Merging Strategies

Day 10

$$\begin{tikzcd}
 & & \text{Merging Strategies} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\text{cherry-pick} & \text{fast-forward} & \text{ORT} & \text{octopus} & \text{Squash.}
\end{tikzcd}$$

1.> git cherry-pick applies a specific commit from one branch onto another without merging the full history.

2.> A fast-forward happens when the current branch pointer is simply moved ahead to match the merged branch.

3.> ORT:⇒ ORT is Git's default merge strategy since Git 2.34, designed to be faster and handle complex merges.

4.> octopus :⇒ git merge with octopus is used to merge more than 2 branches at once.

5.> squash combines multiple commits into one to simplify history Before merging.

1.> → git branch bug/126

→ touch gfg.txt] commit (Took lot of time).

→ update the file.

↓

again Do commit

⇓

Doing Multiple commits.

→ git switch master.

→ git merge bug/126] Make Master branch Commit history Messy.

⇓

git merge --squash bug/126.

⇓

commit

All commits will be 1 Commit and then Master will be saved

⇒ git log --oneline. (only one commit is relate to bug 1126 file).

⇒ feature branch is unaware of the Master branch changes. (Interested only in particular commit). Want to copy to My Branch.

$\boxed{\text{git cherry-pick commid\_id}}$ → can pass Multiple Commit Id's.

⇒ $\boxed{\text{git cherry-pick --contine}}$

$$\begin{tikzcd}
 & \text{Branches} \arrow[dl] \arrow[dr] & & \\
\text{Feature} \arrow[rr, bend right] & & \text{Master} \arrow[ll, bend left] & \\
 & & \text{Integrating} \arrow[r] \arrow[dr] & \text{Merging} \\
 & & & \text{Rebasing.}
\end{tikzcd}$$

$\boxed{\text{Rebase}}$
⇒

mK dir git-day-second

⇓

Cd to this Dir.

⇓

git init

⇓

git log

⇓

create a file & commit

⇓

git branch feature/Post. (on Master only)

⇓

git switch feature/Post

⇓
New file created & commit
⇓
git switch master

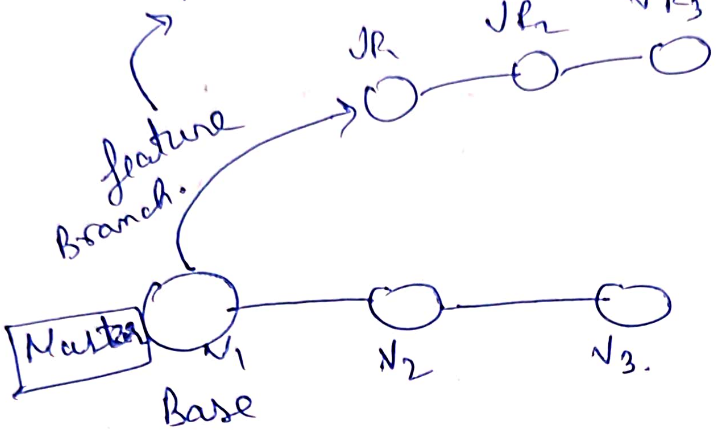
} I am only one switching b/w Master & feature.

⇓
git another file & add (second by master)
⇓
git switch feature|post.
⇓
Make one more file & commit.
⇓
git log --oneline.
⇓
git switch master
⇓
one more file & commit.
⇓
git switch feature|post
⇓
add new file and commit.
⇓
git log --oneline
⇓
ls
⇓
git switch master
⇓
git log --oneline. (on Master).
⇓
git switch feature/post.

rebase (Vs) Merge

↳ No extra merge commit Happen

Must Have all commits that are in Master branch.



Base commit

⇓

→ git rebase master (on feature Branch).

In this history is maintained in a better way. (linear way).

(No conflict as both Have Different Data).

⇓

ls

⇓

git log (History Here is Different than Merging).

↳ history is scattered by time.

Before Merging do the Rebasing part

. will not Make any changes in the Master.

rebasing is done (Alert Mode).

(Makes sure Not Breaking anything).

↳ use only in feature branches. (Not pushed to GitHub)

(own private branches).

⇓

git switch master.

⇓
git log.

(want to rewrite the History).

⇓

git rebase (-i) HEAD~3.

interactive. latest commit Id.

⇓

Insert Mode (reword).

rebase → History → Commit (Modify the commit Message with new commit Id).

⇓

edit ⇒ go back to this commit and Made some Modifications.

⇓

touch sfg1.txt (forgot to add last Time).

⇓

amend file into Particular Commit.

~~git [illegible]~~

⇓

git rebase --continue.

git garbage collector

↓

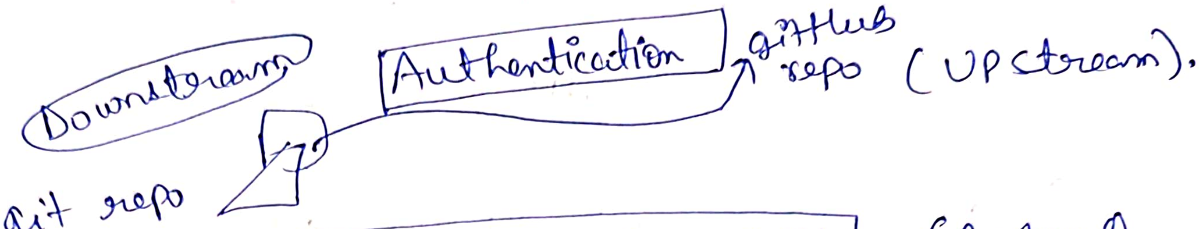
Removeds all the unused commits

git stash)
git stash pop

GitHub (Need to Integrate.

⇒ git remote -v (whethere My local Git Knowns remote repo or not).

origin ⇒ just an alias.



Can use HTTPS or SSH (Protocol)

. Need to put the token at any Configuration.

git remote set-url origin ←——

https://username: token @ ———— repo name.
→ Take this from file.

⇓

git remote -v

⇓

git push -u origin. branch_name.

Behind the scenes used HTTPS protocol.

Setting →

Delete feature branches once that part is done.

Open Source projects

tensorflow

→ git fetch (what is Happening in remote System).
↳ Local git will go back to GitHub and fetch Metadata from it.

⇓

git merge (from GitHub to Git).
↳ Merged GitHub master branch changes into local gitHub branch.

$\boxed{\text{git pull} \Rightarrow \text{git fetch + git merge}}$
↳ fast-forward.

fork (Repository) Means add in your GitHub A/C.
(GitHub A/C)
[Contribute] option
↳ to the original Repo (which is open source).
(Intra a GitHub A/c).

→ [git clone ⟶ url]

Ssh is more Secure → Create a pair of Private/public Key
[then HTTPS]
↙ ↖ Username/Password.
Token

↙ Give to GitHub.

⇒ ssh-keygen ↓
[notepad ⟶] ↓
(GitHub A/C) → Delete the Token.

Setting → SSH & keys → (Paste the Public key).

⇒ git remote add origin — ssh from GitHub.

⇒ git remote remove origin

⇒ SSh -T git@github.com

check connection.

→ git Config -l

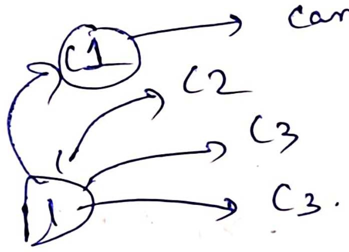
Imp username & useremail.

Change config info

⇒ git config user.name new name

⇓

can set username something like this.



⇒ all the repos.

git config user.email — email.

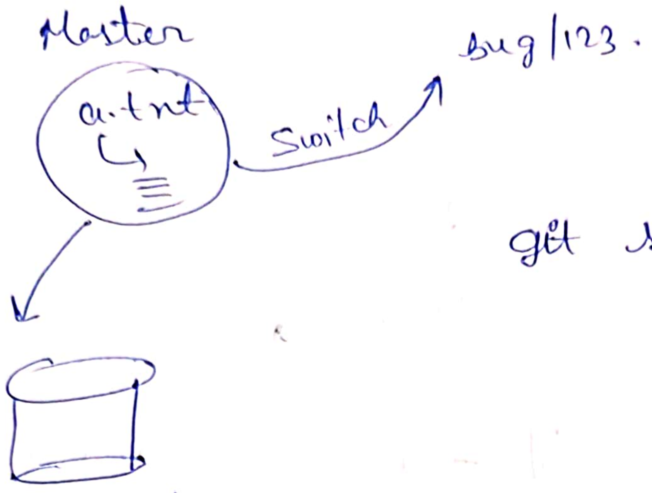
Global ⇒ git config user.name

⇒ git stash

git stash list → 1 Start will be Here.

git stash save — incomplete a.txt changes → Message Here

Flow Diagram



Stash Area.
will be removed from working Area.

git stash show → all the stash list with latest on top.

git stash pop → latest one → to take Back Data

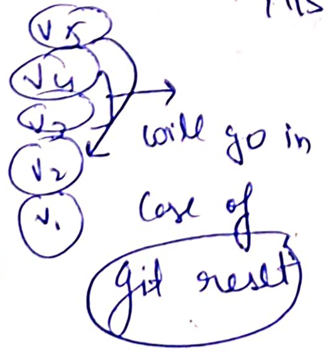
Poop stash

⇒ git stash drop (stashNo)

⇒ git hooks → After commit → Automatically push the changes or like formatted or not (Python). Can add some custom conditions / send Notification to someone.

1.) git revert ⇒ go back to previous version and commit History should not go.

will create one extra Commit → revert commit



git revert HEAD~1
git revert HEAD (will revert the latest message).