Date
Page

- Git is a open source distributed version control system. Ho
- Control system :- This basically means that Git is a content tracker.

- Day 1 :- Agenda :
- Introd^n to GIT
- Repository
- Storing changes into GIT repository
- Branching
- Hygieines
- Connection with Remote Repository (Push & Pull)

- Introduction
- Git is a open source Distributed version control system.
- Control system :- Git is a content tracker.
- Version control system a) The code which is stored in GIT keeps changing as more code is added.

  b) Also many developers can add code in parallel. So version control system helps in handling this by maintaining a history of what changes have happened
- Distributed version control system :-

  a) Git has a remote repository which is stored in a server and a local repository which is stored in the computer of each developer

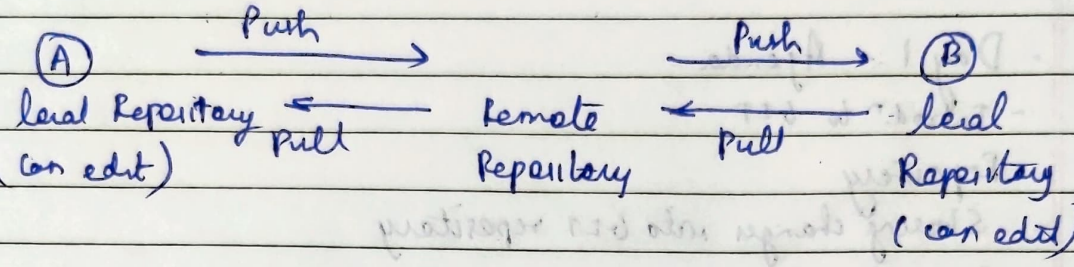
  b) This means that the code is not just stored in central server, but a full copy of the code is present in all the developer's computers.

  c) Git is a Distributed version control system since the code is present ~~on~~ in every developer's computer.

Date
Page

- Repository

A repository (or repo) is a location where all the files for a particular project is stored.



- git --version.
- mkdir ecommerce. (creates a directory - Directory is a file system folder where you can put all documents, images, videos etc)
- cd ecommerce (select the folder ecommerce).
- git status.
- git init (initialize git into folder and creates a folder .git)
- cd .. (To go to previous step).
- ls (list all folders present in repository.)

- Master is a default branch.
- Steps for setup :-
  1. Clone the repo
  2. Install go
  3. install postgres
  4. run migrations.

- notepad Readme.md. (Creates Readme file).
- Untracked files (all new creates files).
- git add Readme.md (send all new files to staging area)

Staging must be done before ~~com~~ committing.

- make changes → Take it into staging area → commit it.
- Head is a pointer.

Date
Page

- git commit -m "Added Readme file with setup steps".
  ↳ message.

- To make configuration
  - git config --global user.email "you@example.com"
    - | | - user.name "Your name"

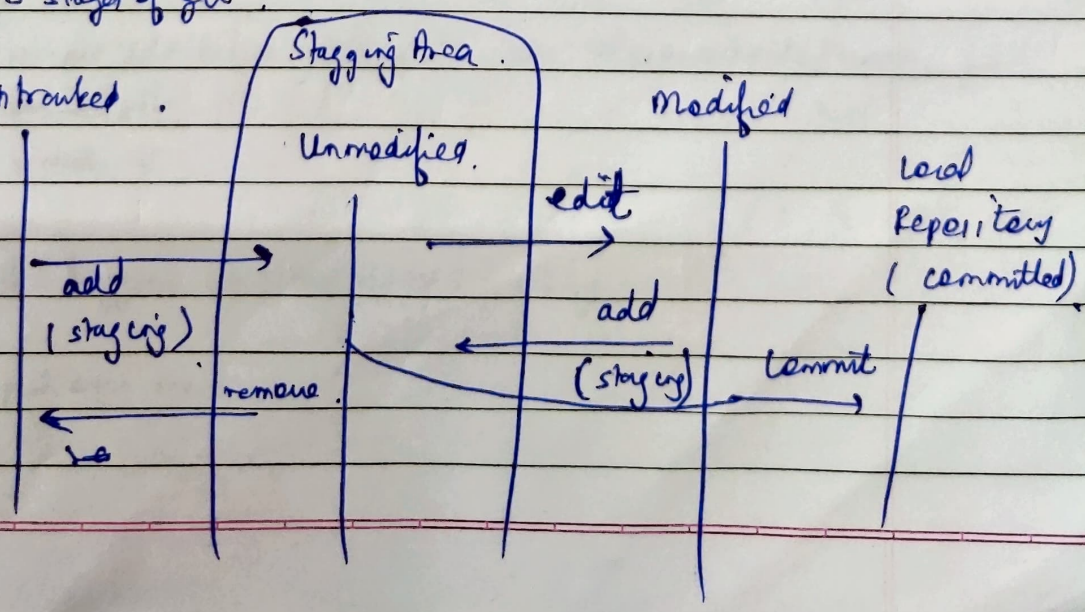
  If no global key word is used it will make locally.
  To check name & value use
  - git config user.name
  - git config user.email.

- git log (shows all the changes we did).

  head always point to latest commit. (head → master)

- cat HEAD
  → Allows us to create single or multiple files, view view contain of file, concatenate files and redirect output to the terminal or to file.

- cat ~/.gitconfig
- The 3 stages of git.



log, reset, diff, show, checkout, revert, soft, hard.

Date
Page

- git ~~reset~~ Readme.md (~~test~~ reset the changes from modified area to stagging area).

git checkout Read.md (~~checou~~ checkout is getting out from the local or remote repository (into your local working directory) make changes from stagging area to modified area.

- git diff (show the difference ~~betn~~ betn ~~addo~~ new and previous files)

or we ~~and~~ can directly use
- git reset --soft HEAD~n.
  → By default 1st one we can specify the no. of HEAD in front.

- git ~~tol~~ reset ~~--soft~~ --hard HEAD~n

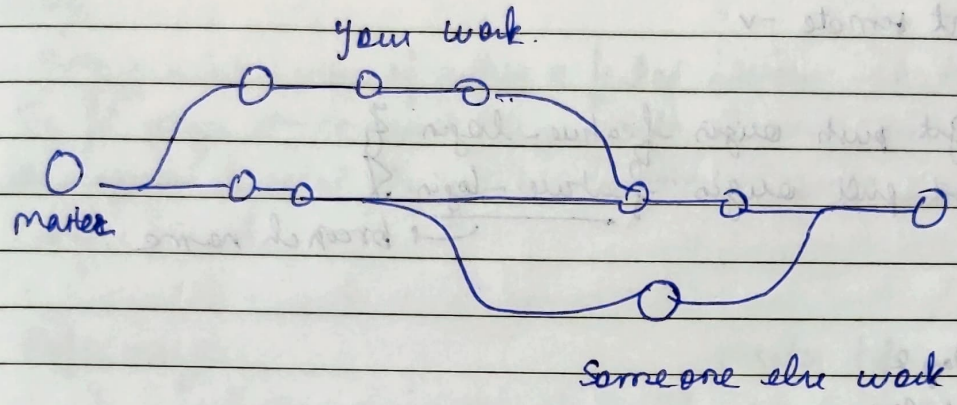
Soft will keep your files, and stage ~~all~~ all changes back automatically
hard will completely destroy any changes and remove them from local directory.

- type Readme.md → will display all the txt from Readme.md changes.
- revert (~~deleds~~ Delete and also maintain the history)

. ssh keys are used to ~~for~~ authenticate

Date
Page

- GIT Branch.
  - A branch ~~rep~~ represents an independent line of development.
  - You can think of them as a way to request a brand new working directory, staging area and project history.
  - New commits are recorded in the history for the current branch.



→ git branch Feature-login.

→ git branch → two branch ① feature-login
② * master

→ git checkout feature-login → will switch from master to feature-login

→ git branch -d feature-login → use to delete the branch. branch
(deletes permenantly).

- Remote.

→ git remote add origin git@github.com: ~~rel~~ pranavkonde/swiggy_git_commerce_git.

→ git remote -v.

→ ssh-keygen → will ~~use~~ create ssh key.

→ gd push origin master.

Origin - alias pointing to ~~re~~ remote repository.

Date
Page

- git checkout feature-login

- for cloning
  git clone link (git@github.com ...)

  git remote -v

- git push origin feature-login
  git pull origin feature-login
  → branch name.

- Day 2.
  - Git Merge.
  - Merging strategies
  - Merge conflicts
  - Git Rebase
  - Git Rebase vs Merge
  - Creating Pull Request..

- Git hygiene for commit.

  Bad - git commit -m "Password feature fix changed"
  Good - git commit -m "Update password ~~a~~ length to 8 frame"

  Good-to have - Your Story Id should be included in the message.

- Git hygiene for branch.
  - Start a branch name with a group word (bug/story/wip) ~~as~~
    ex:- bug-log-alignment-issue.

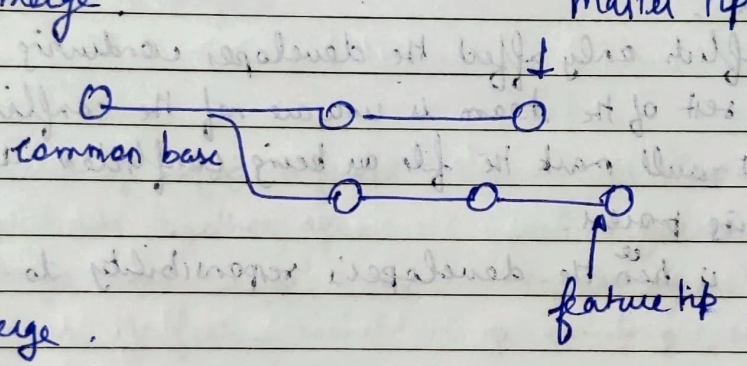
Git merge will combine multiple sequences of commits into a unified history.
In the most frequent cases, git merge is used to combine two branches.

Date
Page

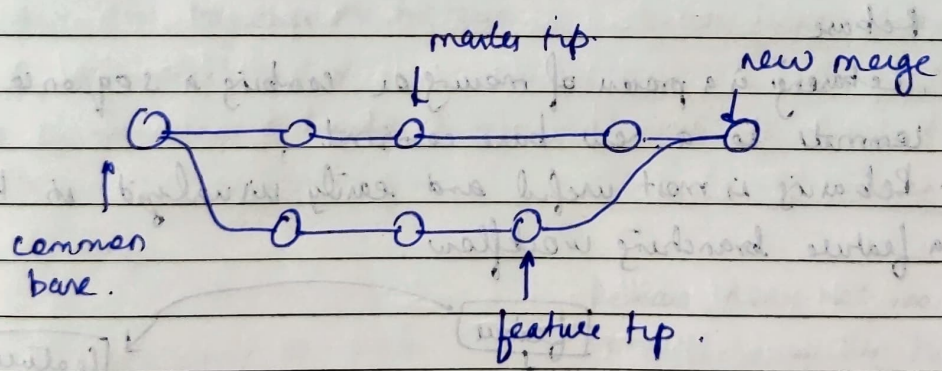
- Use PMT unique task en wp-8712-cold-testing-module.
- use hyphen and slash as separators.
- Avoid using number only.
- PMT - Project management tool.

- Merge.
- Merging is Git's way of putting a forked history back together again.
- The git merge command lets you take the independent lines of development created by git branch and integrate them into a single branch.

Befor merge.



After merge.



- git merge feature-registration.

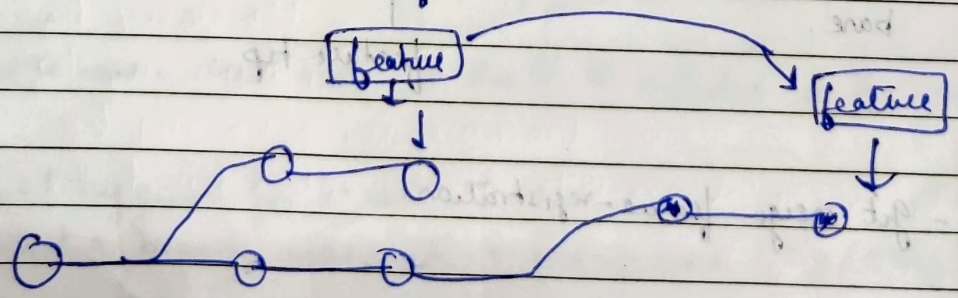
Head → current branch.
merge & Rebase are use to solve ~~rare~~ problem (conflict)

— git checkout -b "feature-registration
· to create new branch and switched to that branch

— git log --graph --decorate --all --online

— Merge conflicts.
- Conflicts generally arise when two people have changed the same lines in a file.
· OR if one developer deleted a file while another developer was modifying it.
· In these case, GIT cannot automatically determine what is correct.
· Conflicts only affect the developer conducting the merge
· The rest of the team is unaware of the conflict
· GIT will mark the file as being conflicted and halt the merging process.
· It is then the developer's responsibility to resolve the conflict

— Rebase
· Rebasing is a process of moving or combining a sequence of ~~commits~~ commits to a new base commit.
· Rebasing is most useful and easily visualized in the context of a feature branching workflow.



* - new commits.

ls / dir.] The upstream is from where you clone the repository and downstream is any project that integrates your work with other works.

- git rebase master

- git rebase --continue

- git push origin registration-button-component-update

Now we can create ~~manual~~ manual or directly on Github (pull request)

- Pull request :- A pull request is an event in Git where a contributor asks a maintainer of a Git repository to review code they want to merge into a project

- ~~Rebase~~ Merge

① It is a way of putting a forked history back together again.

② git merge creates a new "merge commit" in the feature branch. both ties together the histories of both branches

③ If you want to update public branch you should use git merge

④ If you want to ~~push~~ to update public branch you should use git merge.

Rebase

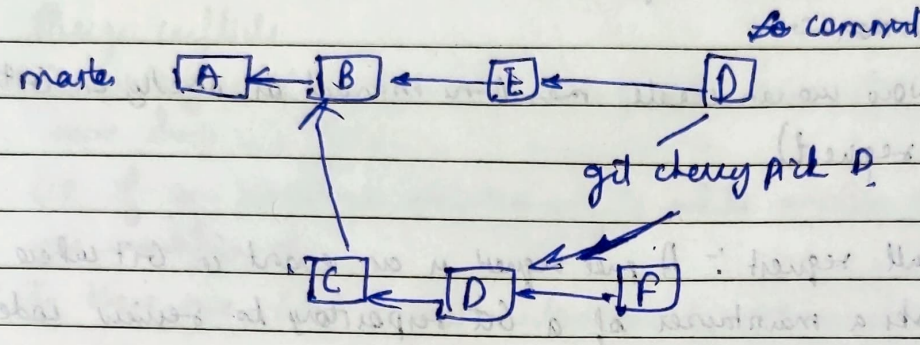
① It is also a way of putting forked history back together again.

② While git rebase moves the entire feature branch to begin on the tip of the master branch, effectively incorporating all of the commits in master.

③ Rebase does not use commit; rebasing re-writes the project history by creating brand new commits for each commit in the original branch.

④ If you want to update your feature branch you should use git rebase

- ~~cherry~~ cherry pick - Stands for applying some commit from one branch into another branch. In case you made a mistake and committed a change into the wrong branch, but you does not want to merge the whole branch, you can select the commit and apply it on another branch.



git cherry-pick <commit hash>>

- Set multiple remote repositories for the same project. You can use as many remote as you want, but you can only have one remote named "origin". The "origin" remote is not special in any way, except that it is the default remote created by Git when you clone an existing repository. You can configure a second remote, push to pull from that remote, and set up some branches to track branches from that remote instead of origin.

Steps :-

- git remote add github url.
- git push master.
- git push -u github my-branch
- git branch -set-upstream other-branch github/other-branch

Date
Page

- Squash.
Squash is used to squash the previous commits into one, It is not a command instead, it is a keyword. The squash is excellent technique for group specific changes before forwarding them to others. You can merge several commits into a single commit with the compelling interactive rebase command.

Steps:-

```
git log --online.
~~git rebase -i Head ~3~~
git rebase -i HEAD ~3
git push origin master.
```

- How to delete one commit or multiple commit permanately
deleting the commit in git must be approached in one of two ways depending on if you have or have not pushed your changes. Running these commands will Delete your working directory changes. Any changes to tracked files in the working tree since "commit" are also discarded. Be sure to separately save any changes you'd like to have

1. If your changes have not been pushed yet simply enter the command
- git reset --hard HEAD~1
This will discard all working tree changes and move HEAD to the commit before HEAD.

2. If you have already pushed your changes you will need to run the following code.
- git push origin HEAD --force
If others have pulled this branch you would be ~~the~~ better off starting a new branch. If you don't do this when someone else pulled, it will just merge onto their work. and you will get it pushed back up again

- Diff with --force push and --force-with-lease :

→ : force overwrites a remote branch with your local branch.

: force-with-lease is a safer option that will not overwrite any work on the remote branch if more commits were added to the remote branch (by another team-member or coworker or what have you). It ensures you do not overwrite someone else's work by force pushing.