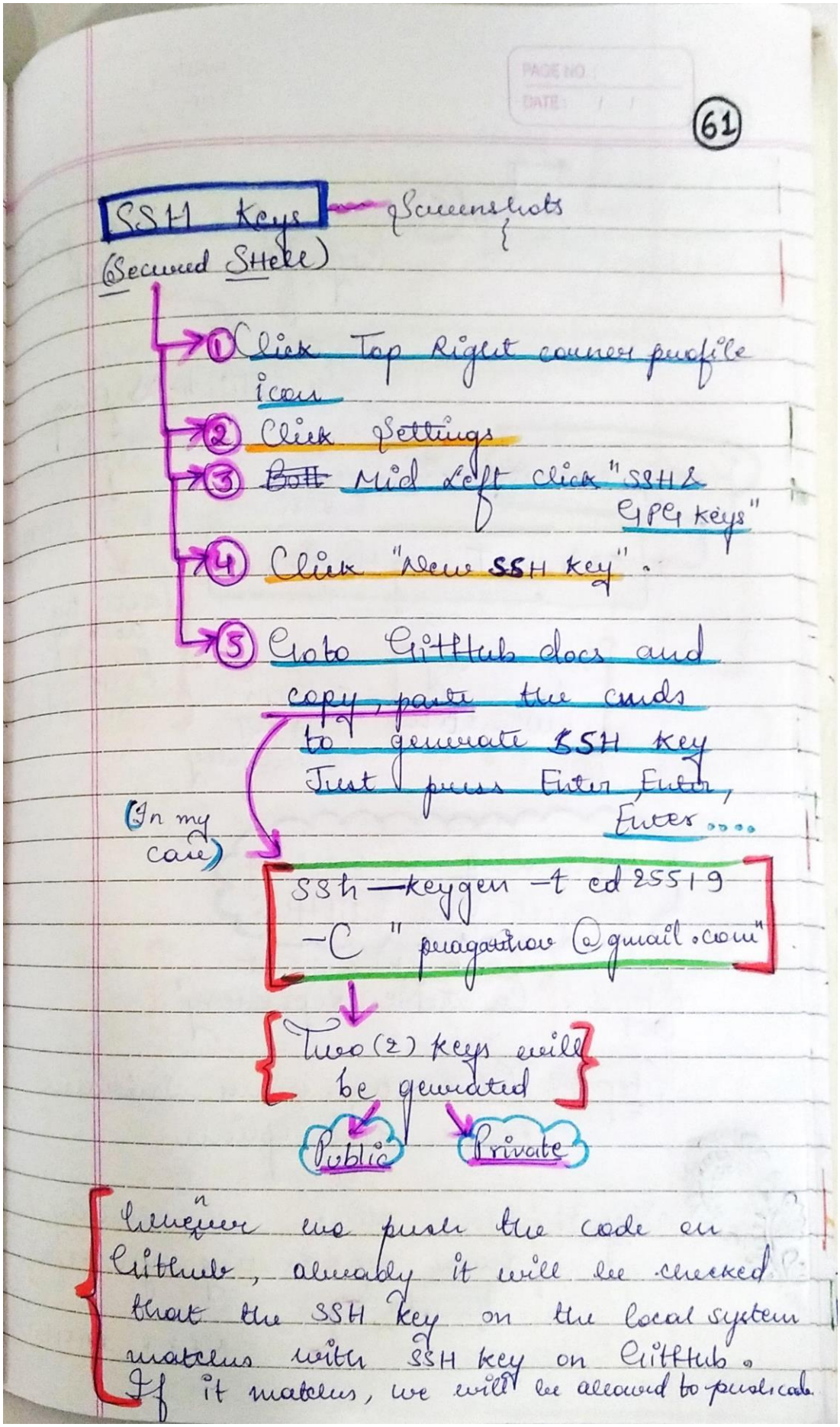

61
SSH keys (Secured SHeLL) — Screenshots
1. Click Top Right corner profile icon
2. Click Settings
3. Mid left click "SSH & GPG keys"
4. Click "New SSH key".
5. Goto GitHub docs and copy, paste the cmds to generate SSH key Just press Enter, Enter, Enter....
(In my case)
ssh-keygen -t ed25519 -C "pragadhesh@gmail.com"
Two (2) keys will be generated
Public
Private
Whenever we push the code on GitHub, already it will be checked that the SSH key on the local system matches with SSH key on GitHub. If it matches, we will be allowed to push.

PAGE NO.
DATE : / /

6. Now put the public key file on GitHub

In GitHub → Settings → SSH & GPG keys → New SSH key

Paste the data of public SSH key file on it

Finally execute → Screenshots [28]

```
ssh -T git@github.com
```

How try to authenticate → Enables SSH key successfully

Uploading an existing project on GitHub

Step 1 : Click [+] & select "Create New Repository"

Step 2 : Give the name, initialize with diff. options.

Step 3 : Now since we already have an existing git repo

∴ Cmds

NEXT PAGE

git remote rm origin ← to show prev origin

git remote add origin https://github.com/pragati111/portfolio_upload.git

git push origin master

Working with Remotes → Screenshots [30]

Remote : Something that controls anything from a different location

Adding more remotes (By def → "origin")

Cmd

git remote add my-remote <URL of Repo you want to push & fetch code from>

<remote-name>

Cloning a Github Repo → Screenshots [31]

Use → If we want to download a s/w on Github

→ Play with someone's code (Experiment)

→ Make contribu"

Cmd : git clone <URL of Repo to be cloned>

(64)

Social coding With Github

fetch changes from Github

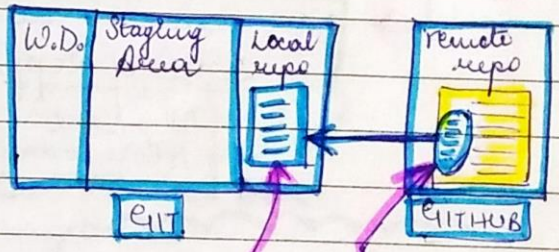
Git fetch and. download commits, files & refs from a remote repository into your local repo.

BUT

Doesn't bring those changes to the "working Directory"

CMD

git fetch



New changes by some other developer

New changes fetched by us

Those changes

- In case there are many branches & we want to fetch from master using origin remote → then use this cmd →

`git fetch origin master`

If from some other branch you want to fetch the code use,

`git fetch <remote-name> <branch-name>`

→ will get the changes in local repo & not working dir.

Now if you want this code to be in working directory → use this cmd → `git merge`

OR

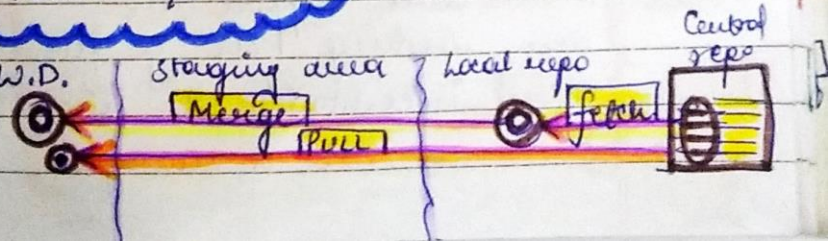
`git merge origin/master`

In a nutshell,

- **Fetch** → get code (change) from central repo to local repo.
- **Merge** → get this code from local repo to working area/repo.

Cmd to delete remote → `git remote remove <remote-name>`

Git Fetch v/s Pull



PAGE NO.:
DATE: / /

△ In a nutshell,

Git fetch + Git Merge = Git Pull

Git fetch: From central dir to the local repo

Git Merge: From local repo to working dir

Git Pull: Directory from central directory to the working area

Command to pull

Either simply write `git pull`
OR (specify anything)

`git pull <remote-name> <branch-name>`

NOTE: Fast-forward is the way of merging

The o/p of pull will show that it has first fetched & then did fast-forwarding

Con/Disadvantage of PULL:

Problm of merge conflicts → MERGE CONFLICT: Same file changed by two branches

How two suppose we are working on a file called index.html's 2nd line & same line is changed

try the above demo & push to central repo so on pull merge conflict will occur.

∴ Fetch + Merge is safer.

Forking A Repository

Usage → In case you want to make contribution to any code

OR

If you just want to play around with someone else's code

You (on github) have to fork that repository on your profile ((FORK ICON) present on TOP RIGHT)

Then this forked repo is to be cloned on local system using

`git clone <Repo-to-be-cloned URL>`

Now you can do modifications & push them to the central repo (forked on our profile)

PAGE NO.:
DATE: / /

Making a Pull Request

After forking & cloning the repo & making changes to it & finally pushing it to the central repo, we want to contribute this extra piece of code written by us to the place from where we forked this repo

↓

To do this

↓

We have to create a Pull Request

Step 1: Click [Contribute]

Step 2: Then click [Open Pull Request]

(It will be showing → "This branch is <no. of Commits> Commit ahead of <place from where code is forked>")

& (Open a pull request to contribute your changes upstream)

↑

{Place from where we forked the code}

Step 3: Finally give the comment & ~~commit name~~ title after clicking "Create a Pull Request".
(Also we can attach some files & folders if we want).

→ Finally this Pull Request will be opened & on the other side, proper whose code was forked, they on their profile will get an option to merge pull request

Merging a Pull Request

People who have the opportunity to merge the pull request also have an option to leave a comment for ~~people~~ person who has opened pull request to make some more changes or something else

If the ~~Organization~~ to whom changes to the code is made like if they can merge pull request

Now the person can check the comment & accordingly do changes & create new pull request

PAGE NO.:
DATE: / /

Github Issues

→ If the maintainers of the project want developers to contribute to the project because they know there is a bug in their project or they want some new feature

→ Then they can create an issue.

→ This will be the hint for the developers on where to contribute.

→ finally if contribution made (pull request opened by devs) is good then issue can be closed by the project maintainers.

Sync Git & Github repo with Upstream

Suppose you forked a project on your profile and now there exist some contributions made by some other developers to that project.

→ Now your

forked repository will be behind the upstream (means the main project)

This means we have to sync our forked repo with the main project repo i.e. upstream

To Do This → Click

Fetch upstream

Then click:

Fetch and merge

Now add the remote having this repo's URL

git remote add upstream <Repo-URL>

Now pull the code to local repo → execute this command

git pull upstream main

Now even your local repo is up-to-date ☑

(72)

HOSTING PAGES

PAGE NO.:
DATE: / /

Github Pages

Designed to host personal, organization, or project pages from Github repository.

1. Go to the project you want to host using Github pages feature.
2. Go to Settings.
3. Select Pages option from LHS. (left hand side)
4. Select the source
   - Branch: master
   - /(root)
   - Click Save
5. Finally, it will give us the website link saying,

Your site is ready to be published at https://xyz.github.io/project/ (page name)

User name

On clicking it you will be able to see your site running

NOTE: index.html is always considered as the first file by Github

* Also, we have an option to enter the custom domain if have bought eg: pragatiagarwal.com

Miscellaneous Git Tools

[Some cmds used for additional works]

1. Display the changes in commit

   Cmd → git show commitid

   [first 4-5 characters only needed]

2. Stashing

   [modified tracked files and staged changes]

   Takes the dirty state of working directory & saves it on a stack of unfinished changes that you can reapply anytime

When it is used?

If suppose you were working on some file and in b/w you

PAGE NO.:
DATE: / /

get the call from manager to make some changes to some other file → BUT

You don't want to commit the current changes & also without losing the current work want to switch to other branch

SOLUTION

Stashing

cmd → git stash , git stash apply , git stash save

Just when you want to switch to other branch without committing changes to this current branch use this cmd

Apply stashing

Getting the file back (from stack) to which we applied stashing

Step 01 : Execute

git stash list

We can see the complete list of stashes we have made

→ INTERNAL WORKING

uses git diff internally and checks for all the differences made & finally stores those changes in the STACK

stash.0 → STACK

Step 02 : Taking back the stashes to the working area

Cmd → git stash apply stash@{0}

Comes on its own

Our original cmd

git stash apply stash@{0}

Now we can again continue working from where we left & the things & when content with the changes, commit them

git add .
git commit -m "cool feature"

PAGE NO.:
DATE: / /

Finally we have taken back the stash, worked on it & committed the changes → Then no more we need the stash

∴ we should delete it

Cmd to delete the stash

→ git stash drop stash@{0}

check which one you want to delete by first listing all the stashes using the cmd.

git stash list

Cleaning your working directory

You may not want to stash some work or files in your working directory

↓

BUT simply get rid of them

To do this: Cmnd

git clean -f -d

-f: forcefully deleting

-d: for all the subdirectories

Helps us make back to the prev commit

Used to delete all the changes

NOTE: In case you have added the file to the staging area & you want to clean it, use restore cmnd to bring it back to the working directory then use this clean cmnd.

Changing Commit messages

Changing the last commit

git commit --amend

will open the vim text editor

write the commit msg & some comments

NOTE: press esc key then type :wq, press enter to save the changes made.

To make changes → Press i

PAGE NO.:
DATE: / /

→ Changes made to the last commit msg can be seen using `git log --oneline` cmd

Changing commits content

Changing the actual content of the last commit :

- Make changes in the working directory.
- Add files to the staging area.
- Rerun the same command.

NOTE : Command `git commit --amend` ~~`git log --oneline`~~ will stay unchanged (same cmd will be used)

Means commit (wrong-commit) is already done but we will make more changes (maybe some undo work), then will add these changes to staging area

In this final step instead of writing `commit -m` cmd, use `git commit --amend`

→ It will ask using `vim` editor to whether make changes to the commit msg or not.

→ Finally make the desired changes to the commit msg → Press `esc` key,
→ Type `:wq` and press enter.
Save modifications

OR

Type `:q!` and press enter.
Doesn't save modifications

Rewriting History

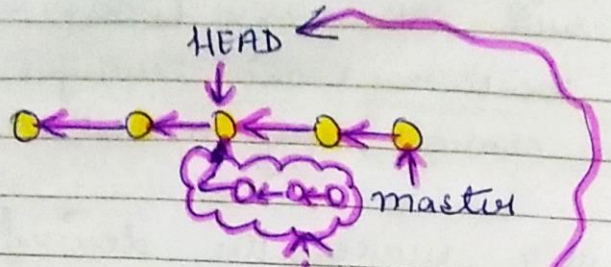
1. Traveling to past commits

→ Checkout commit
→ `cmd` : `git checkout commitid`

Visual explanation
Detached the head to the prev commitid specified
HEAD
HEAD
master
∴ we are no more in master branch

To whichever commit we want to go we can specify its commitid here

PAGE NO.
DATE: / /



Now from this commit, we can create a branch & do modifications like adding features

BUT

If you want to undo this operation → Use this CMD

git switch -

(2) Making new branches from old commits

→ To list all the commits
  → git log --oneline

→ Go to the commit you want
  → git checkout <commitid>

Make the desired changes to the code.

→ Check the status
  → git status
  (will show dirty)

→ Add & commit the changes

git add .
git commit -m "<msg>"

You can make as many commits as you want

Now, suppose at this moment we want to again move back to master branch → Here if we use

git switch -

and

To create a new branch CMD

git switch -c branchname

Git will warn us that if we don't save these new commits & goto master it will delete them

Therefore

(3) Reverting Commits

You can undo the changes of any commit that you want

CMD → git revert commitid

DANGEROUS COMMAND!

PAGE NO.
DATE : / /

→ Vim editor will appear which will ask for commit msg → press `i`, Make changes, Press `esc`, Type `:wq`, Press `enter`. DONE!!

↳ OKAY, Now we used `git revert commitid` as the cmd

Now if suppose we check the logs we will find that new commit is created, but we only want to undo the commit & not commit the undo-changes made i.e.

→ To explicitly commit the revert changes → For this there is a command

`git revert -n commitid`

→ To undo the revert done

cmd → `git revert --abort`

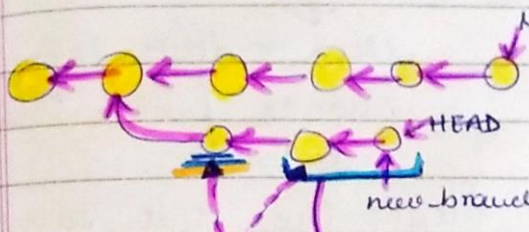
Most Dangerous Command of Git (83)

a) Git Reset Command

- delete all the commits from the given commit id onwards

i) `git reset --soft commitid`

ii) `git reset --hard commitid`



Suppose we want these commits to be deleted & this to be the last commit of this branch

To do this we will take the commitid of this commit & use the cmd → `git reset --soft commitid`

`--soft` → we still have an option to undo the commit deletion made

`--hard` → no option for undoing deletions

by-default cmd if not used explicitly

DATE: / /

How to confirm the deletion of commits → use → `git clean -f -d`

5) Deleting commits from GitHub

Suppose we made some silly commit & pushed that on GitHub

How to delete this change from GitHub

Step 1: Use that same prev. command

`git reset --hard Commited`

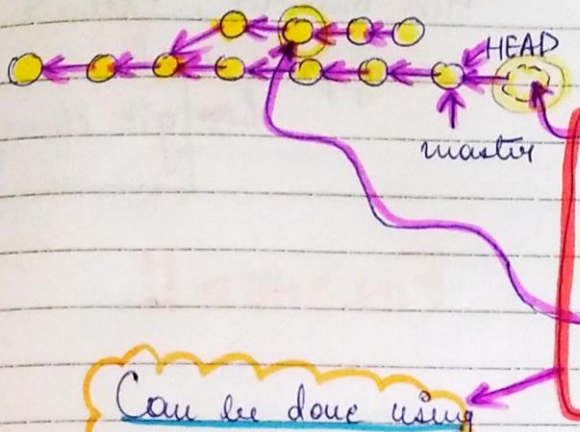
Step 2: Now push this change on GitHub as well

`git push -f origin master`

6) Cherry-picking commits

Pick the changes from a specific commit from any branch & merge with another branch.

Cmd → git cherry-pick commitid



Step 01: Go to the branch from where you want to pick the commit.
→ git checkout branchname

Step 02:
Now see the logs of this branch → git log --oneline

To see the changes in any commit → git show commitid

Step 03: Go back to the master branch when you are done deciding the commit to be picked → git checkout master

Step 04: Now in the master (since we want the changes in this branch), execute the cherry-pick cmd

↳ `git cherry-pick <commit-id>`

FINISHED!!