 2nd release on the development branch,

before we create release 2 branch for the second release we merge

release 1 branch to development branch, then we create release 2

branch from the development branch. So whatever the issue we faced

in release 1 that should not be visible in release 2 and further release.

Same thing will be happening for the further releases.