

## SRE day -4 (my sql and git)

Create and populate source table, or generating insert statements using select statements.

```
-- Create and populate source table
CREATE TABLE source_employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10,2),
    department VARCHAR(50)
);

INSERT INTO source_employees VALUES
(1, 'John Smith', 50000, 'IT'),
(2, 'Sarah Johnson', 60000, 'HR'),
(3, 'Michael Brown', 55000, 'IT');
```

```
-- Generate INSERT statements from SELECT
/* here we are generating queries from source table and creating all these queries and then copying and pasting them to directly run
and we use 'quote keyword' to insert quote in front of our string character for our strings in our queries*/
SELECT CONCAT(
    'INSERT INTO employees (emp_id, name, salary, department) VALUES (',
    emp_id, ', ',
    QUOTE(name), ', ',
    salary, ', ',
    QUOTE(department), ');'
) AS insert_statement
FROM source_employees;
```

Using on delete cascade and on update cascade and on delete null, so as to handle problems like when we delete something from parent table then it automatically deletes value for the child table so as to avoid inconsistency and we define all these command when defining structure for our child tables

```
Query 1 x
Limit to 1000 rows

445 create table courses(
446     course_id int primary key,
447     course_name varchar(100),
448     credits int);
449
```

```
create table students(student_id int primary key,
    first_name varchar(50),last_name varchar(50),email varchar(100));
```

```

create table enrollments(
  enrollment_id int primary key,
  student_id int,
  course_id int,
  enrollment_date date,
  foreign key(student_id) references students(student_id),
  foreign key(course_id) references courses(course_id)
  -- on update cascade
  on delete set null
  -- on delete cascade
);

insert into students values(1,'uakdhksa','sadgashgdyudwq','asdasgd@email.com');
insert into students values(2,'2uakdhksa','sadgashgdyudwq','asdasgd@email.com');
insert into students values(3,'3uakdhksa','sadgashgdyudwq','asdasgd@email.com');
insert into students values(4,'4uakdhksa','sadgashgdyudwq','asdasgd@email.com');

```

```

-- This works fine
INSERT INTO courses VALUES (1, 'Mathematics', 3);
#INSERT INTO students VALUES (1, 'John', 'Doe', 'john@email.com');
INSERT INTO enrollments VALUES (1, 1, 1, '2025-02-12');

select * from courses;
select * from enrollments;

update courses set course_name='History' , course_id=5 where course_id=2;
delete from courses where course_id=5;

```

## Using triggers to do logging based on some action:

```

create table enrollments_logging(
  enrollment_id int primary key,
  action varchar(200),
  change_date date);
-- trigger is based on an action that we perform on table
create trigger before_course_deletion
before delete on COURSES -- before operation_name on table_name
for each row
insert into enrollments_logging(enrollment_id, action, change_date)
values(old.COURSE_ID, 'DELETE BASED ON DELETION FROM COURSE AND STUDENT', NOW());

select * from enrollments_logging;

```

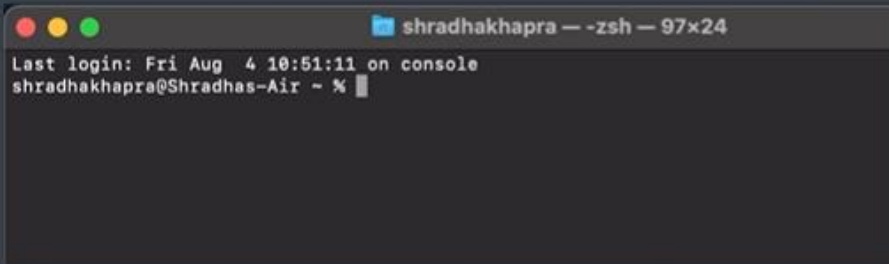
```

SELECT
  e.name,
  COALESCE(d.department_name, 'No Department') as department, -- we use coalesce when we need to replace a null value with some message
  e.salary,
  CASE
    WHEN d.department_id IS NULL THEN 'Unassigned'
    WHEN e.salary > 55000 THEN 'High Paid'
    ELSE 'Standard Pay'
  END as status
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;

```

# The Terminal

A text input and output environment.



A terminal is a text based input and output environment.

Terminal is basically used to install packages in node js and it also helps in running various commands in git and also in our programming.

And as terminal directly interact with the systems, therefore it also provides us with access to files, which are only visible to developers in the system.

Some terms related to terminal are:

## Different Terms

- **Command Line** - any interface that is used by entering textual commands (gen. **Windows** centric)
- **Terminal** - This is a type of command line (gen. **Mac** centric)
- **Console** - A command-line interface used to work with your computer
- **Shell** - A program running on terminal
- **Bash** - A popular shell on Mac OS/ Linux
- **Z-Shell** - Another shell (default)

Command line is an interface where user can enter textual command, and terminal is a type of command line, which performs some action when we enter some command.

And shell refers to any program that runs on terminal.

All window users need to install "**git bash**". As it helps to run all the git commands and as well as all the commands related to terminal.

## Basics

**ls** = list files  
(show my files)

**pwd** = print working directory  
(where am i?)

**clear** = clear screen

## Navigation

### Inside & Outside Directories

**cd** = change directory

*cd Desktop*

*cd .. = back button*

**cd+ space+ '..'**

For autocompleting commands we can press tab in our terminal. And if we want to run any previous command again that we recently used, then we can use the up and down arrow keys for traversing recently used commands.

And if we want to move two or more folders in one command then we can mention path in the cd command:

```
shradhakhapra@Shradhas-MacBook-Air ~ % pwd
/Users/shradhakhapra
shradhakhapra@Shradhas-MacBook-Air ~ % cd Desktop/Delta
```

And similarly if we want to move two directories back:

```
shradhakhapra@Shradhas-MacBook-Air Delta % pwd
/Users/shradhakhapra/Desktop/Delta
shradhakhapra@Shradhas-MacBook-Air Delta % cd ../../
```

# Paths

## Absolute & Relative

`cd Desktop/Delta` (relative path)  current location

`cd /Users/shradhakhapra/Desktop` (absolute path)

`/` = root directory

`~` = home directory

The relative path refers to the path from our current location, that is from the current working directory we are on. For relative path we should know all the files and folders within the directory and we should also know that our target file is in which folder.

Whereas the absolute path refers to the exact location of the file (or folder) we want to go. and we can access it from anywhere. As the path for a file always remain same.

The relative path is written as it is, whereas the absolute path is started with an forward slash “/”.

We can give “absolute path” from any position and it would take us there, that is even if we are within some file of our target folder.

```
shradhakhapra@Shradhas-MacBook-Air ~ % cd Desktop/Delta
shradhakhapra@Shradhas-MacBook-Air Delta % ls
CSS          JS          README.md
shradhakhapra@Shradhas-MacBook-Air Delta % cd CSS
shradhakhapra@Shradhas-MacBook-Air CSS % cd /Users/shradhakhapra/Desktop/Delta
```

But if we run the relative path here, it would say file doesn't exist. As it checks only the current working directory for that file or folder.

```
shradhakhapra@Shradhas-MacBook-Air Delta % cd CSS
shradhakhapra@Shradhas-MacBook-Air CSS % cd Desktop/Delta
cd: no such file or directory: Desktop/Delta
shradhakhapra@Shradhas-MacBook-Air CSS %
```

Similarly if we want to enter our root directory, which contains all the details related to the system, even the user login etc.

```
Macintosh HD -- zsh -- 66x25
shradhakhapra@Shradhas-MacBook-Air CSS % cd /
shradhakhapra@Shradhas-MacBook-Air / %
```

And similarly we can go to home directory which tells us the current working directory, and to verify it we can “print current working directory”

```
shradhakhapra@Shradhas-MacBook-Air / % cd ~
shradhakhapra@Shradhas-MacBook-Air ~ % pwd
/Users/shradhakhapra
shradhakhapra@Shradhas-MacBook-Air ~ %
```

## Making Directories

**mkdir** = make directory

This command is used to create a folder. But before creating a folder we would change our directory to the folder where we want to create a new directory.

```
Delta -- zsh -- 71x12
shradhakhapra@Shradhas-MacBook-Air ~ % cd Desktop/Delta
shradhakhapra@Shradhas-MacBook-Air Delta % mkdir NewDir
```

And then we simply write mkdir and mention the folder name we want to give to our newly created folder.

**Note:** we can't create a folder with name same as already created folder.

```
NewDir -- zsh -- 71x12
shradhakhapra@Shradhas-MacBook-Air NewDir % ls
Hello
shradhakhapra@Shradhas-MacBook-Air NewDir % mkdir Hello
mkdir: Hello: File exists
shradhakhapra@Shradhas-MacBook-Air NewDir %
```



We can also create a folder using relative and absolute path if we want to create a file at some specific location:

```
shradhakhapra@Shradhas-MacBook-Air NewDir % ls
Hello
shradhakhapra@Shradhas-MacBook-Air NewDir % mkdir Hello
mkdir: Hello: File exists
shradhakhapra@Shradhas-MacBook-Air NewDir % cd ../../
shradhakhapra@Shradhas-MacBook-Air Desktop % mkdir Delta/NewDir2/Hello2

shradhakhapra@Shradhas-MacBook-Air Desktop % █

shradhakhapra@Shradhas-MacBook-Air Desktop % mkdir /User/shradhakhapra/Desktop/Delta/NewDir2/Hello2/abc
```

The above relative path command would create hello2 folder within newdir2. And in our absolute path it would create a folder named abc, within hello2 folder.

## Flags

Flags are characters that we pass with commands to modify their behaviour

**manual Command**

- man ls** - give info about ls command
- man mkdir** - give info about mkdir command

**with Flags**

- ls -l** It prints all the additional information also with the file names.
- ls -a** It prints all the files which start with '.' (NOTE: hidden files start with '.')
- ls -la** It is combination of both above flags, it prints all files even hidden with extra info

The "man command" is used to open manual or information for the specified command.

**Flags refers to some character which are passed with some commands to pass some external information, that modifies the behaviour of that command.**

And when we run man command for some command it also tells us about all the flags that we can use with that command.

**And to exit manual mode, we press q** (and with it we directly quit the manual mode.)

## Touch Command

Used to create files

```
touch index.html
```

```
touch app.js
```

```
touch abc.txt
```

The touch command is used to create a new file in our directory, and if we write a touch command with some type of extension then a new file of that specific extension is created.

```
Hello — zsh — 63x15
shradhakhapra@Shradhas-MacBook-Air Hello % pwd
/Users/shradhakhapra/Desktop/Delta/NewDir/Hello
shradhakhapra@Shradhas-MacBook-Air Hello % touch index.html
shradhakhapra@Shradhas-MacBook-Air Hello % touch style.css
shradhakhapra@Shradhas-MacBook-Air Hello % touch app.js
shradhakhapra@Shradhas-MacBook-Air Hello %
```

## Deleting Files & Folders

**rm** - removes files

It helps to remove files from our directory.

**rmdir** - removes empty folders

It helps to remove only empty directories. And to delete a folder we should not be in the target folder.

**rm -rf** - removes any folders

It helps to remove any folder or directories which may contain some files or folders, and here in "rf" flags, 'r' stands for "recursive" and 'f' stands for "forced" delete.

As name suggests this command is used to remove folder.

```
shradhakhapra@Shradhas-MacBook-Air Hello % rm app.js style.css
index.html
shradhakhapra@Shradhas-MacBook-Air Hello %
```

The caution should be taken while using the rm command as once a file is removed it can't be retrieved.



## What is Git?

Free & Open Source **Version Control System**

tools that help to track changes in code

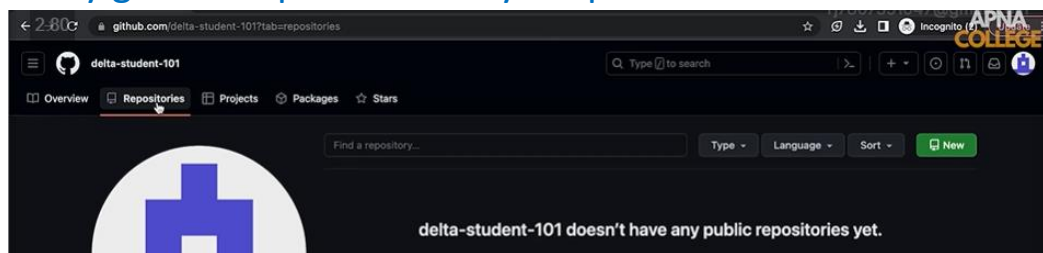
## What is Github?

Website where we host **repositories online**

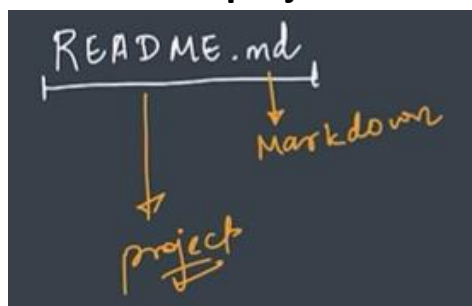
Now after we have created an account on github, now we would learn how to create repositories, so that we can upload our projects on github:

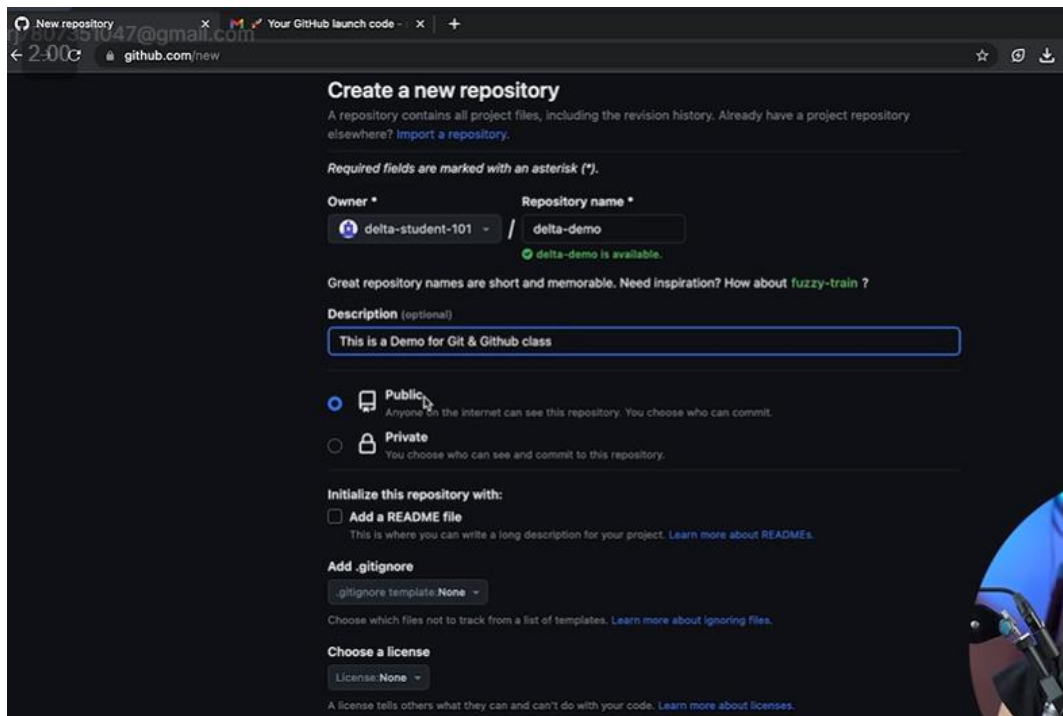
- Create a new repository : delta-demo

- Firstly go into repositories in your profile and click on new .



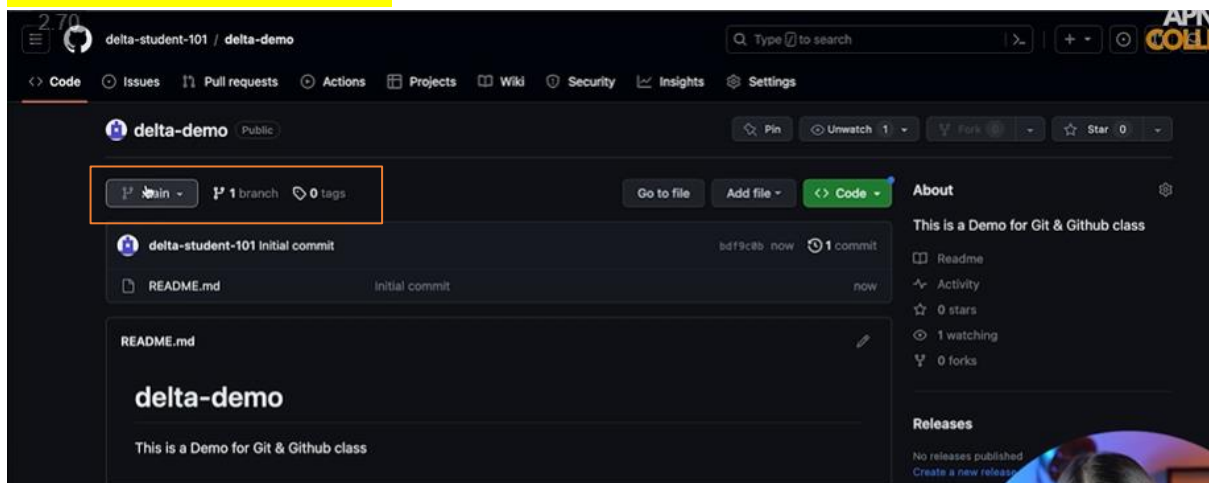
- Then we can give repository name and also select if we want it to be public or private, we generally keep it public so that recruiters and HR's can verify it .
- Then we would see a checkbox saying to create a README.md file, here md stands for mark-down. The README.md file tells us about the project details, like what is project name , what is its purpose etc. **we generally write all the details that we want to viewer to know about our project here.**





- Then we simply click on create repository.

Now when we got to our profile there we can open our repository, And it would look like



and here we can see three headings like main, branches, tags. And then we have a “readme.file” and there is a pencil symbol and we click on it to make any changes in our readme file. and then the data of “readme.md” file is also given here.

And if we notice we would see commit written with the readme.md, in git the changes occur in two steps, firstly we make changes using “add” keyword and secondly to make that changes permanent, we need to commit them using “commit” keyword.

## Using Git

- Command Line (Most Popular)
- IDE / Code Editors (like VSCode)
- Graphical User Interface (like GitKraken)

```
shradhakhapra@Shradhas-MacBook-Air ~ % git --version  
git version 2.24.3 (Apple Git-128)
```

- Now using github is a good practice but as a developer we tend to use git with our terminal.
- And another way to use is with VS code where we have some extensions which helps us to use the git commands.
- And we also have some GUI's like gitkraken which helps us to use git.

**but we prefer the command line using VS code method the most, as it has no limitation and we can fully access the git using command line, whereas when using with GUI's there may come a point where some commands don't work.**

## Configuring Git

```
git config --global user.name "My Name"
```

```
git config --global user.email "someone@email.com"
```

Here we are globally(so that git knows throughout the project all changes are made for which user and which id, if we want to work on a specific project then we can use "local" as well).

**Now we would open a folder in VS code, and in the bottom left corner we have some symbols like cross and triangle after clicking on it, we would get a small window and there we would select the terminal tab.**

Now lets discuss some git commands:

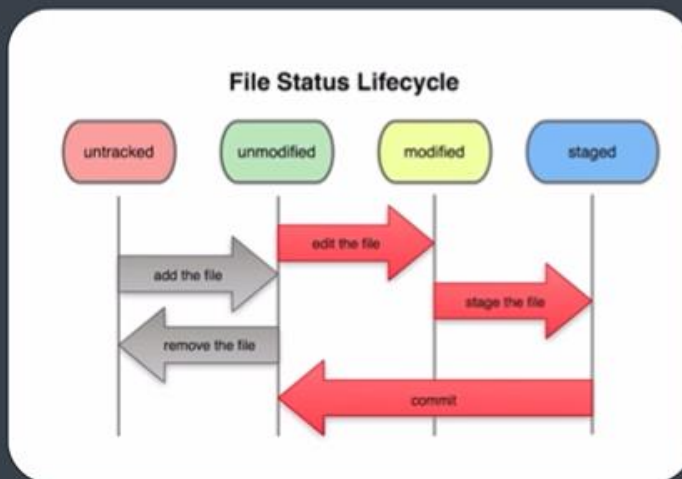
## Basic Commands

- **Clone** - Cloning a repository on our local machine

`git clone <- some link ->`

- **status** - displays the state of the code

`git status`

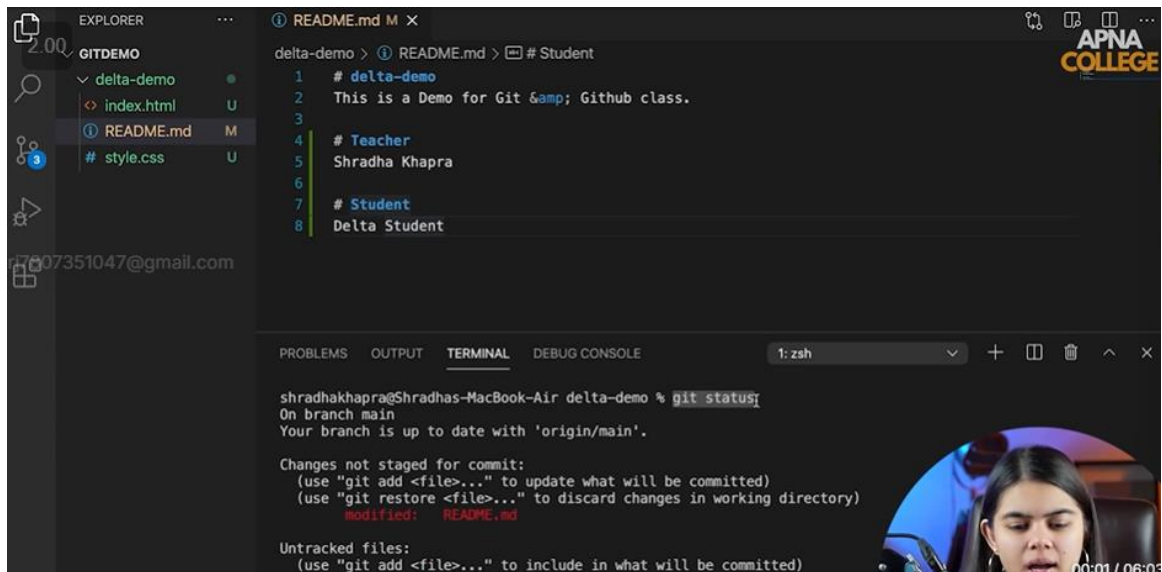


- **The clone command is used to create a copy of a repository on our local machine. In simple words it means to download and create a copy of that repository on our machine.**
- **The status command show us the status of our code, that is whensoever we make any change in our code. The status of code gets changed every-time.**

Now here also we have various status:

- **When we create a new file, it goes in the untracked status and we need to add them to git.**
- **Now in already created files if changes not made, there status is marked as unmodified.**
- **If changes made they are marked as modified.**
- **And if we commit to our changes then that file is marked as staged so that it is tracked.**

For instance say we created new files named index.html and style.css using terminal. And wrote some content in README.md



```
delta-demo > README.md # Student
1 # delta-demo
2 This is a Demo for Git & Github class.
3
4 # Teacher
5 Shradha Khapra
6
7 # Student
8 Delta Student

shradhakhapra@Shradha-MacBook-Air delta-demo % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

Now when we run the “git status command”,

We can see that it shows changes that hasn’t been committed yet , like when we added extra content in README.md file therefore its status is shown modified ,whereas the new files that we created came under the untracked files heading,

Now to commit those changes we need to follow a series of steps,

### Basic Commands

- **add** - adds new or changed files in your working directory to the Git staging area. `git add <- file name ->`
- **commit** - it is the record of change  
`git commit -m "some message"`
- **push** - upload local repo content to remote repo  
`git push origin main`

Like firstly we use “add command” on modified files, and there status is changed “to staged”. And after that we finally use “commit command” to finalise that change.





Like if we want to add index.html file into our repository,

```
shradhakhapra@Shradhas-MacBook-Air delta-demo % git add index.html
shradhakhapra@Shradhas-MacBook-Air delta-demo % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    style.css
```

We see it is showing index.html file as staged and to finalise it , we now just need to perform the commit command on it.

**Now with this way we can enter files one by one, but some times we have a lot of files and it may take a lot of time, therefore to avoid that we use: "git add ."**

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: zsh
shradhakhapra@Shradhas-MacBook-Air delta-demo % git add .
shradhakhapra@Shradhas-MacBook-Air delta-demo % clear
```

And now if we run the git status command

```
shradhakhapra@Shradhas-MacBook-Air delta-demo % git add .
shradhakhapra@Shradhas-MacBook-Air delta-demo % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   README.md
    new file:   index.html
    new file:   style.css

shradhakhapra@Shradhas-MacBook-Air delta-demo %
```

**And after that we finally commit. Now commit is used after we have made a bunch of important changes. And with commit we also need to mention an important message telling about the changes we made.**

```
shradhakhapra@Shradhas-MacBook-Air delta-demo % git commit -m "Add new files"
```

But now if we run the “git status”, we would see

```
shradhakhapra@Shradhas-MacBook-Air delta-demo % git commit -m "Add new files"
[main 29678ff] Add new files
3 files changed, 18 insertions(+)
create mode 100644 index.html
create mode 100644 style.css

shradhakhapra@Shradhas-MacBook-Air delta-demo % git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Now here if we notice it is showing our branch is one step ahead of ‘origin/main’ by 1 commit, it means we have made one extra commit then our original repository on github.

**Now to push our changes to github also, we would use the “push” command.**

- **push** - upload local repo content to remote repo

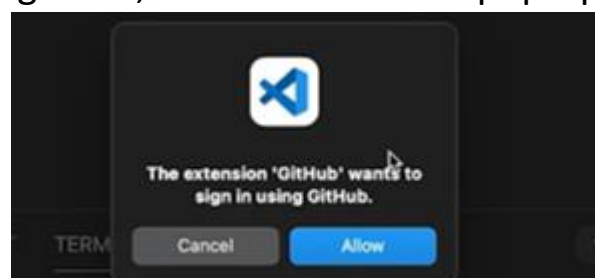
```
git push origin main
```

Here origin refers to the repository from where we cloned to our local machine.

And here “main” is the name of our branch.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  1: zsh
shradhakhapra@Shradhas-MacBook-Air delta-demo % git push origin main
```

And after pressing enter, we would see this pop-up



Now say if we want to upload our local project from our machine to the github and create a new github repository, using terminal . then we do it using the “git init” command while being inside that folder we want to add on github as an repository.

- **init** - used to create a new git repo *git init*

*git remote add origin <- link ->*

*git remote -v* (to verify remote)

*git branch* (to check branch)

*git branch -M main* (to rename branch)

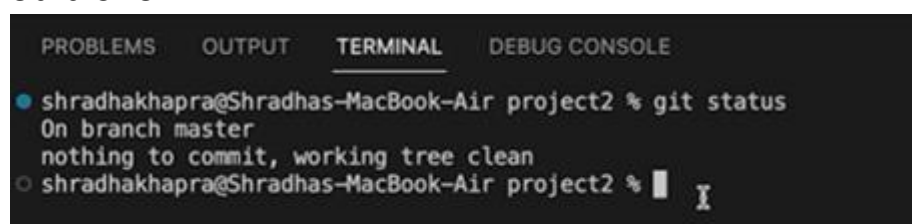
*git push origin main*

And now after that our project would be added as an repository on github. Then we can simply create multiple files. And add some data into them.

- And to add them into staged mode so as to commit we type: “git add .” and it would add all the files into staged mode.
- After that we would use the commit command to “finalise the changes”.

“git commit -m “added data and created a new README.md file” ”

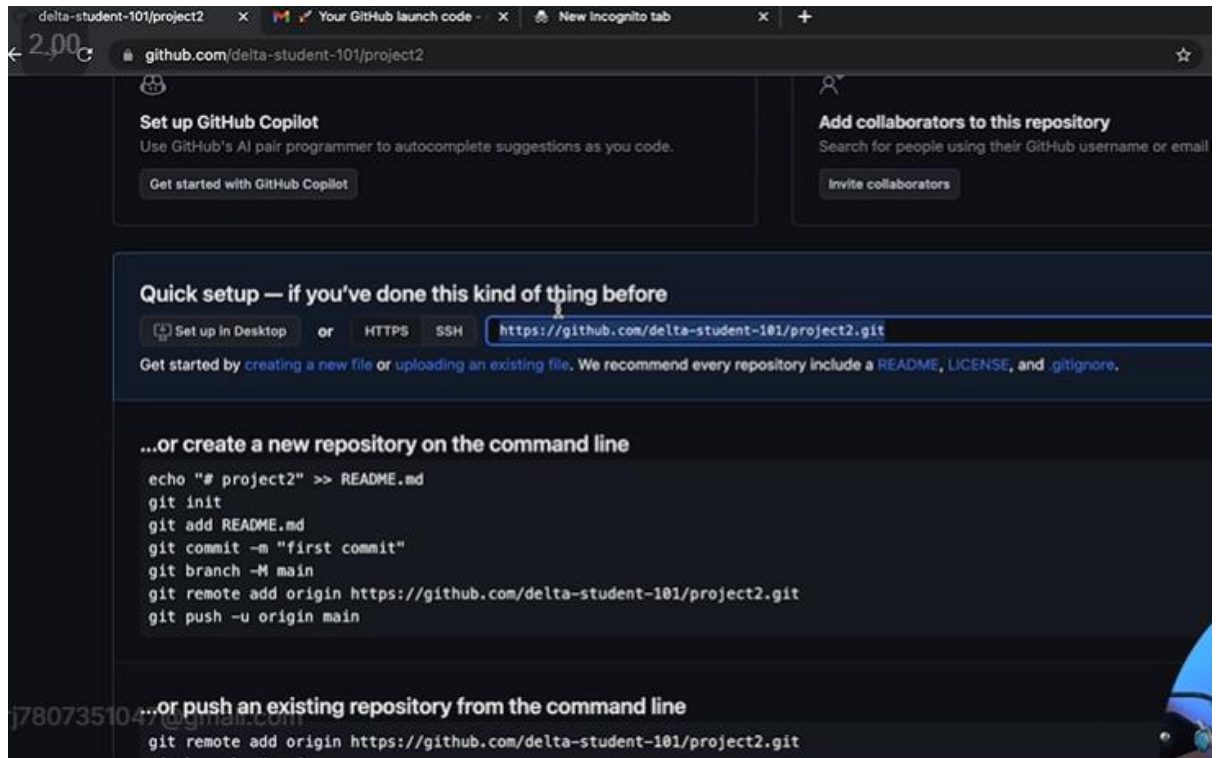
And it would show:



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
shradhakhapra@Shradhas-MacBook-Air project2 % git status
On branch master
nothing to commit, working tree clean
shradhakhapra@Shradhas-MacBook-Air project2 %
```

And then finally we would also like to push our project2 folder as a repository on github and for that we need to perform following steps:

- Firstly open github go to profile and the repositories and then create a new repository. And after that when we scroll there we can see a lot options to push our code onto that new repository, but we would copy the link of our repository from there:



- Then after copying that link, we run this command:

```
git remote add origin <- link ->
```

And would paste our link here. This basically signifies that, we are setting that repository as our remote and are naming it as "origin".

```
shradhakhapra@Shradhas-MacBook-Air project2 % git remote add origin https://github.com/delta-student-101/project2.git
shradhakhapra@Shradhas-MacBook-Air project2 %
```

- And to verify it we can write,

```
shradhakhapra@Shradhas-MacBook-Air project2 % git remote -v
origin https://github.com/delta-student-101/project2.git (fetch)
origin https://github.com/delta-student-101/project2.git (push)
shradhakhapra@Shradhas-MacBook-Air project2 %
```

- And it tells us now, our current remote that is "student name and project name" which means now our online repo is linked with our project.

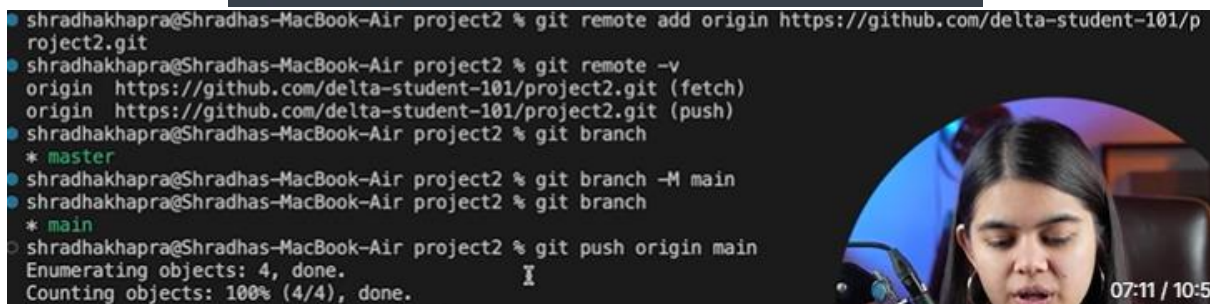
- After that we can run `git branch` to check our branch, but it would show master, and as it is a little derogatory therefore we would update that name to “main”

`git branch` (to check branch)

`git branch -M main` (to rename branch)

- And now to finally push our project, then we would write `git push origin main`, which mean to push our project folder onto the origin's (the link that we set as remote to link our project folder with the github repository that we created) main branch.

`git push origin main`



```

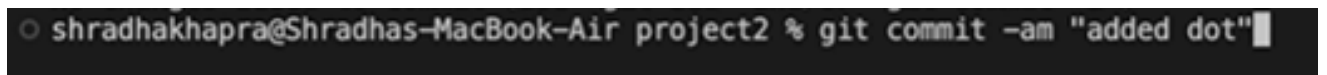
shradhakhapra@Shradhas-MacBook-Air project2 % git remote add origin https://github.com/delta-student-101/project2.git
shradhakhapra@Shradhas-MacBook-Air project2 % git remote -v
origin https://github.com/delta-student-101/project2.git (fetch)
origin https://github.com/delta-student-101/project2.git (push)
shradhakhapra@Shradhas-MacBook-Air project2 % git branch
* master
shradhakhapra@Shradhas-MacBook-Air project2 % git branch -M main
shradhakhapra@Shradhas-MacBook-Air project2 % git branch
* main
shradhakhapra@Shradhas-MacBook-Air project2 % git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
  
```

Now to avoid writing origin main again and again, we can set upstream as “origin main” by passing this command one time

`git push -u origin main`

After that when soever we want to push some code, we can simply write “`git push`”, instead of writing the whole syntax of command.

And when we have a single file, to which we made the changes :  
**“then we can commit and add in one line”**



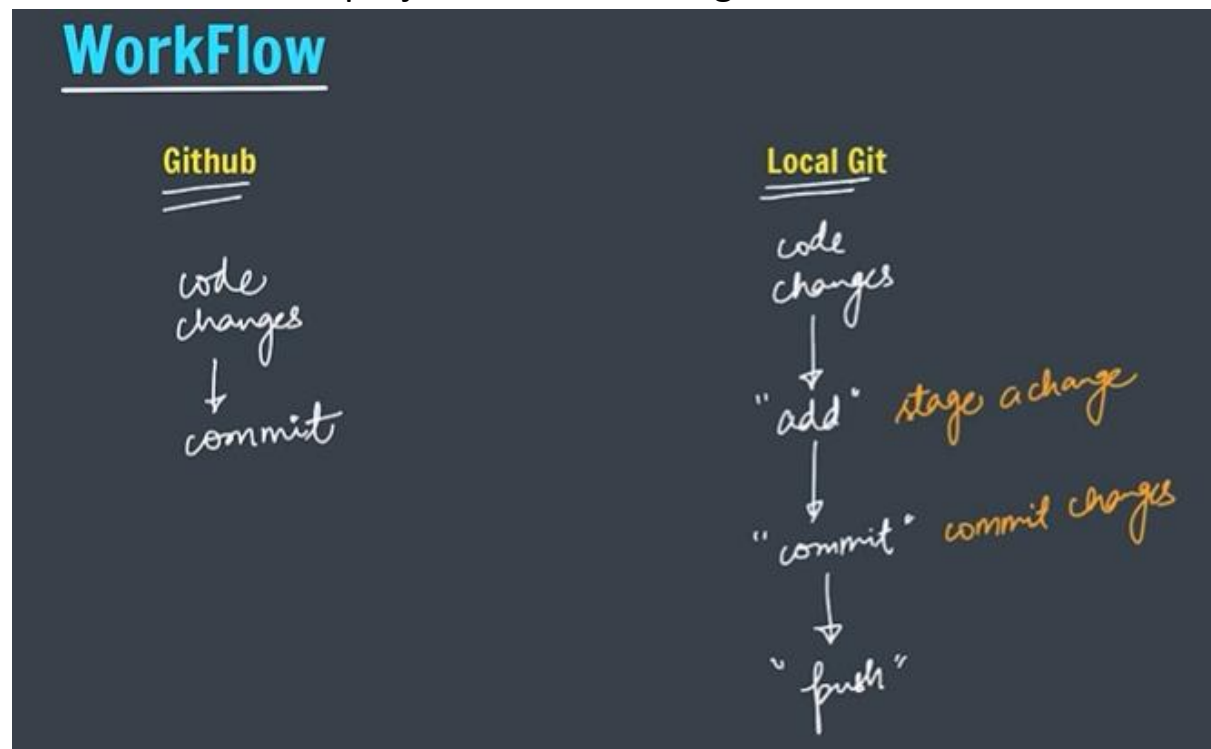
```

shradhakhapra@Shradhas-MacBook-Air project2 % git commit -am "added dot"
  
```

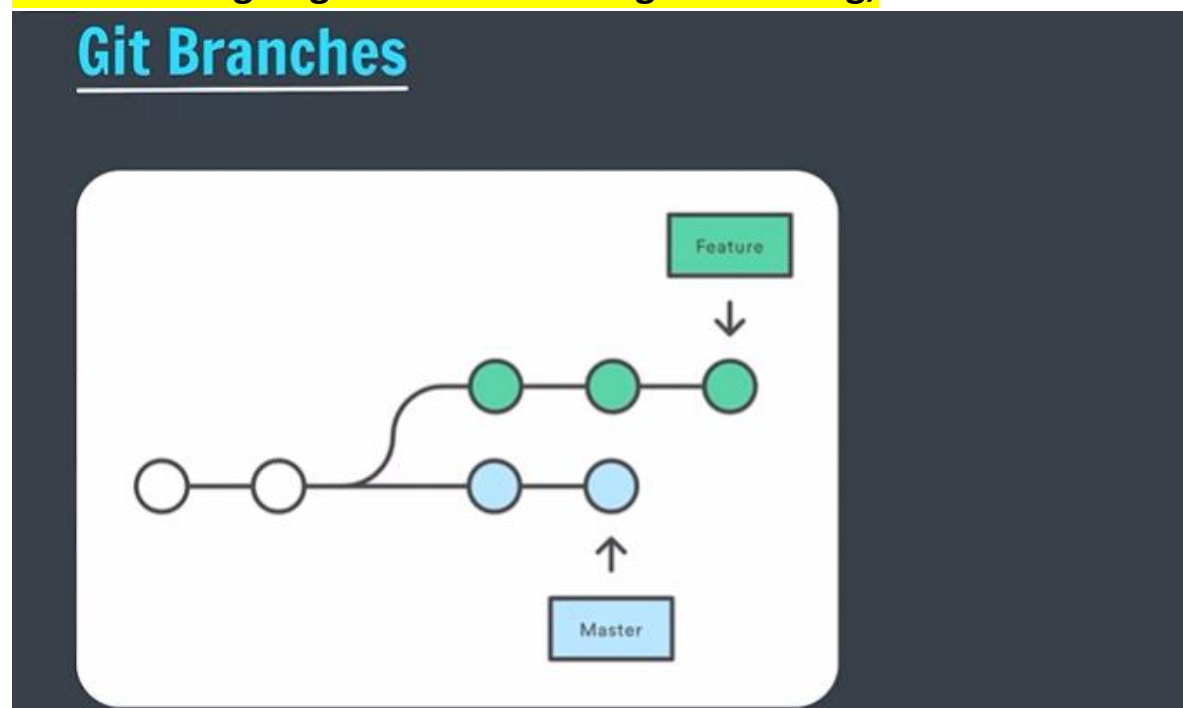
Here a stands for adding and m stands for message, therefore -am.



So a short summary of actions that we perform on github and when we work on a local project connected to github are:



Now we are going to discuss about git branching,



Here we basically create a copy of our original code and try and make changes into that, without affecting the original code, and if the features sits right, then we can merge it with the original code.

# Branch Commands

`git branch` (to check branch)

`git branch -M main` (to rename branch)

`git checkout <- branch name ->` (to navigate)

`git checkout -b <- new branch name ->` (to create new branch)

`git branch -d <- branch name ->` (to delete branch)

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  [x] zsh -
```

```
● shradhakhapra@Shradhas-MacBook-Air project2 % git branch
* main
● shradhakhapra@Shradhas-MacBook-Air project2 % git checkout -b feature
Switched to a new branch 'feature'
● shradhakhapra@Shradhas-MacBook-Air project2 % git branch
* feature
  main
● shradhakhapra@Shradhas-MacBook-Air project2 % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● shradhakhapra@Shradhas-MacBook-Air project2 % git branch
  feature
* main
● shradhakhapra@Shradhas-MacBook-Air project2 % git checkout feature
Switched to branch 'feature'
● shradhakhapra@Shradhas-MacBook-Air project2 % █
```

```
● shradhakhapra@Shradhas-MacBook-Air project2 % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● shradhakhapra@Shradhas-MacBook-Air project2 % git branch
  feature
* main
  test
● shradhakhapra@Shradhas-MacBook-Air project2 % git branch -d test
Deleted branch test (was e2c733c).
● shradhakhapra@Shradhas-MacBook-Air project2 % git branch
  feature
* main
● shradhakhapra@Shradhas-MacBook-Air project2 % █
```

But now say we want to push our code of the branch that we created, now for that we need to set the upstream once again

```
shradhakhapra@Shradhas-MacBook-Air project2 % git checkout feature
Switched to branch 'feature'
shradhakhapra@Shradhas-MacBook-Air project2 % git push
fatal: The current branch feature has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin feature

shradhakhapra@Shradhas-MacBook-Air project2 % git push --set-upstream origin feature
Enumerating objects: 4, done.
Compressing objects: 100% (4/4), done.
```

Note: here feature is the name of the branch.

Now after this if we want to merge the branch with our code,

## Merging Code

`git diff <- branch name->` (to compare commits, branches, files & more)

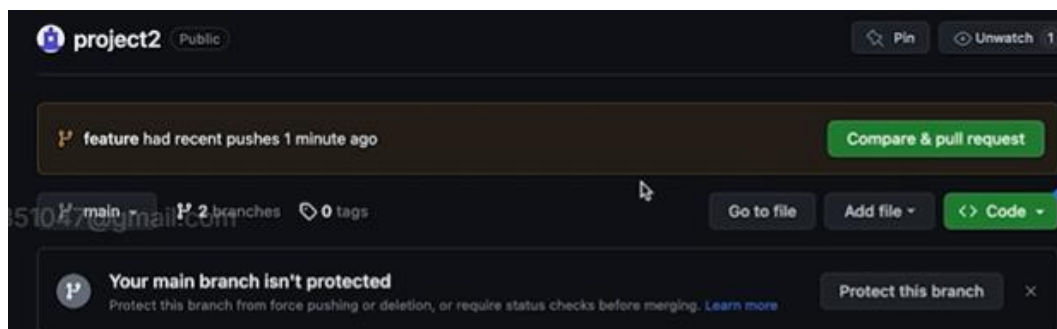
`git merge <- branch name->` (to merge 2 branches)

OR

Create a PR

In this way we compare the two branches using “git diff”, that is to know how much a branch is ahead of main branch or vice versa. And then we use the merge command so as to merge the mentioned branch with the main branch.

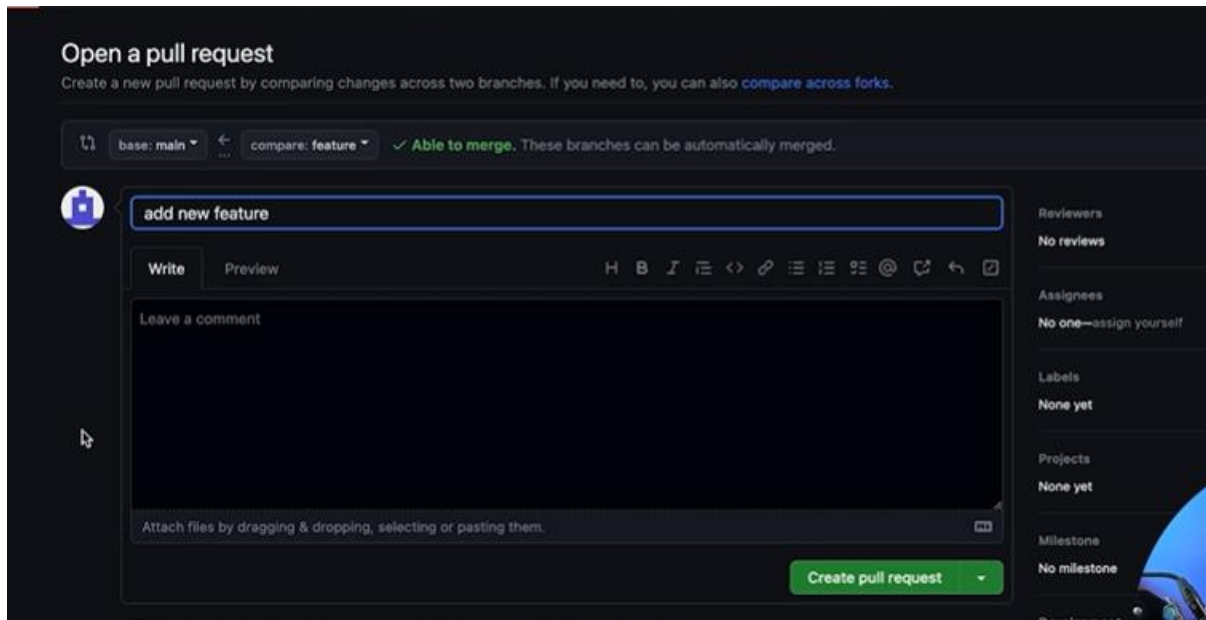
Another way of merging two branches is using a pull request (PR), When so ever we want to merge two or more branches and we have multiple branches , the github shows a button of “compare and pull request”



## Pull Request

It lets you tell others about changes you've pushed to a branch in a repository on GitHub.

And when we click on “compare and pull request button” we get:

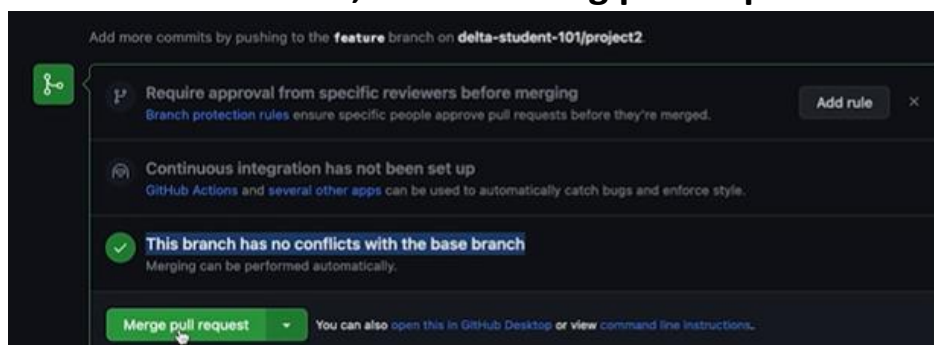


Here we can see the two branch name we want to merge, that is as the arrows show in above image, we are trying to merge the feature branch with the main branch.

And then we can write a message , like what we are adding with the merge.

And we can also leave a description comment telling every detail.

**Now there we would see, after creating pull request:**



Now here it is mentioned no conflicts, a conflict is arised when same changes are made in main as well as branch file, as then github don't know with which features to go with.

After that we can simply click on merge to merge the two branches.

After merge we can now delete our feature branch.

But now if we would open our VScode, we would notice the both branches haven't merged yet,

**Therefore to pull the changes of merging from github to local machine we use the “pull” command:**

## Basic Commands

```
git pull origin main
```

used to fetch and download content from a remote repo and immediately update the local repo to match that content.

rj7807351047@gmail.c

Just like , we use “push” when we want to push our changes from our local machine to the github, similarly when we want to bring the changes from the github to our local machine we use the “pull” command.

And this command would pull all the changes from our origin , we would pull all the changes to our main branch.

**But for the scenario where some conflicts occur,**

## Merge Conflicts

An event that takes place when Git is unable to automatically resolve differences in code between two commits.

```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
3 <<<<<< HEAD (Current Change)
4 Adding a new change-feature
5 =====
6 Adding a new change-main
7 >>>>>> main (Incoming Change)
8
```

That is when in two branches we make the changes at the same spot, the git gets confused, that is to go with which change. Which give rise to conflicts.

VS code gives a lot of features, that is like to choose between what feature we want to go with, or we can directly make the changes and choose the results we want after the merge.

But after that we again need to “add” that file and as merge itself is a commit in itself therefore we can directly merge now.



## Fixing Mistakes

Case 1 : staged changes

```
git reset <- file name ->
```

```
git reset
```

If we have added some unnecessary features and we want to unadd them, then we use the reset command which would reset all staged changes.

Now if we want to reset changes of a single file, then we can mention the file name, otherwise if we want to reset all staged changes then we would simply run `git reset`.

Case 2 : committed changes (for one commit)

```
git reset HEAD~1
```

If we committed to some wrong feature, then we use this command, here head refer to the last change that we committed. Here 1 refers to, that we want to move 1 step behind commit.

Case 3 : committed changes (for many commits)

```
git reset <- commit hash ->
```

```
git reset --hard <- commit hash ->
```

For this we can run `git logs`, it shows the hash code for all the commits we have performed upto now, and the point to which we want to reset we copy that hash code and paste it with the `git reset` command and it undo's all commit.

But after that , if we want ki reset hone ke baad we also don't want to see all the changes that occurred with the previous commit, the we pass the command with –

## What is Forking?

A fork is a new repository that shares code and visibility settings with the original "upstream" repository.

Fork is a rough copy.

That is if we want to make changes with repository created by some other person, then we can fork it,

It would create a copy of that repository into our own account.

Now say we made the necessary changes, then we can now create a pull request with original repository owner telling about the changes we made.

And we can simply create a pull request for it.