Git

→ Version Control:-
Like in word-dox when we write something and got to know there is spalling mistake or something we simply undo that change so, Basically ∙ do Version control
when we do development we need version control to help in development.
version control is a tool that track and manage all the changes in system.
Git is a version control tool.
Git vs Github.
Git is a version control tool or simply a software which work to control your changes with some additional capability.
Github :- Github is a servies or hosting servies.
Github is a Git repository hosting servies.
Repository:- Repository is a folder which has capability Git also track.

Git, Github, Github Desktop setup.
Go to git-scm.com and download Git After Installing Git go to cmd.

```
git --version → check Git version.
```

Go to Github.com and creak an account
Then again open cmd.

```
git config --global user.name "RituChoudhary01"
git config --global user.email "ritu9806541427@gmail.com"
```

To verify username and user email.

```
git config --list
```

Download Github Desktop from desktop.github.com and config it with Github credentials.

Install Git for mac.
First install Homebrew go on brew.sh and copy install Homebrew command and paste on terminal.
After install Homebrew Run command brew install git
git --version to check Git version.

Creating a Git Repository:-
make a new folder on desktop "dot_dummy-repo"
Now this is an empty folder.
open terminal
pwd - present working directory.
cd Desktop:- change directory to Desktop.
cd dot_dummy_repo → change directory to dot_dummy-repo
ls → list all the file or directory present in folder.
It is a folder not a repo to check we run
git status → tell about current status of repo.
git init → convert normal folder into Git Repo.
initialized empty Git repository in user/ritu/Desktop/dot_dummy_repo/.git/
.git is new folder created. This folder is hidden By default.
(To watch hidden folder in a parent folder to run Command + shift + period(.))
ls -al → list all files including hidden
you can see 3 types of file.
(i) . (Represent current directory dot_dummy-repo)
(ii) .. (Represent parent Dir of curr dir: Desktop)
(iii) .git (Meta Data of Git Repo).

Parent (..) Desktop

. → ls -al
dot_dummy_Repo
(curr directory)

Now git Repo. is created. use git status to check / verify.
Git knows everything you do with this repo.
create a txt file in dot_dummy_repo and run git status again on cmd.
Now you can see there is a untracked file and to include it use git add test.txt.
How to clone someone else Repo from Github and Collaborate.
To clone a Repo you need URL of that Git Repo.
Go to their Git Repo webpage on Github click Code then HTTPS then copy the URL
The URL you see on browser is same as it.
But the only Diff is Repo URL is ended with .git extension.
After copying the URL run a command
~~you must~~

```
cd.. →
git clone URLHere NameofNewFolder.
```

In this folder you can see all the files of cloned repo.

```
cd ClonedRepoName:
```

Now you can run any git cmd on this Repo. like git status.
Now change the content of any file inside clone Repo folder. Then again check git status. Git will automatically track the New

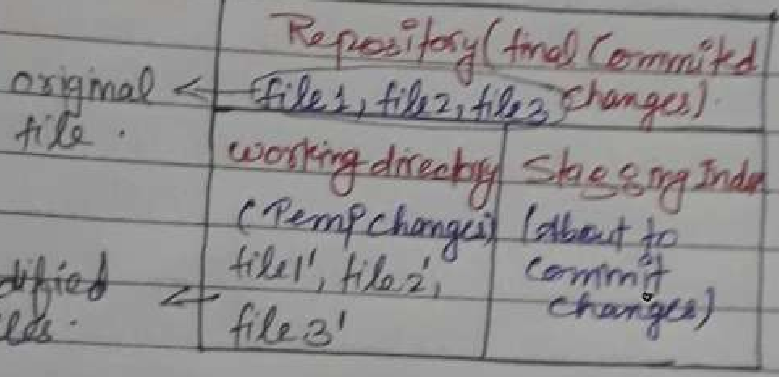
change and already what file is created/deleted/modified.

git diff - To see what changes exactly has been made.

Life cycle of a change.

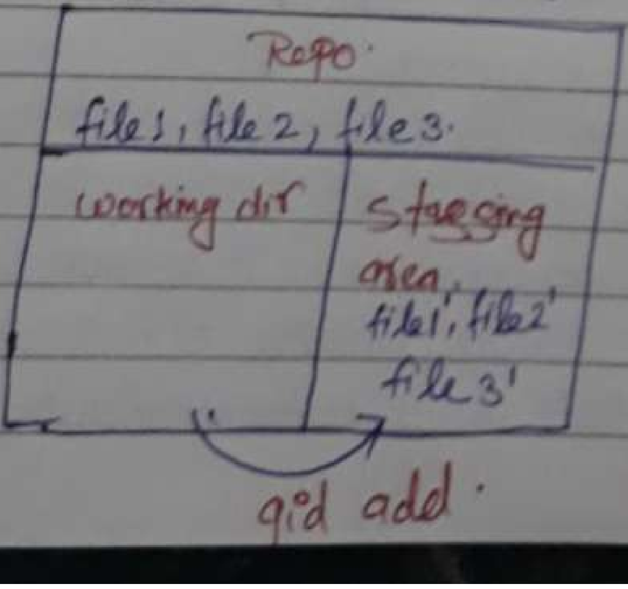
We in our folder dot-dummy repo. suppose it has many files.

(original files)
① file1.js → file1'.js
② file2.js → file2'.js
③ file3.js → file3'.js
(modified files)

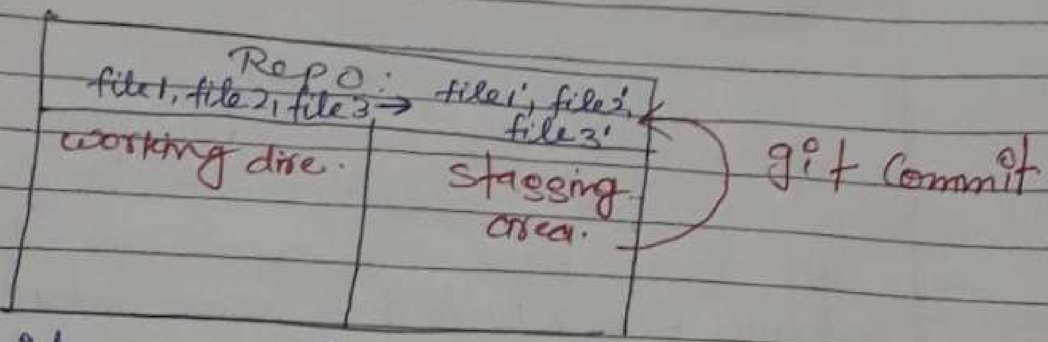


Repository (final commited changes)
file1, file2, file3
original file.
working directory (Temp changes) file1', file2', file3'
modified files.
Staging Index (about to commit changes)

Suppose you do some changes in the original files. All the modified files consider to be in working Direc. (temp changes) without affecting the original Repo. To save these changes in the original Repo you have to first use git add cmd to move the modified files from working dire to staging area.

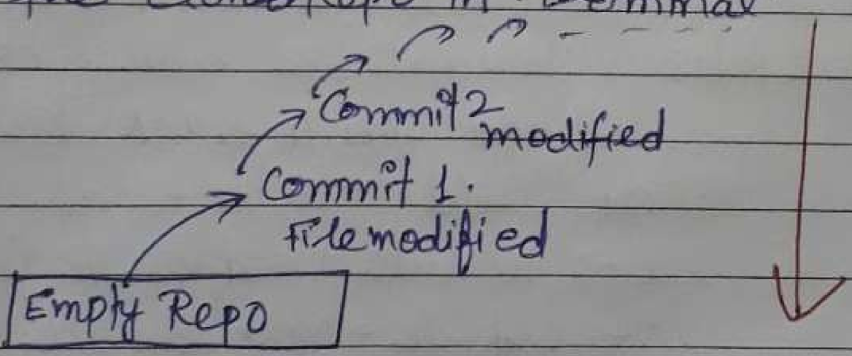


After staging the files we use git commit command to permanently save these changes made in files.



Commit id are unique code that are created whenever a new commit is Recorded.
file1.js → Commit Id (ABC)
file2.js → Commit Id (ABCD).

Revewing a Git Repo History.
Commit:- is a parmant change saved by git.
Commit:- is a save point. after that whatever. Change you do git will track it differently.
open the clonedRepo in Terminal.



git log:- To see all the Commits from top (Latest) to bottom (oldest)
· Press (q) for exit out of log.
we can Also see all the commits in Github.
git Log -3 → show the the latest (n) commits. now n = 3.
git log -p → Along with showing all the Details of all commit. it also shows what

Changes has been made in each commit.
Press (q) for exit.
You can also see the changes made in each Commit by GitHub UI.
git log --oneline → show you only the SHA ID and commit msg of each commit.
git log --stat → It shows all the commits Details along with it shows what files are changed.
git show SHA ID (commit) → It shows what changes has been made in a particular Commit.

Let's make a Commit:
First make a empty Folder on Desktop.
cd Desktop.
mkdir myrepo → make a new folder myrepo.
ls
git init
Add a new file (main.cpp) in new Repo folder then use
git status → you will see a suggestion that use.
git add cmd to track this file
git add → Add file to track.
if → newfile → add file to Track
→ old file → Move the file to staging index.
git add main.cpp
git commit -m "init demo.txt"
→ Commit msg.
git status → There is Nothing to Commit.
git log → you'll the latest Commit / one commit
git show SHA ID → To see the changes made in files during this Commit.

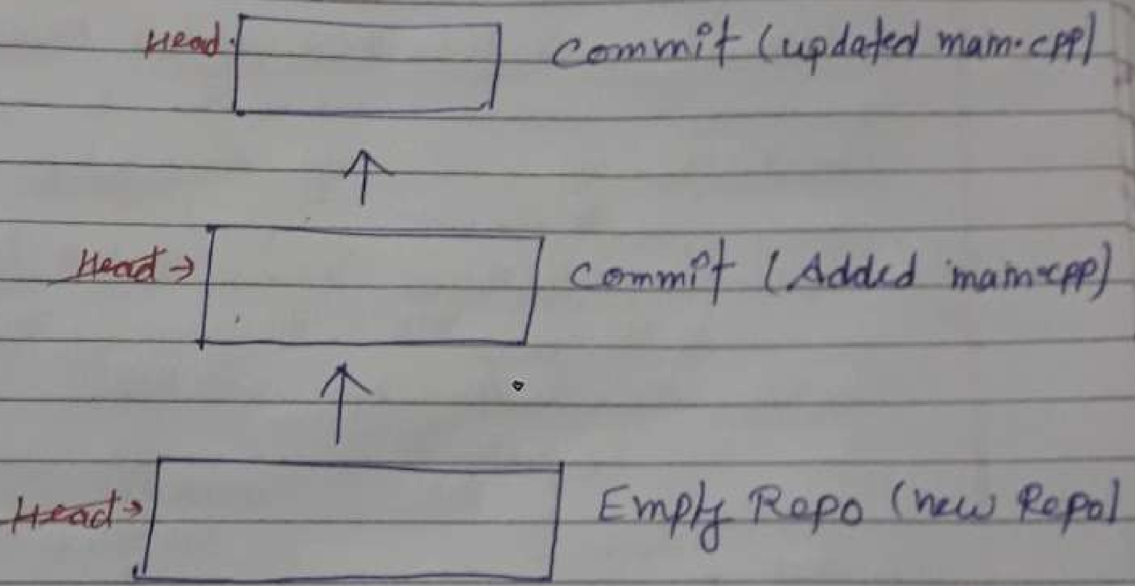
Now modify the already created main.cpp file then use.

git status – main.cpp file has been modified.

git add. → Move all the Modified files to staging area. But this cmd is Not Recomended.

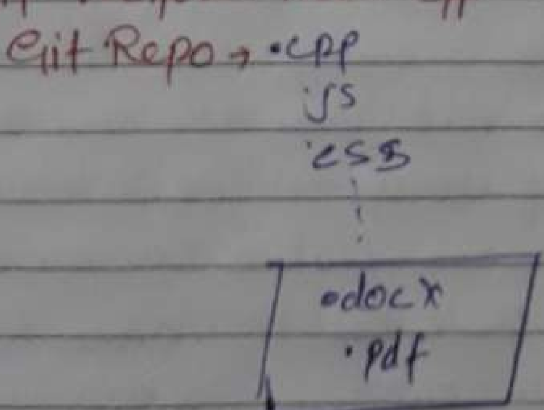
git commit -m "updated main.cpp"

git log → Now you'll see 2 commit and Head is moved to latest commit.



If you mistakely do any changes in main.cpp file but not want these changes to be commited and went the original file back you can discard the changes in working dir.

git restore main.cpp.



If you want these files Not to be tracked by Git.

Create a file named .gitignore.

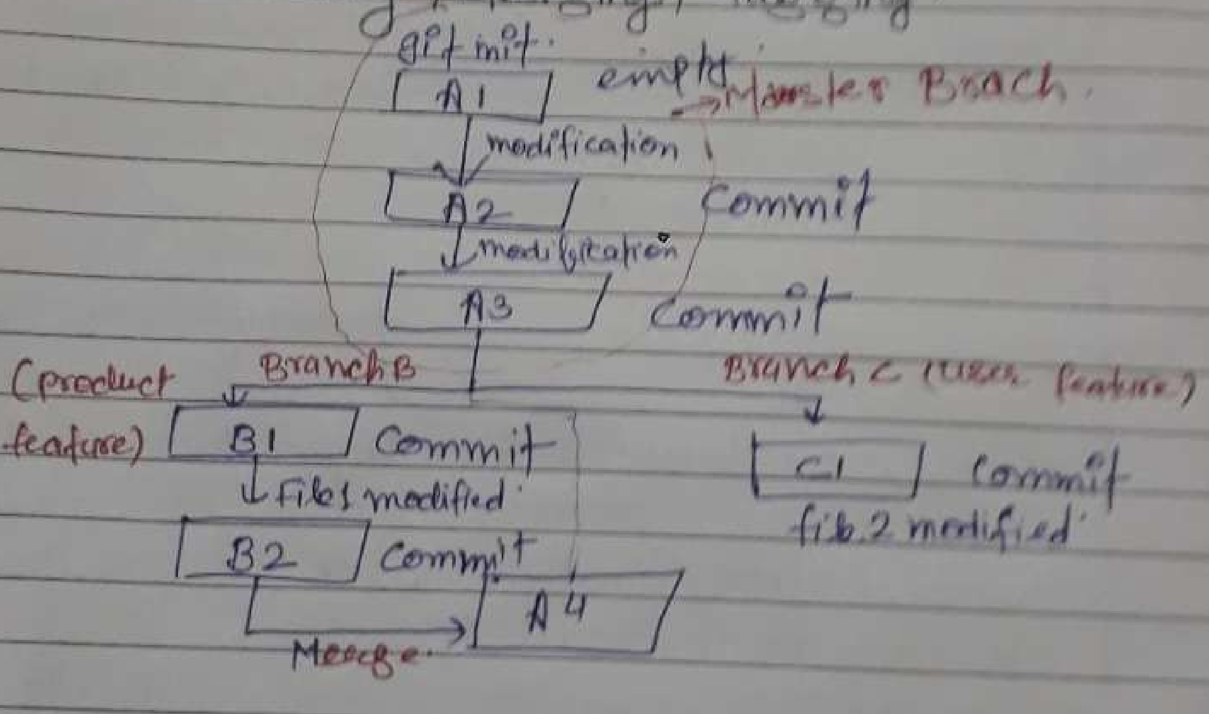
In this file you write the file names that git

should not be tracked.
eg.:-
*.docx → all docs files.
demo.txt
assets/images/*.png
But you must commit .gitignore file.

```
git add .gitignore
git commit -m "adding git ignore file"
git log
```

Now Head will be move to this latest commit.

Branching, Merging, Tagging.



Suppose you initialize a empty Repo (A1) and by doing modification and commits you are now on (A3) Commit. Now you want 1 of your friends to work on Project and add some feature so, you made 2 new Branches (B&C) Both the Branches has the same code of commit (A3). Now you working on file 1 from B branch and your friend is working on file 2 from (C) branch. whatever changes they are doing with the

file & other branchs code remain untouched.
Now let's say from (B) branch developed feature then merge his code to the main branch.
Open terminal.
→ cd Desktop/myrepo.
git status.
git log
git restore .gitignore
git status. (you are now on Main Branch].
git branch → show all your branches including master it also shows where Head is Pointing.
git branch branchName → creat a New branch.
git branch quick sort
git branch. Now on maste branch.
git checkout quicksort → switch to quicksort Branch.
git branch. Now on quicksort Branch.
git log.
Do some changes in a file main.cpp (add quicksort) then use.
git Status.
git add main.cpp
git commit -m "quickSort code added"
git log
git checkout master
Now you are on main Branch & will not see quicksort code in the file bcz that code is only related to quickSort Branch.
Now creat a New Branch & move to it.
git checkout -b bubblesort
git branch Now on bubblesort
git log.

```
git branch bubble Sort
git checkout bubble Sort
```

Instead of using these 2 commands separately you can use only one command to create a new Branch and also Move Head to it.

~~git Checkout -b~~

```
git checkout -b bubblesort
```

Now modify main.cpp file.

```
git add main.cpp.
git commit -m "bubblesort code added"
```

Instead of adding / staging the file and commiting it seperately you can do this task by using one command only.

```
git commit -am "bubble Sort code Added"
```

Now Both the features are developed. It's time to Merge the branch into main branch.

Let's merge first bubbleSort branch changes on Master Branch.

```
git checkout maste.
git merge bubblesort
git log
```

=> All the changes / commits of bubble sort branch will be added to master Branch.

Now bubblesort Branch is of No use, so we can delete it by

```
git branch -d bubblesort
git branch.
```

Now it's time to Merge Quick Sort branch.

```
git checkout maste.
git merge quickSort.
```

you'll see an error Merge conflict.

bcz on the same no. of files there is some code of master Branch and of quicksort Branch Also git status.

you've 2 options Now

① Fix the conflick by editing and adding codes of both files together, then commit or

② use git merge --abort to abort the Merge.

But we want the code of quick sort also, so we'll resolve the conflict manually, then use.

we editing code. | git commit -am "quick"

```
git add main.cpp
git stakes.
git log.
```

Tagings ⇒ Tag a particular commit or SHAID

```
git tag -a betaVersion SHAId -m "my beta Version"
```

git log → you can see tag: betaVersion on the specified commit.

Suppose you added more lines in index.js and added merge sort then commit it.

```
git commit -am "Added Merge sort"
```

git log → you can see Head is pointing to the latest commit you did now and tag is still pointing to the Last second commit.

Now you was not the latest commit to tag so, first delete the previous Tag.

```
git tag -d betaVersion
```

tag Name.

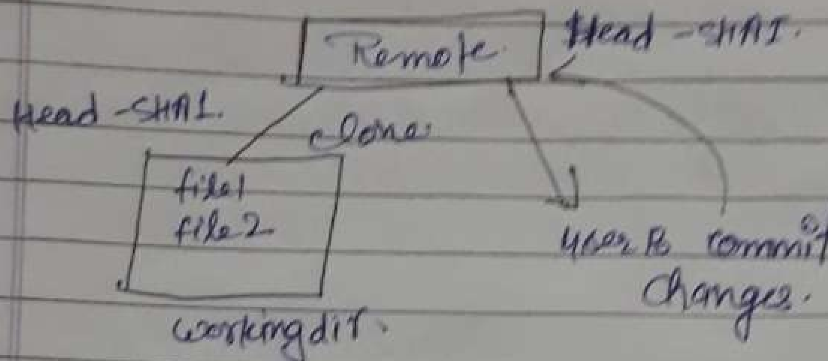
git log. (Delete tag Donot show)

Git Stash Suppose you cloned a Repo from Remote server and modified file1, file2 in your working Dir.

But before you commit your code. someone else commit their code to remote Now remote's

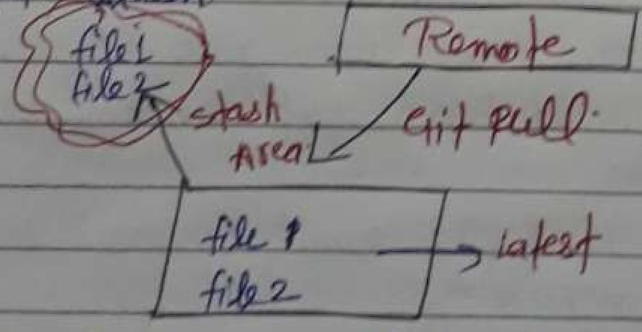
SHA1 D is one commit a head of your commit.

If you do git pull used to updak the local repo From Remote repo.



There might be a problem bcz if someone who (user B) changes file1, file2 also

your files are old and you're fetching and updating files from remote to update your local files But in this case your changes will be gone If you want your changes also along with latest code you can use git stash.



It will copy your files of working dir and paste it to stash area and Empty your working Dir you can pull the latest files from Remote now

Now you'll use git stash apply to call your files back from stashing Area.

```
git pull
git stash
git stash list → To see stash Area.
git pull
```

git stash apply. → unstashing your files.
Again Merge conflicts.
Resolve the conflict Manually
git commit -am "correct value of"
git log.
git push -u origin master.
origin master is Base root of Repo.

Undo commits:-

commit :- means final change but we can also undo this if needed.
Git Commit ~~-~~ --amed :- amend the most recent Commit.
Git revert → Revert given commit.
Git reset → Delete commit (Dangerous command)
suppose you did some mistake in your code and accidently commit it what to do?
you can revert your commit.
git revert SHAID
tells you what commit you are reverting and. opens editor mode.
Press ESC to switch into Command Mode.
:wq → to save and exit
git log → you can see we reverted the wrong commit Now we have code without any mistake.
git reset --soft SHAID.
of correct code before any mistake was done.
git log - Now you can see the wrong commit has. been deleted and will not even seen in log. And Head moved to the previous Correct commit.

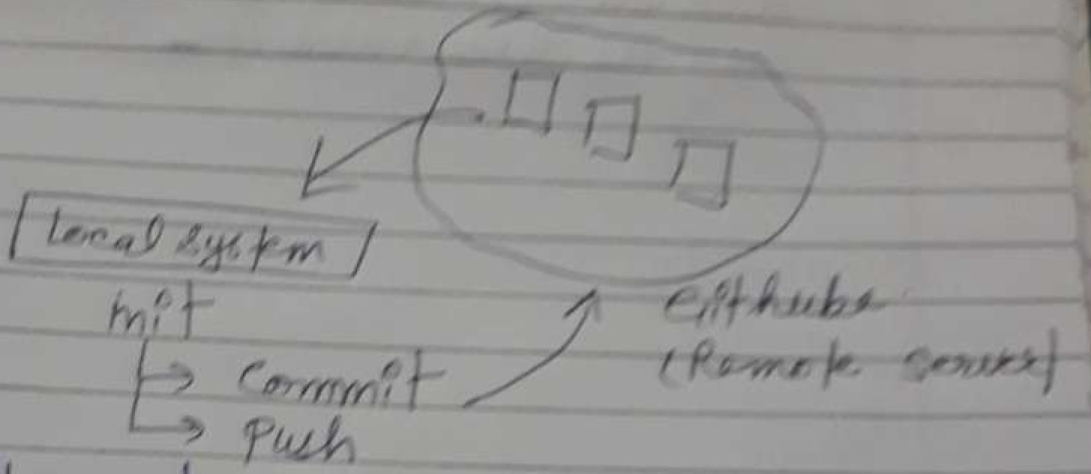
suppose there are some changes in working Dir. before using git reset.
If you would have choosen --hard It will Discard the local changes also working Dir will get empty.
In case of --mixed.
The changes made in working Dir would still remain & the changes seen as modification means you have to stage it before commit.

In case of --soft.
The changes made in working Dir. would still remain & It automatically staged your changes. No need to use git add before commit.

--soft changes will show as staged.
-- mixed change will show modification.
--hard. change will be deleted
we mostly use --soft, --mixed.
git revert → History is maintained and reverted commit can be seen using git log.
git reset → Just move the Head and Delete the above commits and you have to take care of your working Dir. changes also by using , soft, mixed or Hard.
git commit --amed. we can only change the description of the most recent commit only.
Suppose you added insertion-sort code in main.cpp file then commit it with a message "counting sort Added"
But the commit msg should be "insertion sort added" so, Now we amend it to

change commit msg:
git commit --amend : opens editor mode for you to change the Description
After changing commit Desc. press esc, write :wq git by.

How to push a commit & Git GUI.



① First create and empty Repo on Github
② Make a directory on your local system
③ git init
④ Add a REAME.md File in the local dir and Commit it.
git commit -am "first commit"
⑤ git config --global user.name "—"
git config --global user.mail "—"
git config --list

⑥ gid remote add origin url
Mapping local Branch to Remote Branch.
⑦ git push -u origin master → Branch Name.
Push local Dir commits to Remote server.
If it asked your username & password befo
pushing.

Username: Enter your username
Password: for password go to Github setting >
Developer setting > Personal access tokens >
Tokens (classic) > Generate New token >
Gen New Token (classic)
Note - access token
Expiration → No expiration
Tick all checkboxs > Generate Token
Copy the token & paste it